DISSERTATION

HUMAN-GUIDED, AI-ACCELERATED SYSTEM DYNAMICS VIA PIPELINE ALGEBRA

Submitted by

Kirk Reinholtz

Department of Systems Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2025

Doctoral Committee:

Advisor: Kamran Eftekhari Shahroudi

Patricia Aloise-Young
Steve Simske
Wade Troxell

ABSTRACT

HUMAN-GUIDED, AI-ACCELERATED SYSTEM DYNAMICS VIA PIPELINE ALGEBRA

System Dynamics (SD) gives Systems Engineers (SEs) and Model-based Systems Thinking (MBST) practitioners in general a rigorous way to reason about feedback-rich problems, yet adoption remains low because assembling a causal-loop diagram (CLD), validating its logic, and converting it into an executable model are still labor-intensive and require specialized skills. Human analysts remain central to judgment, but they should spend their time on insight rather than tool wrangling. This dissertation demonstrates that chat-based large language model (LLM) pipelines can remove that bottleneck, automating polarity-reversal checks, loop-dominance mapping, latent embeddings that capture joint structure-behavior signatures for similarity search, missing-loop discovery, and first-cut model synthesis, thereby lowering the entry barrier for SEs and cutting typical SD turnaround from days to minutes.

Powered by ChatGPT o3, a transformer pre-trained on internet-scale corpora of prose, source code, and mathematical notation, a single interactive session can (i) read narrative text, (ii) propose syntactically complete CLDs, (iii) diagnose structural anomalies, and (iv) translate diagrams into executable SD code. The pipeline then runs an LLM-generated simulator and, through chain-of-thought prompting, iteratively tunes loop structure and parameters until simulated behavior reproduces a reference mode that the same LLM mined from prose and refined via targeted web search. These feats rest on three enablers: cross-modal pattern learning that maps language to graph and code representations; chain-of-thought prompts that force the model to externalize intermediate reasoning; and an agentic, simulator-in-the-loop refinement cycle that tests and revises its own drafts. The full loop finishes in minutes, far faster than manual workflows, and while domain judgment is still

decisive at checkpoints, no specialized SD software expertise is required.

Three studies validate the approach. First, the pipeline outperformed forty-three graduate students and matched an instructor benchmark when extracting Janis's *groupthink* CLD, finishing each run in under fifteen minutes. Second, it converted qualitative CLDs into quantitative executable SD model simulators in minutes through expert-in-the-loop refinement. Third, symbolic routines with no LLM involvement computed generalized loop sets for 678 models and clustered 59K equations from a 1K-model corpus; by clarifying how feedback structures share influence across behavior modes, loop sets provide SEs a principled aid to loop dominance comprehension and are slated for integration with the LLM toolkit.

Every transformation, whether an LLM invocation, a symbolic routine, or a shell command, is expressed in Pipeline Algebra (PA), a typed workflow language that serves as an executable notation for thought, records explicit pre- and post-conditions, supports deterministic replay, and aligns naturally with meta-algorithmic control. Forthcoming primitive operators, exposed as terminal functions through the OpenAI functions API, will let GPT itself invoke loop-set analytics, polarity-reversal detection, and higher-order transformations such as `map` and `comap`, unifying them within the same declarative fabric and steering control from bespoke orchestration code toward the language model, thereby laying the groundwork for self-optimizing model laboratories that combine formal mathematics with AI-guided pattern discovery.

By combining LLM automation with a rigorous workflow backbone, the approach lets SEs exploit SD with far less overhead and far greater throughput. By shifting the analytic burden from tool wrestling to insight building, it invites a wider pool of engineers and decision makers to deploy feedback-based modeling, an advance that can sharpen climate-mitigation policy and other responses to feedback-driven challenges.

ACKNOWLEDGEMENTS

Thank you, Prof. Kamran Eftekhari Shahroudi, my advisor, for bringing me into the program, instilling in me the ways of System Dynamics and Systems Thinking, and providing the many gentle nudges that kept me on track technically and personally through this journey.

Thank you, Prof. Patricia Aloise-Young, Prof. Steve Simske, and Prof. Wade Troxell, my committee members, for guiding my transformation from engineer to academic.

Thank you, Stephen Wolfram, for your invention of Wolfram Mathematica, which gives me power to put my thought to notation and without which this work would have been vastly more difficult and exhausting.

Thank you, Ras, for helping me discover—and then sharpen—how I think.

Thank you, Guillermo, Giovanna, Ashley, and Leith, for helping me maintain my sanity through dance, movement, and introspection.

Thank you, Kelli, for showing me this is possible.

And most of all, I thank Virginia for her infinite patience and more than a few trajectory-altering and out-of-the-box conversations about this work.

TABLE OF CONTENTS

1 Introduction

1.1 Lay Summary

A highway lesson in hiding–in–plain–sight feedbacks

Los Angeles poured US$ 1.6 billion into widening Interstate 405 so drivers could "erase" rush-hour traffic. It worked—for about a week. Five years on, the average afternoon trip through the Sepulveda Pass was *slower* than before the lanes opened, a textbook case of *induced demand*: smoother flow lowers the "price" of driving, which lures extra trips until the new pavement clogs again.(Chiland, 2019)

The same boomerang shows up across domains. Raising levees reassures homeowners and spurs riverside building that magnifies the next flood; over-prescribing antibiotics brings quick relief today, but breeds drug-resistant bacteria that are far harder and costlier to treat tomorrow; cheap public borrowing juices growth now, yet saddles future budgets with interest payments that crowd out schools and infrastructure. Each "fix" hides a feedback loop that later bites back—exactly the blind spot systems thinking seeks to reveal.(Cecchetti et al., 2011; Collenteur et al., 2015; Llor & Bjerrum, 2014)

Climate change is the highway story at planetary scale. Burning subsidized fossil fuels delivers instant light, heat, and profit-global support for coal, oil, and gas topped US$ 7 trillion in 2022 (counting both direct and implicit subsidies)-while the true cost arrives decades later as stronger storms, rising seas, and mitigation bills that soar when action is deferred. Near-term revenues create powerful constituencies for delay, just as construction dollars help sell new freeway lanes.(Black et al., 2023; *Climate Change 2023: Synthesis report. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, 2023)

This dissertation takes the first practical step toward letting far more people spot such *loops before the concrete is poured or the carbon is burned.* It shows how a napkin-sketch cause-and-effect map, or even less, can become a working computer experiment in one afternoon and prototypes a smart, chat-style AI guide that walks users through each step in plain English. By compressing months of expert labor into hours, the work opens systems thinking to a much broader circle-seasoned systems engineers, domain specialists, and *citizens* alike. While a full one-click tool remains future work, the results here demonstrate the core idea and chart a clear path toward citizens, policy teams, and engineering groups testing quick fixes virtually, making evidence-based choices, and meeting their duty to steward the shared systems on which we all depend.

Methods such as System Dynamics can diagnose these long-tail behaviors, but they currently demand substantial specialized training and are therefore often skipped. This work shows that months of modeling effort can be compressed into hours and require much less modeling expertise by working with a chat-style AI that can both reason and invoke domain-specific tools using a particular vocabulary well suited to both human and AI comprehension, raising the central *Research Problem* of the dissertation: *Can automating key steps in model construction and comprehension broaden the community that can apply System Dynamics effectively and allied methods effectively?*

1.2 Thesis, map to chapters

Our central thesis is that we can answer our Research Problem (RP) in the affirmative: *Can can broaden the base of systems engineers and Model-Based Systems Thinking (chapter 3) (MBST) practitioners who can effectively apply system dynamics (SD) by automating key aspects of model construction and analysis?* Yes, we can. We investigate this thesis through five research questions well known as SD friction points, all in the context of highly automating each so that they are more easily and broadly performed, perhaps even more effectively performed:

(RQ1) *Can all conditions under which polarity reversals occur be automatically identified?*

(RQ2) *Can forms of loop dominance be automatically detected and characterized?*

(RQ3) *Can we determine both (a) a compact basis set that captures the reference modes of behavior, and (b) a compact structural representation of the model?*

(RQ4) *Can we identify missing feedback loops that are likely to produce unintended consequences?*

(RQ5) *Can we synthesize plausible models that exhibit a desired behavior directly from a behavioral specification?*

If these research questions can be affirmatively addressed, we can automate a set of high-effort tasks that currently require deep familiarity with the specialized tooling, terminology, and reasoning methods of the system dynamics community. Importantly, these demands are orthogonal to the domain-specific knowledge that systems engineers already possess about their system-of-interest (SOI). As a result, the manual application of system dynamics imposes a significant cognitive and technical burden. By automating these arcane, but essential, modeling steps, we increase leverage for domain experts, reduce barriers to adoption, and enable broader, more scalable use of system dynamics methods in real-world engineering contexts.

The major contributions of this dissertation recur across multiple chapters, often in different methodological and empirical contexts. The Contribution-to-section map (Table 1.1) provides a quick-reference map that links each contribution to the sections where it is developed or applied.

Table 1.1: Major contributions and where in the dissertation they are developed or applied.

Contribution	Key evidence locations
Cross-cutting Models of Knowledge	
Morphological Pipeline Algebra (PA)	Morphological Pipeline Algebra (MPA, PA) (§ 2); P43 Janis for IS 2025 Revised (§ 12); LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra (§ 13); Missing Loops (§ 14); Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (§ 15);
Attributed Directed Graph (ADG)	Attributed Directed Graph (ADG) (§ 7);
Falsifiable Prompting Protocol	Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (§ 15)
Research Questions	
RQ 1 Ambiguous Polarity	Analytical Conditions of Loop Polarity Reversal (§ 9)
RQ 2 Dominant loops	Generalized Loop Sets (§ 11)
RQ 3a Behavior Embeddings	SYSE532 guest lecture nov2023
RQ 3b Structure Embeddings	AI Outperforms 43 grads developing CLD of Janis Groupthink (§ 12), LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra (§ 13)
RQ 4 Missing Loops	Missing Loops (§ 14)
RQ 5 Behavior to Structure	Behavior to Structure (§ 16)

1.3 Summary of Dissertation

Have I answered my Research Problem(RP) in a methodological, systematic, and epistemic fashion? I claim I have, evidenced by the following, and further, I claim that the the answer to the RP itself is the affirmative.

I speculatively answered the RP in my prelim, some 16 months ago: ".. Note LLM and general AI tech and available models is evolving at extremely high rate, and I'm speculating on my ability to use them for this application. Or maybe I'll be able to buy and train one from OpenAI over a coffee break. Who knows...."

In A Pragmatic Working Model of Agentic Large-Language Models (chapter 4) we define the basic machinery of the agentic LLM. Of particular note is its ability to decompose a problem into a number of steps, and direct itself to perform those steps while also adjusting the steps as required based upon the progress toward the goal. This is the "agentic" and "chain of thought" mechanisms in action.

We developed Pipeline Algebra (chapter 2) as a formal language with which to interact with the agentic and CoT aspects of the LLM, so that the CoT can be exchanged between the LLM and the software executing the PA pipeline. PA is designed to be easily comprehended by human, LLM, and algorithmic software. PA thus forms a repeatable, methodical and epistemic representation of the method used to perform a task. The formal nature of PA makes it highly manipulable, potentially facilitating meta-algorithmic methods to be performed by and applied to the LLM CoT.

We use PA and the LLM to accelerate the cycles in Model-Based Systems Thinking (chapter 3) and to shift much of the specialized activities from the human to the LLM, thereby reducing the need for the systems engineer to have much specialized knowledge and experience. This is the mechanism by which we answered the RP in the affirmative.

In Missing Loops (chapter 14) we apply PA and LLM to the problem of identifying policy loops that are missing and consequently make the policy vulnerable to subversion. Here we find evidence that the LLM, perhaps via its token embedding into a high-dimensional latent manifold and multi-head self-attention layers that can selectively amplify remote regions of its knowledge vector space and so fuse disparate semantic clusters into novel continuations (AKA "alien thinking"), can perform behavior analysis heretofore thought to require numerical methods.

Useful new technical model analysis tools are presented in Analytical Conditions of Loop Polarity Reversal (chapter 9), where we present a novel aid to the long-standing challenge of developing a well-formed model, which is defined to include the absence of loop reversals; and Generalized Loop Sets (chapter 11), where we add a new novel method of loop dominance analysis, which is a key method of gaining insights into the behavior of a model. In Equations of the corpus (chapter 10) we analyze the patterns of mathematical relationships used in a large corpus of existing model, which (future work) will accelerate model construction via evolutionary optimization.

And finally in Behavior to Structure (chapter 16), we support our claim that the RP has been answered in the affirmative. In this section we apply our methods to a typical SD task that would be a challenging semester project even in a graduate SD course. The objective is to determine the causes of a large programmatic failure by analyzing the failure report, developing an SD model using standard and novel structure motifs, then refining the SD model structure to better align with the reference mode behavior in the report and otherwise available from public sources. The result is evidence of two plausible but unreported management behaviors. The task took about 30 wall-clock minutes and 20 AI compute minutes to perform. We applied MBST, CLD to SD translation, ADG, Falsifiable Prompting, PA with COT and agentic AI. Very little knowledge in systems dynamics was required. Domain knowledge is advisable in order interact effectively with the agentic AI.

2 Pipeline Algebra

2.1 Summary

Pipeline Algebra (PA) is a workflow language grounded in category theory. Every stage is a *typed morphism* $f : A \to B$ whose domain A and codomain B are explicit, making the specification statically typed and machine–checkable (Spivak, 2014).

Much of our work can be summarized in PA as shown in Types, morphisms, mechanisms (Figure 2.1).

State–Passing Core

PA adopts a *state–passing* discipline: each operator receives the current logical state, returns an updated state, and never mutates hidden globals (Moggi, 1991). Because operators are pure and idempotent (Hughes, 1989), the same logical inputs always yield the same logical outputs, enabling memoization (Michie, 1968), deterministic re-runs, and straightforward parallel or distributed execution.

Functorial Semantics

A semantics functor

$$[\![-]\!] : \mathbf{P} \longrightarrow \mathbf{Exec}$$

maps the abstract category $\mathbf{P}$ of operators to an executable DAG $\mathbf{Exec}$, preserving identity and composition. Equational laws proved in $\mathbf{P}$—fusion, reassociation, dead-code elimination—therefore hold after compilation (Mac Lane, 1998). Real-world effects (API calls, randomness, I/O) are reified in the threaded state by interpreting each implementation in the Kleisli category of a state monad, so the algebra remains *semantically pure*: identical logical inputs produce identical logical outputs, regardless of hidden work.

Higher-Order Operators

PA provides higher-order combinators such as `Map`, `CoMap`, `Reduce`, `Branch`, and `Merge`. These lift ordinary morphisms to collections, conditional paths, and scatter–gather patterns, supporting sophisticated meta-algorithmic orchestration (Simske, 2013) without leaving the algebraic notation.

Practical Consequences

- **Reproducibility & Auditability** — Memoization plus explicit state create a deterministic audit trail, crucial for scientific replication and compliance (Peng, 2011).
- **Parallel & Cloud-Scale Execution** — Purity and explicit dependencies allow direct compilation to schedulers such as Airflow or SageMaker, leveraging DAG execution models (Lee & Parks, 1995).
- **Epistemic Artifact** — One PA file is simultaneously an executable specification and a human–readable explanation ("notation as thought" (Iverson, 1980)), supporting make-like incremental builds (Feldman, 1979).

In sum, PA combines categorical rigor with practical orchestration tools, delivering a coherent, optimizable, and reproducible foundation for modern workflows that blend deterministic code, numerical computation, and large-language-model services. The terminal functions are domain specific with semantics typically apparent from the operator name.

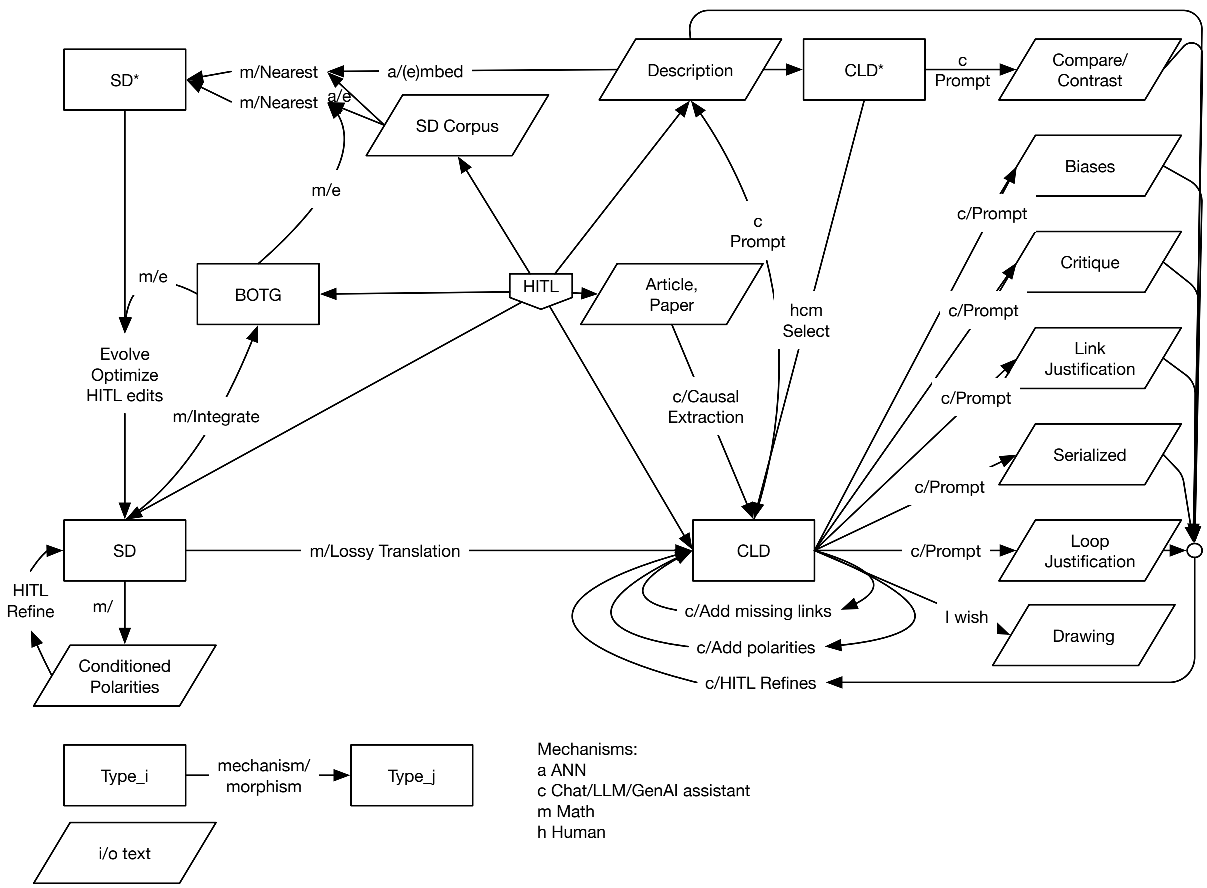


Figure 2.1: Much of our work is described and implemented as morphisms of types. Here we present the major morphisms. Types are usually Associative Arrays or lists thereof. Morphisms add KV pairs, delete keys, or mutate values. Most of the morphisms are further described herein using our Pipeline Algebra. The implementation closely follows the notation. Most morphisms are stateless idempotent operators on AA's or compositions (including high level functions e.g. map, comap, composition, lambdas, curry, meta-algorithmic operators, error handling) thereof. Operators that interact with stochastic services e.g. ChatGPT or Gemini may memoize results so as to provide apparent idempotence.

2.2 Core Concepts

Core Concepts

Morphisms as the atomic unit. Morphological Pipeline Algebra (MPA) casts every transformation as a *typed morphism* $f : A \to B$. The idea descends from the "objects and arrows" vocabulary of category theory (Spivak, 2014) and its popularization for programmers (Milewski, 2019). Because composition $g \circ f$ is defined only when $\mathrm{cod}(f) = \mathrm{dom}(g)$, a pipeline is simultaneously an executable artifact *and* a machine-checkable proof that intermediate objects are well formed.

State threading and purity. Each morphism receives and returns a single associative array that threads right-to-left (the *Kleisli-monad* pattern) so operators remain *pure and idempotent*: identical inputs always reproduce identical outputs. Any stochastic sub-routine must write its seed into the threaded state, preserving determinism. This discipline enables whole-pipeline memoization (aka stored completions) and embarrassingly parallel execution (Moggi, 1991).

Two complementary surfaces. MPA exposes a *math-ish* arrow calculus for equational reasoning and a *code-ish* binding layer that attaches those arrows to concrete executables (e.g. Python functions, Docker containers). Equations such as

$$\mathrm{Map}\, f \;\circ\; \mathrm{Map}\, g \;=\; \mathrm{Map}\,(f \circ g)$$

hold both as algebraic identities and as runnable code, realizing Iverson's dictum that "notation is thought" (Iverson, 1980).

Functorial semantics and DAG compilation. A semantics functor $[\![-]\!] : \mathbf{P} \to \mathbf{Exec}$ maps morphisms in the syntactic category $\mathbf{P}$ onto nodes in an executable DAG $\mathbf{Exec}$, preserving identities and composition. Today the functor drives MPA's native interpreter; tomorrow the same mapping can emit Beam- or Step-Functions-compatible graphs for distributed runs (Amazon Web, n.d.; The Apache Software, n.d.). Because $[\![-]\!]$ is structure-preserving, algebraic rewrites (fusion, dead-code elimination) lift automatically to the run-time graph.

DAG execution in practice. Directed-acyclic-graph schedulers dominate large-scale data and ML workflows. **Apache Beam** lets users wire transforms together in *Python* (or Java/Go), while **AWS Step Functions** expresses control flow in the JSON/YAML-based Amazon States Language (Amazon Web, n.d.; The Apache Software, n.d.). Both platforms are proudly *code-ish*: authors hand-craft resource hints and container images. MPA provides the complementary *math-ish* layer: a few compositional equations yield the same DAG *by construction*, which the functor then exports to Beam or Step Functions for cloud-scale scheduling, all while preserving typing, purity, and equational correctness.

Domain packages out of the box. Although the calculus itself is domain-agnostic, MPA ships with ready-made types (`SDModel`, `CLD`, ...) and assertions (`issd`, `iscld`) encoding long-standing System Dynamics conventions (J. W. Forrester, 1961; J. Sterman, 2000). Researchers in other fields can load their own type libraries without touching the kernel.

LLM calls as first-class morphisms. Every prompt to a large-language model is wrapped as a morphism whose post-condition specifies what must hold in the returned state (e.g. "slot `V` now contains a non-empty variable list"). Violations abort the run, turning brittle "prompt-and-pray" scripts into falsifiable, typed program steps.

Self-contained yet open. MPA includes a lightweight interpreter, so exploratory pipelines run instantly on a laptop, yet the very same specification can be functor-translated into Beam or Step Functions graphs when scale demands. Thus the language stays nimble for prototyping while remaining rigorous enough for audited, industrial-grade workflows.

2.3 Motivation and Design Rationale

Motivation and Design Rationale

Notation as a vehicle for thought. MPA was conceived to realize Iverson's maxim that *"notation is thought."* Authoring a pipeline as a concise chain of arrows obliges the analyst to expose the *epistemic structure*, rather than merely the mechanics, of an investigation. The very act of sketching a morphism diagram therefore doubles as reasoning about the problem at hand (Iverson, 1980).

Escaping YAML / JSON DAG clutter. Mainstream orchestrators such as Apache Beam ask users to compose pipelines in verbose Python or YAML/JSON files. Even modest workflows can sprawl into pages of boiler-plate that obscure intent. The equivalent MPA description `F --parse--> +VM --polarities--> +LP` fits on a single line while remaining fully executable; whenever cloud scale is needed, the semantics functor can export an isomorphic Beam graph (The Apache Software, n.d.).

From imperative loops to algebraic combinators. Imperative `for` loops are easy to type but hard to reason about. Following Hughes' *Why FP matters* argument, MPA replaces such code-oriented control flow with algebraic *higher-order operators* such as `Map`, `Comap`, explicit composition, and similar constructs, so recurring patterns become reusable, equationally tractable idioms (Hughes, 1989).

Falsifiable prompting instead of "prompt-and-pray." Typical LLM workflows stitch together raw strings or improvised JSON snippets with an eye toward rapid deployment rather than epistemic rigor. In MPA every prompt is a typed morphism whose **post-condition** must hold in the returned state. If that contract is violated, whether by hallucinated content, schema mismatches, or other latent errors, the run halts immediately, preventing faulty outputs from cascading through the remainder of the pipeline (Ji et al., 2023; OpenAi, 2023).

Reproducibility by construction. Modern pipelines touch many stochastic components: LLMs sample next tokens under temperature or nucleus settings; retrieval systems depend on mutable indexes; even a Google-search step may reorder links between invocations. Left unchecked, identical logical inputs can yield divergent outputs minutes apart. MPA therefore *memoizes* every morphism call **together with its complete input snapshot** (arguments and configuration). On re-execution the interpreter first consults the memo store; if a byte-for-byte match is found it replays the stored output, bypassing the stochastic service. The result is a "time-machine" mode that can reconstruct the exact state of a prior run even when the live world would not repeat it (Feldman, 1979; Michie, 1968). Deterministic replay thus becomes the default, aligning with reproducibility guidelines in computational science (Peng, 2011).

Instrumented for double-loop learning. Every associative-array snapshot carries its full transformation history. Analysts can therefore audit not only *what* happened but also *why*, enabling reflective double-loop learning and transparent epistemic audits (Argyris, 1977).

One algebra, many modalities. MPA unifies LLM calls, symbolic mathematics, and human edits under the same typing discipline. A Wolfram-Engine simplification(Wolfram Research, 2023a), a ChatGPT critique, and a domain-expert adjustment all appear as morphisms governed by identical semantic rules. The native interpreter runs on an ordinary laptop, yet the same script can be (future work) exported to cloud DAG engines for massive parallelism while maintaining formal rigor and practical usability.

2.4 Research Impact

Research Impact

Concise notation at a glance. The arrow calculus of MPA renders a full workflow on a single line, saving reviewers from paging through dozens of YAML or Python files and letting them grasp the data flow immediately (Iverson, 1980).

Typed, auditable workflows. Because every morphism carries explicit domain, codomain, and post-condition, a script forms a well-typed proof of correctness. Reviewers can run a type-checker instead of relying on informal assurances.

Deterministic replay. Operators are pure and memoized; the interpreter stores each call together with its input snapshot. Re-execution consults the store first, guaranteeing byte-identical output for identical inputs, which simplifies verification and regression testing (Michie, 1968; Peng, 2011).

Executable equations. The same equations that document a transformation are the code that runs it. Spec–implementation drift disappears, making independent replication far smoother.

Complete provenance. Each associative-array snapshot records inputs, outputs, parameters, and human decisions. The resulting log supports transparent epistemic audits rather than black-box reporting.

Falsifiable prompting. LLM calls are wrapped in morphisms whose post-conditions must hold. Violations: hallucinated content, schema errors, or similar, halt the run before bad data propagate (Ji et al., 2023; OpenAi, 2023).

Platform flexibility. The same script runs natively for quick tests and can, in future work, be exported to DAG engines such as Apache Beam or AWS Step Functions for cloud-scale execution, avoiding vendor lock-in (Amazon Web, n.d.; The Apache Software, n.d.).

Easy domain extension. Although only a System-Dynamics kit has been implemented to date, any researcher can introduce new types and assertions with minimal boiler-plate; higher-order operators continue to work unchanged.

Foundation for agentic automation. Because every construct is machine-checkable, a future agentic LLM could plan, verify, and execute an MPA pipeline under type-safe supervision, opening a path to trustworthy autonomous workflows.

2.5 Related Work

Related Work

Category theory as foundation. Eilenberg and Mac Lane introduced the language of objects and morphisms that underpins modern category theory; their framework is the conceptual bedrock for our practice of expressing every pipeline step as a typed morphism (Eilenberg & MacLane, 1945). Milewski later showed how these abstractions map onto real programs, a pathway echoed by MPA's policy of compiling its "math-ish" arrows directly to executable code (Milewski, 2019).

Data-flow and pipeline models. Apache Beam promotes a DAG-oriented model for large-scale data processing. MPA can export a similar graph while adding static types and equational proofs absent in Beam (The Apache Software, n.d.). Flow-Based Programming views software as networks of black-box processes; MPA keeps the pipeline metaphor but swaps untyped boxes for pure, typed morphisms (Morrison, n.d.). **Orchestration and scheduling.** Because operators are stateless, a complete MPA pipeline can (future work) run on AWS Batch with minimal glue (Amazon Web, 2025a). Optional (future work) DAG export also targets Amazon Steps (Amazon Web, 2025b)

Domain-specific modeling. Forrester's *Industrial Dynamics* established the stock–flow formalism that MPA manipulates when parsing System-Dynamics models (J. W. Forrester, 1961). Sterman's *Business Dynamics* (J. Sterman, 2000) codified causal-loop practice; MPA mirrors these conventions in its use of the CLD ADG and various CLD structure transformations and assertions.

Provenance and symbolic tooling. Palantir Foundry ships an interactive lineage graph for enterprise-grade provenance and collaboration (Palantir, 2025). MPA positions itself as a lighter, formally-typed alternative that still records complete provenance without the overhead of a full enterprise stack. Finally, Wolfram Engine(Wolfram Research, 2023a) offers a symbolic-math runtime that MPA can call as a morphism, illustrating how code-level tools plug into the algebra's math-level layer.

2.6 Limits and Future work

Local, Mathematica-based execution. The reference interpreter is implemented in *Wolfram Mathematica* and leverages its native multi-process / multi-thread kernel for parallelism (Wolfram Research, 2023b). Although this scales on a single server, standard operators such as `Map` do not automatically spill out to the cloud. A future *dynamic service-binding layer* is planned to translate a finished pipeline into an Apache Beam job or an Amazon Step Functions state machine, binding each morphism to an appropriate cloud service at run time.

LLM Assumption PA was conceived as a pipeline notation for interacting with chat-type LLMS. This assumption reflects into the operators, in particular there is Prompt which assumes it is prompting a chat type LLM. It is future work to relax the assumption, so Prompt is "send a string to something get a string back", no matter human or LLM or other. Other operators, e.g. "PlotColumnOfSpreadsheet" (not really an operator) might utilize an LLM or a conventional algorithm or a human. Future work is to remove such biases from PA and generalize it so that the semantic obligations of the operator are distinct from the choice of tactical method. This will also simplify meta-algorithmic manipulations of PA.

First-order type system. Only first-order objects are currently supported. Polymorphic collections, higher-rank arrows, and effect annotations remain open research topics. Extending the checker while keeping type inference decidable and fast will be challenging. No compelling use case for this has yet presented itself.

Side-effect discipline. All effects are threaded through an implicit state monad. Accurately modeling acquire–use–release lifecycles for files, GPUs, or cloud leases will require a linear or affine type system.

Meta-algorithmics. Simske's notion of *meta-algorithmics*, i.e. using adaptive policies to select, sequence, or ensemble multiple algorithms for robustness and cost control, has been sketched in PA but not yet realized (Simske, 2013). Future work will promote meta-operators to first-class morphisms so that a pipeline can carry an explicit *strategy layer*. That layer would (i) evaluate quality or cost metrics produced by earlier steps, (ii) consult a library of alternative morphisms, and (iii) dynamically bind the "best" operator under a user-supplied objective function. Concretely, we envision a `Select_Best{f_1,. . . , f_k}` combinator that, similar to AutoML loopers, spawns candidate morphisms inside a `Comap`, ranks their outputs, and passes the winner downstream. Adding types for accuracy, latency, and monetary cost, along with Bayesian or reinforcement-learning controllers, would let PA pipelines self-tune in situ while preserving formal proof obligations. Realizing this agenda requires (a) richer effect and metric types, (b) a memo-aware search scheduler, and (c) hooks for external optimizers, but would position PA as a fully adaptive, self-optimizing workflow algebra. It is likely that applying meta-algorithmics via generative AI will surface may interesting new capabilities.

Domain coverage. Built-in operators focus on System Dynamics and Causal-Loop Diagrams. Adapting the calculus to fields such as bioinformatics or econometrics will necessitate new type libraries and reference pipelines.

Human-in-the-loop ergonomics. Interactive checkpoints rely on free-form prompts inside a terminal session. A dedicated UI that surfaces state, validates edits, and re-executes only the affected sub-graph would broaden accessibility.

Rewrite system It is irritating to manually rewrite a pipeline from one notation (e.g. the implementation) into another (e.g. dense PA PA Cheat Sheet (Equation 2.1)). It would save author time and irritation to automate the transliterations. The various notations are isomorphic and so this shouldn't be too difficult.

Toward chat-oriented, inverted control flow. We ultimately aim to shift orchestration into the chat window itself. An agentic LLM would plan and type-check an MPA pipeline, then trigger terminal operators through *AWS Lambda* or concise Step Functions micro-flows. Execution would proceed "from the chat outward," turning a conversational interface into a fully instrumented SE workbench. Achieving this vision demands robust tool-calling APIs, runtime type enforcement, and graceful fault recovery: all challenges that define the next phase of research.

2.7 Notations

Notations

We use three equivalent notations. The most formal is the equational form shown in PA Cheat Sheet (Equation 2.1), typically used when epistemic considerations are foremost. A graphical form is shown in Morphisms, Mechanisms, Types (Figure 2.2), more suited to explanation. The graphical form is based on the inline arrow notation: $F \xrightarrow[\text{parse}]{} +VM \xrightarrow[\text{issd}]{} +\emptyset \xrightarrow[\text{polarities}]{} +LP \xrightarrow[\text{iscld}]{} +\emptyset$.

Operator arguments can be literal or extracted from the passed state, positional or named: $\mathrm{f}(T_{positional}, \mathit{literal} \mapsto \ldots, \mathit{indirect} \mapsto \mathrm{Key}(\mathit{myarticle})\}) \xrightarrow{\text{insert}} P_{problemstmt}$. Updates to the passed state are implied by the RHS of the outer arrow.

– NOTE Notional: some details elided or simplified

$(f \circ g)(x) \equiv f(g(x))$

$(f; g) \equiv g(f(x))$ – ; may be newline for top to bottom narrative

$$
\begin{aligned}
&\mathrm{Map}(f)\{a, b, c, \ldots\} \equiv \{f(a), f(b), f(c), \ldots\} \\
&\mathrm{Comap}(\{f, g\})(x) \equiv \{f(x), g(x)\} \\
&\mathrm{Doprompt}(\\
&\qquad \mathrm{Key}(P_{problemstmt}) \\
&\qquad \mathrm{Session} \rightarrow \mathrm{Autonomous}, \\
&\qquad \mathrm{Endpoint} \rightarrow ''chatgpt - o1'' \\
&) \xrightarrow{\text{insert}} L_{problemstmt} \\
&\circ \mathrm{ExpandTemplate}(\\
&\qquad \mathrm{Key}(T_{base}), \\
&\qquad \{prompt \mapsto \ldots, article \mapsto \mathrm{Key}(myarticle)\}, \\
&) \xrightarrow{\text{insert}} P_{problemstmt} \\
&\circ \mathrm{Set}(\{T_{base} \mapsto \ldots, myarticle \mapsto \ldots, G_{smecld} \mapsto \ldots, F_1 \mapsto \ldots, F_2 \mapsto \ldots\})(\varnothing)
\end{aligned}
\tag{2.1}
$$

NOTES:

1. Hungarian notation used to improve readability:
   - $A \mapsto$ kvmap
   - $F_{name} \mapsto$ File name
   - $G_{name} \mapsto$ ADG
   - $J_{name} \mapsto$ JSON string or struct
   - $L_{name} \mapsto$ LaTeX
   - $M_{name} \mapsto$ Markdown native chat rv
   - $P_{name} \mapsto$ prompt
   - $S_{name} \mapsto$ String(J|L|M|P|...).
   - $T_{name} \mapsto$ Template
   - $MID_{name} \mapsto$ e.g. "o3" or "o3-Anonymous"
   - $SID_{name} \mapsto$ opaque from OpenSession
   - $TGT_{name} \mapsto$ e.g. "chatgpt"

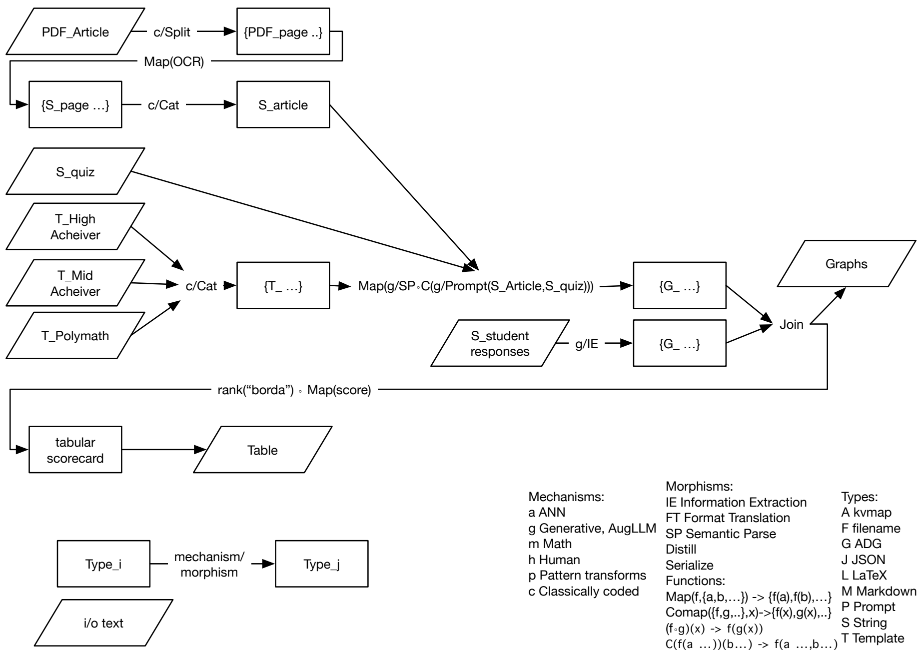


Figure 2.2: The janis ch8 images of pages is OCR'd into text for subsequent processing. The exogenous student responses are textual. The reference CLD is exogenous. The de-identified responses are processed into uniform ADG representation of the response CLDs. AugAI is prompted to answer the quiz question, in three distinct roles (effectively as pseudo-students): B-level student; overachieving student; and overachieving professional polymath. The aggregated set of student, reference, and AugAI CLDs are scored on several measures, and ranked by an aggregation of two measures of loop count.

3 Model-Based Systems Thinking

3.1 Summary

We use the definition of Model-Based Systems Thinking (MBST) as presented in (Shahroudi et al., 2025): It is a structured approach that employs various modeling techniques to enhance the comprehension of complex, real-world issues. It renders rigorous the act of Systems Thinking. This methodology serves to augment one's capacity to apply fundamental principles of interconnectedness and system behavior to diverse problems, whether they are technical, social, or a hybrid of both. It moves beyond simple textual or pictorial descriptions, utilizing dynamic models to represent and analyze intricate situations. The ultimate aim is to improve understanding, which in turn supports more effective decision-making and policy development.

A rigorous method, which is enabled by a framework, languages, and tools, systemically and reliably enforces the disciplined application of principles for a specific purpose, as demonstrated by MBST. The tools within such a method are often formal, which refers to something based on a clear, established set of rules and conventions. For example, the text describes the "formal models" and "formalized languages" like SysML used in MBSE.

This definition of MBST is consistent with that used in (Lavi et al., 2020). Despite the title, this paper does not explicitly define MBST, but we can infer the definition from the context. MBST is the level and quality of systems-thinking skill that becomes visible when students build and refine explicit, function-structure-behavior models with OPM, where OPM is itself a rigorously specified modeling language whose symbols, grammar, and semantics are fixed by an international standard, making the model objectively interpretable and suitable for reliable scoring in their STAR rubric. That is to say, OPM is formal.

While Shahroudi et al. define MBST as a broad, principle-based method for understanding problems, Lavi et al. present a more tool-centric view focused on the formal application of a specific modeling language like OPM. This aligns with the 'rigorous' and 'formal' aspects of the broader definition, but narrows the focus from a general philosophy to a specific, objectively measurable practice. The Shahroudi definition is broader and more applicable than Levi for our purposes: Our method is centered on applying multiple methods in a coordinated fashion, al la "... explicit framework for integrating a broader range of modeling methods ..."(Shahroudi et al., 2025, §11.5.1).

The MBST framework is built around an iterative and incremental process characterized by distinct "slow" and "fast" learning loops. The fast learning loop allows for relatively quick exploration and refinement by exercising an established and validated model through activities like sensitivity analysis, scenario testing, and policy evaluation. In contrast, the slow learning loop involves a more profound re-evaluation of the problem itself; it entails repeating the main steps of the MBST method to update core mental models and redefine the problem when deeper understanding is required or when initial assumptions are challenged. This dual-loop approach facilitates both operational adjustments and deeper, structural learning, echoing established concepts where foundational beliefs are questioned to achieve transformative insights (Argyris, 1977).

This method is designed to be applicable across different fields of study and professional domains, helping to identify significant patterns of behavior within complex data, uncover the underlying causal structures that generate these patterns, and understand the mutual influence of a system's purpose, architecture, and behavior. Such an approach is crucial for pinpointing effective leverage points where interventions can yield substantial impact, and for predicting potential future developments and unintended consequences of decisions or actions (J. Sterman, 2000). It aims to foster a shared, evolving understanding among individuals and teams grappling with systemic challenges.

MBST places significant emphasis on the human and cultural elements within systems. It inherently addresses how human mental models shape decision-making and influence the architecture of systems, recognizing that these models are learnable and adaptable. The adoption and practice of MBST encourages a mindset comfortable with complexity and uncertainty(Hung, 2008), fostering a commitment to continuous learning and adaptation. It necessitates collaboration across disciplines, as holistic understanding rarely resides within a single expertise. By making mental models and their impacts more explicit through modeling, MBST can contribute to shifting an organizational culture towards one that more readily embraces and fosters systems thinking in its daily operations and strategic outlook (Senge, 1990).

MBST is distinct from other model-centric methodologies. While it draws heavily from systems thinking and system dynamics principles (J. Sterman, 2000) and concepts of organizational learning (Senge, 1990), it provides an explicit framework for integrating a broader range of modeling methods. This integration is intended to ensure that various modeling techniques are applied in a way that is consistent with a holistic, systemic perspective. Unlike approaches primarily geared towards the creation of products or services, such as some interpretations of Model-Based Systems Engineering, MBST's emphasis is significantly on deeply understanding the problem itself and developing robust mental models before focusing on solutions. This deep engagement with problem definition and comprehension is considered crucial for developing valuable and sustainable interventions.

3.2 PA Accelerates MBST

Model-Based Systems Thinking (MBST) (Shahroudi et al., 2025) offers a structured framework enabling individual analysts to achieve profound learning, notably via its "slow learning loop," which corresponds with double-loop learning. This advanced learning (Argyris, 1977) involves an analyst transcending mere error correction (single-loop learning) to critically re-evaluate and modify core assumptions, governing variables, and mental models that underpin their approach to complex systems. Such a shift necessitates a re-evaluation of foundational premises guiding perception and action, crucial for transformative insights over superficial adjustments.

Pipeline Algebra (chapter 2) (PA) functions as an integral component of MBST's "rigorous methods," providing the formalism and repeatability essential for this depth of inquiry. PA formally defines analytical workflows as sequences of typed morphisms and pure, idempotent operators acting upon passed state (Moggi, 1991) and drawing on higher-order functional programming principles (Hughes, 1989). Its algebraic and category-theoretic underpinnings (Spivak, 2014) ensure explicit process specification. PA's inherent characteristics—formal notation aligning specification with execution, operator idempotency (ensuring consistent outputs from identical inputs despite stochastic elements like LLMs), and deterministic audit trail generation (Peng, 2011)—render the analytical process transparent, verifiable, and repeatable. This robust formalism and guaranteed repeatability establish PA as a rigorous instrument within the MBST framework for executing complex data transformations and analyses.

Leveraging PA as a rigorous tool allows an analyst to render their mental models explicit and amenable to stringent testing. Specifically, PA's typed morphisms and categorical structure enable the analyst to reify their mental model by formally defining its conceptual elements (entities of thought) and the permissible transformations (morphisms) between them. Consequently, an instantiated PA pipeline constitutes an epistemic representation of the analyst's working mental model. Should PA-orchestrated analytical workflows—potentially integrating diverse methods like AI and human-in-the-loop validation—yield outcomes conflicting with expectations or empirical data, PA's explicitness and traceability enable systematic identification of flawed formalized assumptions or process steps. Such rigorous, model-assisted reflection, facilitated by PA's formal and repeatable nature, empowers the analyst to more effectively challenge and revise their foundational understanding. Congruent with Iverson's assertion that "Notation is thought" (Iverson, 1980), PA aids the analyst in meticulously deconstructing and reconstructing their comprehension, thereby supporting the critical inquiries and paradigm shifts inherent to double-loop learning.

4 Mental Model of Agentic Large-Language Models

This section offers a concise conceptual framework that links the principal technical layers of ChatGPT-style services into one coherent picture. The goal is to equip the reader with a shared vocabulary and set of expectations that later chapters will draw upon.

Everything that follows is a well-informed reconstruction of what is probably happening when a user converses with a chat-style AI service such as *ChatGPT o3*. Because providers do not publish full engineering blueprints, this account relies on research papers, public API guides, and observed behavior; if proprietary details were available, a more granular description could be offered. Treat the model below as an evidence-based hypothesis rather than an authorized technical disclosure.

A large-language model (LLM)(Tunstall et al., 2022) supplies the core capability. Conceptually it is an autoregressive Transformer with hundreds of billions of parameters that has been pre-trained to predict the next token over a vast corpus. The GPT-3 study, which used 175 billion parameters, first showed that such a model can solve unseen tasks from only a few in-context examples or a single instruction, explaining the general-purpose feel of contemporary chat systems (Brown et al., 2020).

Reasoning quality improves when the conversation induces the model (typically but not necessarily implicitly) to write out its intermediate steps. In chain-of-thought (CoT) prompting the model is asked to, or asks itself to, think step by step, emit those reasoning tokens, and only then produce its answer. The extra tokens act like a private scratch pad that lets the network decompose the task and self-check deductive moves, yielding large accuracy gains on arithmetic, commonsense, and symbolic-logic benchmarks (Wei et al., 2022).

CoT becomes more effective inside an agentic loop whose cycle is Think → Act → Observe

→ Reflect. The ReAct framework (Yao et al., 2022) formalizes that pattern: each Thought line is a CoT fragment, each Action line invokes an external capability, and each Observation records the result so the next thought can adjust the plan. Combining explicit reasoning traces with real-world actions is what recent literature calls an agentic LLM.

Those actions are executed through tools, which are named capabilities that the developer exposes to the model. OpenAI's public tooling list includes built-ins such as `code interpreter` (a sandboxed Python executor), `retrieval/file search` (vector search over user documents), simple calculators, and unit converters, while custom tools can wrap anything from "get weather" to "create invoice.".

We assume that commercial hosts allocate a bounded slice of compute to every prompt. OpenAI documents a maximum context window (currently 128 k tokens for GPT-4o-mini), caps generated output (16 k tokens), and enforces per-user rate limits. Internally a scheduler probably also limits wall-clock time to keep latency and costs predictable. Long chains of thought and multiple tool calls therefore consume limited resource-seconds, so both the model and the application author must plan economically.

The system is intrinsically stochastic and continually evolving. At inference time each token is sampled from a probability distribution shaped by parameters such as temperature; identical prompts therefore yield paraphrases or occasionally divergent answers, especially when the temperature is above zero (Holtzman et al., 2019). Over longer horizons the service provider ships new checkpoints, so outputs can drift even with temperature $= 0$.

These capabilities have been given many names as the field has evolved: agentic AI, LLM+, composite AI, generative agents, autonomous agents, tool-augmented LLMs, and, in casual speech, simply AI.

5 Systems Thinking

5.1 Summary

Systems Thinking, as conceptualized by Senge, is a discipline for comprehending wholes, interrelationships, and dynamic patterns rather than isolated, static components (Senge, 1990). It provides a language and framework to understand the forces shaping system behavior, deriving its tools from cybernetics and servomechanism theory to enable more effective systemic intervention.

The genesis of contemporary feedback-centric Systems Thinking resides in System Dynamics, established by Forrester's "Industrial Dynamics" (J. W. Forrester, 1961). This foundational work elucidated how system structure, particularly its feedback loop networks, governs dynamic behavior. Forrester rigorously advocated for quantitative modeling and computer simulation to analyze complex systems, prioritizing numerical validation over purely qualitative descriptions for understanding and managing such systems (J. W. Forrester, 1961).

Senge's "The Fifth Discipline"(Senge, 1990) was pivotal in disseminating System Dynamics-based Systems Thinking to a broader organizational audience. It operationalized this thinking through tools like systems archetypes—recurrent patterns of behavior linked to specific feedback structures—and the use of causal loop diagrams for mapping systemic interrelationships. Senge underscored the necessity of interrogating mental models, the ingrained assumptions shaping systemic perception and action, positing Systems Thinking as the cornerstone of the 'learning organization' to enhance collective understanding and generative capacity.

6 Causal Loop Diagram

6.1 Introduction

A causal loop diagram (M. Goodman, 1973) is a conceptual diagram made up of variables connected by arrows with polarity signs (+/–) that indicate the direction and nature of causality, forming feedback loops that represent the structure of a dynamic system.

We represent a CLD formally as an attributed directed graph (ADG), where the variables are vertices and links are directed edges. In the case of CLD, the edge attributes certainly include the link polarity. We also use various augmented CLD representations, in which case the attribute might include an indication of link delay, magnitude of effect, and other link or vertex-specific attributes.

The semantics of the CLD are quite simple: An arrow from variable A to variable B with a polarity sign "+" or "-" indicates that, if the "value" of variable A is incrementally increased, the "value" of B will incrementally either increase ("+") or decrease ("-"), ceteris paribus. "Increased" generally means "towards $+\infty$", not "away from zero". A loop is defined as (B)alancing if it has an odd number of "-" links, because ultimately a disturbing force will tend to be counteracted, else its (R)eenforcing, and will tend to amplify a disturbing force.

From a semantic standpoint, each variable in the diagram denotes an attribute or condition that can vary in magnitude (more on this restriction to scalar values later), and each link's polarity determines whether an incremental increase (or decrease) in one variable leads to a corresponding increase (or decrease) in another. Reinforcing loops (Rn) encapsulate processes that accelerate growth or decline by sending changes back through the system in a manner that compounds their initial effect. In contrast, balancing loops (Bn) are characterized by self-correcting tendencies, whereby shifts in one direction eventually produce counteractions that moderate or reverse the original trend. Delays add further complexity by introducing lags between cause and effect, an essential consideration for accurately interpreting feedback phenomena and anticipating system behavior.

Causal loop diagrams are limited because variables without clear scalar algebraic meaning, such as those that aggregate their input, make polarity hard to define, which can blur the distinction between accumulation and direct causation and lead to misinterpretation. For example, while water flow rate into a bucket might go up or down, the level of the water in the bucket will always go up (Richardson, 1986b). This is the "bathtub paradox" (Sweeney & Sterman, 2000).

6.2 Why the CLD

In the process of selecting a suitable representation method for this work, both concept maps ((Novak, 2010)) and Causal Loop Diagrams ((J. Sterman, 2000)) were considered after down-select a broad range of representations (see Representations Tradespace Rubric (section 6.6)). Concept maps are well-regarded for capturing broad, associative relationships among ideas, making them a strong candidate for initial brainstorming and revealing general conceptual linkages. However, we hypothesize that when the goal is to highlight feedback loops and explore the dynamic nature of a system, particularly how changes propagate through interconnected components, Causal Loop Diagrams may be more suitable. If this proves accurate, focusing on feedback mechanisms within a CLD might offer a clearer pathway to understanding how system components influence one another over time, ultimately enabling more informed exploration of complex behaviors. We note this approach was advocated by (M. R. Goodman, 1974).

Although this premise is not yet fully substantiated by the literature (see (Trochim et al., 2006)), our work aims to provide evidence in support of the notion that CLDs, augmented by advanced AI capabilities, can more effectively illuminate system dynamics.

Traditionally, the non-quantitative nature of CLDs has limited their perceived applicability in high-stakes contexts ((Richardson, 1986a), (OECD, 2017)). We propose that large-scale language models, e.g. ChatGPT o1 ca. 12/2024, can shift this dynamic. By projecting textual inputs into high-dimensional latent vector spaces and retrieving distributed domain knowledge, such models may provide principled assistance in both the elicitation and the semantic evaluation of CLDs. The present study is designed to test this conjecture

6.3 History

The earliest graphical representation of the Causal Loop Diagram (CLD) that we have identified appears in (Maruyama, 1963) (p. 176, figure 3). The accompanying text describes the syntax and semantics we today recognize as the CLD, though the author does not use that name. Balancing loops are labeled "deviation-counteracting," while reinforcing loops are termed "deviation-amplifying." Notably, (Maruyama, 1963) also describes what we now call the "count the minus signs" method of determining loop polarity, which appears to be the earliest mention of this technique.

We find the first use of the term "Causal Loop Diagram," employing syntax and semantics that mirror Maruyama entering the SD community in Goodmans dissertation goodman1972, though apparently not without some controversy ((Richardson, 1986a), (Richardson, 1986b), D-3312-2 in (J. Forrester, 2004), although Richardson uses the CLD in the SD development process in richardson1981). Goodman defines the CLD in detail in (M. Goodman, 1973)). Goodman (M. R. Goodman, 1974) uses the CLD as part of a broader modeling process, first capturing key causal relationships before transitioning to quantitative stock-and-flow System Dynamics models. (Richardson & Pugh, 1981) cites (M. R. Goodman, 1972), suggesting a possible path by which Richardson adopted the nomenclature we use today. In his later work, (Richardson, 1991) devotes a full section (section 4.1) to (Maruyama, 1963), implying that he may have become aware of Maruyama through Goodman.

Many later CLD publications are essentially rehashes of the CLD best-practice rules submitted by Goodman and Richardson ((Richardson & Pugh, 1981) p. 28), though (Tip, 2011) suggested link length as a an indicator of magnitude of delay: We do not use that convention and it does not seem to have caught on in general practice.

(Richardson, 1986a) critiques causal loop diagrams. He discusses how CLDs can be ambiguous or misleading when they overly simplify complex interactions or when modelers rely too heavily on circular or purely verbal logic. He highlights the dangers of imprecise labeling (e.g., "polarity" signs) and the tendency to confuse correlation with causation.

We found the earliest description of "virtuous cycle" and "vicious cycle" in D-4498(1995)((J. Forrester, 2004)), where they are both defined as reenforcing, one "good" and the other "bad". (Richardson & Pugh, 1981) (p. 11) does use the term "vicious cycle" and notes it must be reenforcing, so apparently the nomenclature had been around a while, if not formalized, prior to D-4498.

We find no citations between Forrester and Maruyama, even in Forrester's unpublished works. However, we did locate a single mention of Maruyama in Forrester's archive (J. Forrester, 2004) ("Box 128 Folder 1", file 02-000451725.pdf) within a 1962 issue of "FUTURES" that includes an advertisement for a "Cultural Futurology Symposium: Pre-Conference Volume," listing Maruyama among other contributors. We find no evidence that either influenced the other.

It remains plausible that Forrester was aware of Maruyama's work but chose not to cite it, perhaps because Forrester emphasized the importance of stock accumulations in understanding dynamic systems: an element not central to maruyama1963's purely causal-loop perspective.

6.4 Syntax

Effective 2/2025 we use JSON to serialize CLDs, structured as shown in JSON Example of output syntax (Table 6.1),JSON Explanation of Fields (Table 6.2). JSON is human, machine, and AI readable, writable, and comprehensible. Our prior syntax was an ad-hoc flatfile format, but we have found it is no longer necessary to use a simplified format for AI: JSON seems to be well supported now.

The Attributed Directed Graph (ADG) (chapter 7) is our internal representation.

Table 6.1: Prompt: "Lets say I want to query you to create a causal loop diagram where it is easy for you to both serialize and comprehend the CLD, and it easy for me to parse the serialization for external processing. What format do you suggest? ... OK, lets go with JSON. Make the following modifications. For variables, include add a member 'variable_description'. For links, add a field 'polarity_magnitude' which will be an integer 1..10 (low to high magnitude of effect) and a field 'link_description' that is a textual description of the link, 'link_justification' that is a citation to justify the link, and 'link_delay' that is 1..10 indicating low delay to high delay. Recapitulate the value of JSON and show an example output using the format you proposed modified as specified above.". Generated by ChatGPT o1 on 2/13/2025. I am Kirk Reinholtz, and I approve of this AI generated text.

JSON (JavaScript Object Notation) is a well-structured, text-based data format that is both human-readable and easily parsed by standard libraries in most programming languages. Moreover, JSON is also readily generated and comprehended by ChatGPT, making it a convenient choice for defining a Causal Loop Diagram (CLD). Below is an example demonstrating how to include fields such as variable_description” for variables, and polarity_magnitude”, link_description”, link_justification”, and “link_delay” for links.


```
{
  "title": "My Enhanced CLD Example",
  "variables": [
    {
      "id": "v1",
      "label": "Population",
      "variable_description": "Number of people in the system"
    },
    {
      "id": "v2",
      "label": "Birth Rate",
      "variable_description": "Rate at which new individuals enter the population"
    }
  ],
  "links": [
    {
      "source": "v1",
      "target": "v2",
      "polarity": "+",
      "polarity_magnitude": 8,
      "link_description": "A larger population generally leads to an increased birth rate",
      "link_justification": "Smith & Jones (2020), Chapter 3",
      "link_delay": 2
    },
    {
      "source": "v2",
      "target": "v1",
      "polarity": "+",
      "polarity_magnitude": 9,
      "link_description": "Higher birth rate increases population",
      "link_justification": "Demographics Journal (2019)",
      "link_delay": 1
    }
  ]
}
```

Table 6.2: (Continued)

Explanation of Fields:

- "title": A short, descriptive name for the CLD.
- "variables": A list of variable objects, each with:
  - "id": A unique identifier (used in links).
  - "label": The variable's display name.
  - "variable_description": A textual definition or explanation of the variable.
- "links": A list of link objects, each describing a causal connection with:
  - "source": The ID of the source variable.
  - "target": The ID of the target variable.
  - "polarity": Typically "+" (reinforcing) or "-" (balancing).
  - "polarity_magnitude": An integer 1–10 indicating the strength of the effect.
  - "link_description": A short statement explaining how the source affects the target.
  - "link_justification": A reference or citation supporting the relationship.
  - "link_delay": An integer 1–10 indicating the time delay before the effect takes place.

Because JSON is standardized and clearly structured, it makes it straightforward for ChatGPT (and other systems) to generate, modify, and interpret the details of your CLD, even when complex relationships and additional metadata are included. We have observed that it can take signifiant ChatGPT resources to render JSON. It seems best practice to use iterative prompting, only requesting the JSON after the semantics results have been achieved.

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "https://example.com/cld.schema.json",
  "title": "Causal Loop Diagram Schema",
  "description": "A JSON schema for representing a Causal Loop Diagram (CLD), including
      variables and their causal links.",
  "type": "object",
  "properties": {
    "variables": {
      "type": "array",
      "description": "List of variables in the system",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string",
```

```
            "minLength": 10,
            "description": "A descriptive name (>= 10 characters) for the variable, e.g. 'Greenhouse Gas Emissions Rate'"
          },
          "description": {
            "type": "string",
            "description": "An explanation of what the variable is and where it originated"
          },
          "units": {
            "type": "string",
            "description": "Industry-standard unit for the variable or 'DimensionlessUnit' if no units apply"
          }
        },
        "required": ["name", "description", "units"]
      }
    },
    "links": {
      "type": "array",
      "description": "List of causal links connecting the variables",
      "items": {
        "type": "object",
        "properties": {
          "source": {
            "type": "string",
            "description": "The name of the source variable for this link"
          },
          "destination": {
            "type": "string",
            "description": "The name of the destination variable for this link"
          },
          "polarity": {
            "type": "string",
            "enum": ["+", "-","+/-","unknown"],
            "description": "Indicates whether changes in source cause a same-direction (+) or opposite-direction (-) change in the destination"
          }
        },
        "required": ["source", "destination", "polarity"]
      }
    }
  },
  "required": ["variables", "links"]
}
```

6.5 On Causality

The very name "Causal Loop Diagram" raises the question: What is "causality"? This question has been examined for thousands of years. See, for example, (Pearl, 2000, p. 401-428) in the epilogue "The Art and Science of Cause and Effect." For our purposes, we adopt the view that causal conclusions arise from a mix of empirical evidence and conceptual scaffolding: if changing A results in a change in B, ceteris paribus, we say A causes B. Engineering judgment is required.

One may argue that F = M A does not itself imply a causal direction. However, if we are interested in how thrust influences acceleration, we might choose to say "force causes acceleration." Logical problems can arise if we also try to say "acceleration causes force" in an attempt to construct a "complete" model. If such completeness is taken as the goal, then one would need to model the entire universe, at which point the notion of causality, and even the roles of the modeler and the system being modeled, do indeed disappear. It is up to the modeler to avoid this pitfall, as stepping beyond a causal framework undermines the very idea of causality.

We must decide which variable we regard as manipulable in real-world scenarios. Often, we say an external force is applied (cause), which produces an acceleration (effect) in the mass. Formally, we might choose a structural equation $a = F/M$ (assuming mass is fixed), indicating that if we "do" something to F, for instance, push harder, then a changes in a predictable way. That interventionist orientation fixes the direction of causality in our model, even though the algebraic statement $F = ma$ is symmetrical.

6.6 Representations Tradespace Rubric

Representations Considered

To systematically evaluate a wide range of representational tools, the following rubric was constructed with the assistance of a large language model (ChatGPT 4o) to ensure a consistent template was applied to each method. The author has reviewed and verified the content of each entry for accuracy and relevance.

It is future work to tie these into the model classification scheme in (J. W. Forrester, 1961, Figure 4-1) .

- **Causal Loop Diagrams (CLDs)**
  - Rating: 5

  - Purpose: Show feedback loops and system behavior
  - Structure: Nodes (variables) with + or - links indicating influence, forming loops
  - Scope: Broad system view
  - Dynamic?: Yes
  - Graph Structure: Graph with cycles
  - Seminal Description: (J. Sterman, 2000)
  - Justification:
    * Semantic Range: Captures feedback (explicitly!), delays, and non-linear relationships.
    * Comprehensibility: Relatively easy for humans and AI to understand.
    * Reasoning/Generation: AI can identify loops, suggest variables, and simulate; humans interpret behavior and implications.

- **System Dynamics Models**
  - Rating: 4
  - Purpose: Simulate system behavior over time
  - Structure: Stocks, flows, converters, connectors
  - Scope: Broad system view
  - Dynamic?: Yes
  - Graph Structure: Graph with cycles
  - Seminal Description: (J. Sterman, 2000)
  - Justification:
    * Semantic Range: Captures accumulation, flows, and feedback. More quantitative than CLDs.

    * Comprehensibility: Can be complex for humans and AI due to the math.
    * Reasoning/Generation: AI can simulate; humans interpret and ensure realism.

- **Fishbone Diagrams (Ishikawa)**
  - Rating: 2
  - Purpose: Identify root causes of a particular problem
  - Structure: A central spine with branches representing categories of potential causes
  - Scope: Problem-focused
  - Dynamic?: No
  - Graph Structure: Branching structure (no cycles)
  - Seminal Description: (Ishikawa & Loftus, 1990)
  - Justification:
    * Semantic Range: Focused on categorizing causes, less on feedback.
    * Comprehensibility: Straightforward for humans; AI can parse categories but limited in dynamic interpretation.
    * Reasoning/Generation: AI can suggest categories; humans refine and validate.

- **Bayesian Networks**
  - Rating: 3
  - Purpose: Model probabilistic relationships and conditional dependencies
  - Structure: Directed acyclic graphs (nodes are variables, edges are conditional dependencies)

- Scope: Variable-focused, uncertainty modeling
- Dynamic?: Not inherently (static structure, can be extended over time)
- Graph Structure: Directed acyclic graph
- Seminal Description: (Pearl, 2014)
- Justification:
  * Semantic Range: Captures probabilistic dependencies, not explicitly feedback loops.
  * Comprehensibility: Requires understanding of conditional probabilities.
  * Reasoning/Generation: AI can infer probabilities and structure; humans ensure domain plausibility.

- **Concept Maps**
  - Rating: 4
  - Purpose: Explore and communicate complex ideas
  - Structure: Nodes (concepts) with linking phrases
  - Scope: Can vary, often broad and flexible
  - Dynamic?: No
  - Graph Structure: Graph (can have cycles)
  - Seminal Description: (Novak, 2010)
  - Justification:
    * Semantic Range: Captures a wide range of relationships, including feedback.
    * Comprehensibility: Intuitive for humans and AI.
    * Reasoning/Generation: AI can suggest connections, generate new concepts; humans refine and clarify.

- **Influence Diagrams**

- Rating: 3
- Purpose: Represent decision problems, showing decisions, uncertainties, and objectives
- Structure: Nodes (decisions, uncertainties, values) with arrows indicating influences and information flow
- Scope: Decision-focused
- Dynamic?: Not inherently (often static)
- Graph Structure: Directed graph (often acyclic)
- Seminal Description: (Matheson & Howard, 1984; Pearl, 2005)
- Justification:
  * Semantic Range: Emphasizes decision-related influences rather than broad feedback loops.
  * Comprehensibility: Understandable with domain knowledge.
  * Reasoning/Generation: AI can propose structures; humans add strategic context.

- **Stock and Flow Diagrams**
  - Rating: 4
  - Purpose: Represent accumulations (stocks) and their rates of change (flows)
  - Structure: Stocks (boxes), flows (arrows with valves), auxiliaries connected by links
  - Scope: Often broad, part of System Dynamics
  - Dynamic?: Yes
  - Graph Structure: Graph with flows and feedback loops
  - Seminal Description: (J. W. Forrester, 1961)

    - Justification:
        * Semantic Range: Captures accumulations and feedback in a structured form.
        * Comprehensibility: Understandable once trained; AI can parse and simulate.
        * Reasoning/Generation: AI can draft SFDs; humans ensure the structure represents the system accurately.

- **Process Flow Charts**
    - Rating: 2
    - Purpose: Document the sequence of steps in a process
    - Structure: Standardized symbols (e.g., rectangles, arrows) representing actions and flows
    - Scope: Usually narrow, task or process-oriented
    - Dynamic?: No (represents static sequence)
    - Graph Structure: Directed flow (often no cycles)
    - Seminal Description: (Gilbreth & Gilbreth, 1921)
    - Justification:
        * Semantic Range: Limited to process steps, not feedback loops.
        * Comprehensibility: Straightforward; AI can parse sequences easily.
        * Reasoning/Generation: AI can create charts from text; humans refine correctness.

- **State Transition Diagrams**
    - Rating: 4
    - Purpose: Show how a system transitions between discrete states
    - Structure: States as nodes, transitions as directed edges triggered by events/conditions

- Scope: Focused on system states and transitions
- Dynamic?: Yes (discrete states over time)
- Graph Structure: Directed graph (cycles possible if returning to states)
- Seminal Description: (Harel, 1987)
- Justification:
  - Semantic Range: Captures transitions, not continuous feedback.
  - Comprehensibility: Understandable to humans and AI.
  - Reasoning/Generation: AI can suggest transitions; humans confirm logic.

- **Systemigrams**
  - Rating: 2
  - Purpose: Depict complex systems and their narratives in a cohesive story
  - Structure: Narrative-rich graphs of interconnected elements
  - Scope: Broad, narrative-focused
  - Dynamic?: Not inherently
  - Graph Structure: Graph with story-like connections
  - Seminal Description: (Boardman & Sauser, 2008)
  - Justification:
    - Semantic Range: Broad, can capture complex relationships but less formal.
    - Comprehensibility: More interpretive; AI can assist, but humans must define narrative.
    - Reasoning/Generation: AI can help with structure; humans shape the story.

- **Mind Maps**
  - Rating: 3

- Purpose: Brainstorm and organize ideas
- Structure: Central concept with branching ideas
- Scope: Can vary, often broad and flexible
- Dynamic?: No
- Graph Structure: Tree
- Seminal Description: (Buzan & Buzan, 1993)
- Justification:
  * Semantic Range: Captures diverse relationships between ideas, including feedback loops.
  * Comprehensibility: Intuitive for humans and AI.
  * Reasoning/Generation: AI can add branches; humans refine structure and meaning.

- **Flow Charts with Decision Points**
  - Rating: 2
  - Purpose: Show processes with explicit decision nodes
  - Structure: Boxes for processes, diamonds for decisions, arrows for logic flow
  - Scope: Process and decision-oriented
  - Dynamic?: No
  - Graph Structure: Flow with branching
  - Justification:
    * Semantic Range: Logical sequences and decisions, not feedback loops.
    * Comprehensibility: Clear for humans and AI.
    * Reasoning/Generation: AI can produce flow charts; humans ensure correctness.

- **Rich Pictures**
  - Rating: 1
  - Purpose: Provide a holistic qualitative depiction of a complex situation
  - Structure: Free-form drawings, symbols, text
  - Scope: Broad, qualitative
  - Dynamic?: No
  - Graph Structure: Not formally defined
  - Seminal Description: (Checkland, 1981)
  - Justification:
    * Semantic Range: Very broad but not strictly defined; hard for AI to parse thoroughly.
    * Comprehensibility: Visual and interpretive, mainly for humans.
    * Reasoning/Generation: AI can label elements; humans interpret meaning.
- **Behavior Over Time Graphs (BOTG)**
  - Rating: 3
  - Purpose: Depict variable trajectories over time, identifying dynamic patterns
  - Structure: Time-series plots of key variables
  - Scope: Often broad, used alongside CLDs and Stock/Flow diagrams
  - Dynamic?: Yes (represents changes over time)
  - Graph Structure: Not a node-link diagram, but temporal plots
  - Justification:
    * Semantic Range: Focuses on behavior patterns (e.g., growth, oscillation).
    * Comprehensibility: Easy for humans to interpret time-series; AI can process numeric data.

- ∗ Reasoning/Generation: AI can detect patterns, humans interpret significance and link patterns to structure.

Property Descriptions:

Rating: A subjective numerical assessment (0-5) indicating the overall usefulness of the tool in human-AI collaborative analysis where feedback is important. A higher rating suggests that both humans and AI can easily read, write, and reason about products created using the tool's syntax and semantics.

Purpose: A brief summary of what the tool is designed to accomplish-whether it focuses on identifying feedback loops, exploring conceptual relationships, capturing probabilistic dependencies, guiding decision-making, or documenting processes.

Structure: Describes the fundamental components and notation of the tool, such as types of nodes, edges, and whether elements represent stocks, flows, variables, states, or concepts.

Scope: Indicates the general breadth or domain of the tool. Some tools are best suited for broad, system-level conceptualization, while others focus on narrow processes, discrete states, or root-cause analysis.

Dynamic?: States whether the tool inherently supports or encourages representation of dynamic behavior over time. "Yes" suggests that changes, feedback loops, or time-evolving patterns are integral to the tool's usage. "No" typically means the tool is static and does not inherently represent temporal dynamics.

Graph Structure: Provides a characterization of the tool's underlying graphical form (e.g., directed acyclic graph, graph with cycles, tree structure). This helps clarify how nodes and edges interact and whether feedback loops or cycles can be represented.

Seminal Description: References a key, foundational work-often a book or paper-where the tool was first introduced, defined, or popularized. These citations provide historical and conceptual grounding.

Justification: Explains why the tool is rated or valued as it is. This includes considerations like:

- *Semantic Range:* The kinds of relationships or phenomena the tool can represent.
- *Comprehensibility:* How easy it is for humans to understand and for AI to parse the representation.
- *Reasoning/Generation:* Whether AI can generate or reason about structures in the tool's format and how humans add interpretive or domain knowledge.

7 Attributed Directed Graph (ADG)

An Attributed Directed Graph (ADG) extends the conventional directed graph by assigning zero or more named attributes not only to each vertex and each directed edge, but also to the graph as a whole. These attributes can denote numeric values, string labels, logical flags, or any type of metadata relevant to the application domain (e.g., weights, capacities, probabilities, or version tags). This structure is especially useful in complex systems modeling, where edges and vertices may carry descriptive annotations (such as variable equations in system dynamics) and the overall model may have global properties or parameters (e.g., time stamps or overarching labels). The ADG thus combines the usual graph-theoretic relationships with flexible annotation mechanisms to capture richer semantic information about the system under study.

We use the ADG to represent a CLD and an SD model, among other things. All required information is associated with the ADG as attributes. We do not apply any ad-hoc convenience annotations in our work, though the implementation code does not prevent this.

The ADG is described formally and in the vernacular as follows:

$$
\begin{aligned}
&\text{Let } \mathcal{N} \text{ be a set of named attributes,}\\
&\mathcal{P}(\mathcal{N}) = \{S \mid S \subseteq \mathcal{N}\}\\
&\mathcal{V} \text{ be the set of vertices,}\\
&\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \text{ be the set of directed edges.}\\
&\text{An ADG is a 5-tuple } \mathcal{G} = \big(\mathcal{V}, \mathcal{E}, \phi_V, \phi_E, \phi_G\big),\\
&\phi_V : \mathcal{V} \to \mathcal{P}(\mathcal{N}),\\
&\phi_E : \mathcal{E} \to \mathcal{P}(\mathcal{N}),\\
&\phi_G : \{\bullet\} \to \mathcal{P}(\mathcal{N}), \qquad (7.1)\\
&\text{where}\\
&\quad \phi_V \text{ maps each vertex } v \in \mathcal{V}\\
&\qquad \text{to a subset of named attributes in } \mathcal{N},\\
&\quad \phi_E \text{ maps each edge } e \in \mathcal{E}\\
&\qquad \text{to a subset of named attributes in } \mathcal{N},\\
&\quad \phi_G \text{ assigns a set of global attributes}\\
&\qquad \text{to the graph as a whole (indicated by } \{\bullet\}\text{).}
\end{aligned}
$$

8 SD Model Import

Vensim ".mdl" files were translated into wolfram language syntax using a recursive descent parser algorithm (Aho et al., 1986) with arbitrary look ahead. Two passes were used: The first converted from Vensim syntax and semantics into wolfram language syntax and Vensim semantics; the second converted the first pass to wolfram language semantics. Experimentation was sometimes necessary to determine details of the syntax or semantics.

Only the mathematical essentials were imported: Units, comments and graphical elements were ignored. The imported model can be integrated numerically, manipulated symbolically, and analyzed graphically.

It is future work to implement array variables.

The Vensim variable name character set and structure is much more permissive than those in Wolfram Language, so variables were represented as constant functions with string parameter, e.g. $v[\text{"}\$\$\ variable\quad name\text{"}]$.

We give symbols short names, so that the mathematical structure is not obscured. s_i (or si if we are lazy) is used for stocks, v_i for auxiliary variables, l_i for tables (the more obvious "t" is already used to represent simulation time), f_i for stock flows (these are defined as the integrand of the stock. We make these distinct because it is sometimes useful to have a concrete symbol for the flow of a stock), and c_i for constants, where a constant is defined as an auxiliary variable that is assigned a constant numerical value.

Tables in Vensim are a numerical construct specifying a first-order interpolation where the argument is within the domain of the table, and zero-order extrapolation (typically with warning messages issued) for arguments outside the domain of the table: i.e. the first value extends to $-\infty$, and the last value extends to $+\infty$.

For numerical model execution, tables are converted to native numerical interpolation functions in Mathematica. The resulting function has numerical semantics identical to Vensim for interpolation, but will fail with a message if used for extrapolation. If extrapolation semantics are required, the interpolation table is padded with values at $-\infty$ and $+\infty$ that will cause zero-order behavior on extrapolation without generating runtime warning messages. It is useful to identify instances of extrapolation because they indicate possible weaknesses in the model specification. Note that extrapolation does not change the Image of the lookup, but it does extend the domain to $i\infty$ and $+\infty$.

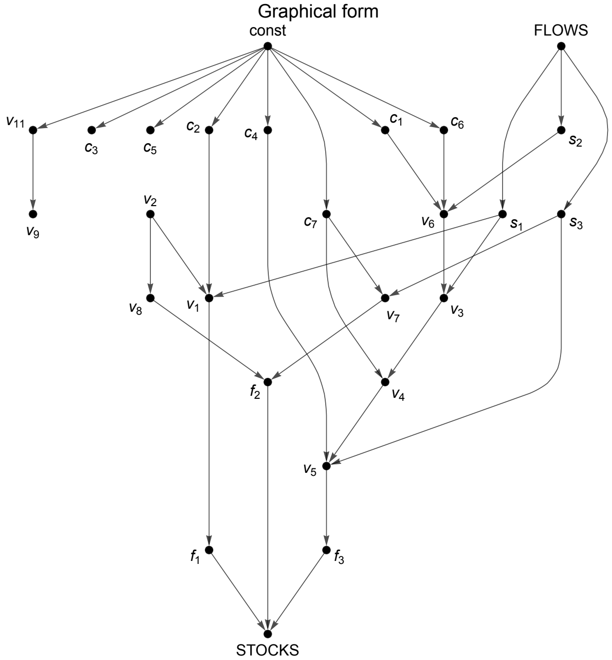
For symbolic analysis, tables are converted to a piecewise representation and all approximate numerical constants in the model are converted to exact rational form. This is because approximate numbers are a numerical concept, rational numbers are an analytic construct that facilitate symbolic manipulations. Note also that we provide both interpolation form, which will reduce to Indeterminate if the domain is violated, and an extrapolation form that is a total function that extends the first value towards $-\infty$ and the last value towards $+\infty$.

A model may be expressed as a directed graph by constructing a directed edge $v_i \to v_j$ for each independent variable v_j in the equation for variable v_i.

A model may be expressed as a directed acyclic graph by removing all edges $f_i \to s_i$ from the directed graph, which breaks all loops between a flow and its stock.

The imported mathematical and graphical representation of the Inventory Workforce Model is shown in Figure: Inventory Workforce Model

(a)
Graphical form



(b)
Integral form

$$s_1[t] == \int_{c_5}^{t} f_1[ti]\, dti + v_2[c_5]$$
$$s_2[t] == 100 + \int_{c_5}^{t} f_2[ti]\, dti$$
$$s_3[t] == 50 + \int_{c_5}^{t} f_3[ti]\, dti$$
$$f_1[t] == v_1[t]$$
$$f_2[t] == v_7[t] - v_8[t]$$
$$f_3[t] == v_5[t]$$
$$v_1[t] == \frac{-s_1[t] + v_2[t]}{c_2}$$
$$v_2[t] == 100 + \left(\begin{cases} 0 & t < 5 \\ 10 & \text{True} \end{cases} \right)$$
$$v_3[t] == s_1[t] + v_6[t]$$
$$v_4[t] == \frac{v_3[t]}{c_7}$$
$$v_5[t] == \frac{-s_3[t] + v_4[t]}{c_4}$$
$$v_6[t] == \frac{c_1 - s_2[t]}{c_6}$$
$$v_7[t] == c_7\, s_3[t]$$
$$v_8[t] == v_2[t]$$
$$v_9[t] == v_{11}[t]$$
$$v_{10}[t] == t$$
$$v_{11}[t] == 0.01$$
$$c_1 == 100$$
$$c_2 == 2$$
$$c_3 == 30$$
$$c_4 == 4$$
$$c_5 == 0$$
$$c_6 == 2$$
$$c_7 == 2$$

c_1 == v[desired inventory]
c_2 == v[expec time]
c_3 == v[FINAL TIME]
c_4 == v[hring time]
c_5 == v[INITIAL TIME]
c_6 == v[inventory correction time]
c_7 == v[productivity]
f_1 == v[expected demand_flow]
f_2 == v[Inventory_flow]
f_3 == v[Work Force_flow]
s_1 == v[expected demand]
s_2 == v[Inventory]
s_3 == v[Work Force]
v_1 == v[chng in exp dem]
v_2 == v[demand]
v_3 == v[desired production]
v_4 == v[desired work force]
v_5 == v[hring laying off]
v_6 == v[inventoryry corrections]
v_7 == v[production]
v_8 == v[sales]
v_9 == v[SAVEPER]
v_{10} == v[Time]
v_{11} == v[TIME STEP]

(c)
Differential form

$$s_1'[\tau] == \frac{1}{2}\left(100 + \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) - s_1[\tau]\right)$$
$$s_2'[\tau] == -100 - \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) + 2\, s_3[\tau]$$
$$s_3'[\tau] == \frac{1}{4}\left(\frac{1}{2}\left(s_1[\tau] + \frac{1}{2}(100 - s_2[\tau])\right) - s_3[\tau]\right)$$
$$s_1[0] == 100$$
$$s_2[0] == 100$$
$$s_3[0] == 50$$

```
Model: InventoryWorkforce.mdl githash 1449b3b
Stock initialization dependencies elided.
Loops in (a) broken at fi->si.
Long names replaced with short names.
Implicit time variable not shown.
```

Figure 8.1: Internal mathematical and graphical representation of Inventory Workforce Model (Mojtahedzadeh, n.d., supplementry material)

9 Analytical Conditions of Loop Polarity Reversal

9.1 Summary

System-dynamics authors have long cautioned that a causal-link polarity should never flip sign. Sterman (J. Sterman, 2000, ch. 5, p. 146) states flatly that "all links should have unambiguous polarities," Richardson and Pugh (Richardson & Pugh, 1981, p. 259) label polarity reversal a "serious bug," and Goodman (M. R. Goodman, 1974) likewise advises modelers to avoid it. Because a reversal undermines the behavioral intuition encoded in the diagram, detecting and eliminating such errors is a priority.

In practice, identification has relied almost entirely on simulation: the modeler sweeps over many initial conditions and watches for sign changes. This brute-force search can miss *latent* reversals that occur only in rare regions of state space, and it offers little insight into *why* a reversal appears. In theory one could inspect the governing equations by differentiating the influence functions, but that analytic route is seldom taken in day-to-day modeling work.

We argue that modern symbolic-mathematics tools now make the analytic route practical. By computing the sign of each causal derivative symbolically, we can derive explicit algebraic conditions under which a link flips. Those conditions immediately suggest constraints for model refinement, eliminating the dangerous states rather than merely detecting them after the fact. Our method is to apply the procedure across a corpus of models, record how many reversals we uncover, how many failures to solve the equations we encounter, and measure the computational cost in each case.

9.2 Abstract

Ambiguous link polarities in System Dynamics models impede human comprehension of the model and often represent bugs in the model, as good practice dictates that ambiguous polarities are to be avoided. We present a novel and practical automated method for computing conditions of link polarity reversal. We begin with the standard mathematical definition of link polarity $\mathrm{SGN}(\partial_{\mathrm{in}_t} \mathrm{out}_t)$, which we resolve for the state conditions under which $\partial > 0$, $\partial = 0$, and $\partial < 0$. A rubric is finally used to assign the polarity as +, -, or $\pm$ based on those conditions. The calculations are performed symbolically, and so the result is the model state conditions under which each polarity condition holds. A polarity of $\pm$ indicates that the polarity is ambiguous, as there are states that will yield both polarities. Additional constraints can be asserted (e.g. domain of constants or variables) to limit the number of false positives.

The math itself is tedious but not novel. Our contribution is some evidence that it is practical to do this calculations on an arbitrary model in a fully automated fashion.

9.3 Introduction

We define link polarity as $\mathrm{SGN}(\partial_{\mathrm{in}_t} \mathrm{out}_t)$ (Richardson, 1995). We define polarity reversal as the conditions under which the link polarity will change. Conditions of polarity reversal are computed by constructing relational expressions of each partial derivative ∂ of every variable in the model in terms of$\partial > 0, \partial = 0, \partial < 0$ and reducing each expression to all conditions for which it is always true, always false, or depends upon variable values. The resulting set of conditions is processed through a rubric to yield the polarity as +, -, or $\pm$.

A loop with an ambiguous link polarity cannot be labeled either Reinforcing or Balancing, as it may be either, depending upon state values (It is possible for two ambiguous link polarities in a path to resolve to a single polarity, but we do not explore that situation further here). Ambiguous polarities are generally to be avoided sterman2000 §"All Links

Should Have Unambiguous Polarities", (Richardson & Pugh, 1981) p. 259.

For example, if the auxiliary variable equation is $v_2 == (c_1 c_5 s_1 s_5)/c_1 1$, one of its partials is $\partial_{s_1} v_2 == (c_1 c_5 s_5)/c_1 1$ and its reduction of $(c_1 c_5 s_5)/c_1 1 > 0$ is $s_5 > 0$, assuming the constraint that all of the constants are > 0. The polarity of this link is $\pm$ (Ambiguous), because it depends on the SGN of $s5$. If we added the assumed constraint that $s_5 > 0$, then the link polarity would be $+$.

Assumed constraints can be added manually to refine polarity determination, which is useful because models often do not formally specify such constraints. Assumed constraints can be added semi-automatically, typically deduced from neighboring stocks and variables and that the range of a constant has the same SGN as the value assigned to the constant. Ideally necessary assumed constraints would be specified in the model itself, but we found this is done infrequently. We have found that adding constraints on stock values $s_i > 0$ often resolves many otherwise ambiguous link polarities. We used heuristics to determine such constraints, e.g. "The stock is named 'population', and that probably can't be negative…". Future work is to find sound methods of determining such constraints.

The set of all possible output values of the lookup (formally, the Image (Renze, 2020) of the lookup function) of all lookup functions may be automatically added to the assumed constraints. This generally improved the fidelity of the computed polarities with modeler-assigned polarities, suggesting that experienced modelers extend the pure definition of polarity as $\mathrm{SGN}(\partial_{\mathrm{in}_t} \mathrm{out}_t)$ in practice.

A typical analysis, in this case of a Bass Diffusion Model (Figure 9.1) (Bass Diffusion Model Math (Figure 9.2)) ((Rahmandad & Sterman, 2012), 2012 Supplementary Materials) is shown in Bass Diffusion Polarities (Table 9.1). The first three polarities are unambiguously "+", deduced because the last column indicates that there are no conditions under which SGN<0. The fourth polarity, for the links$s_2 \rightarrow v_3$, is ambiguous: The SGN may be +, 0, or -. A Rubric (subsection 9.4.5) presented below maps the conditions to the polarity. A detailed analysis of this model, presented below Analysis of Model (subsection 9.5.2), indicates that the modeler used knowledge of model semantics to assign the polarity.

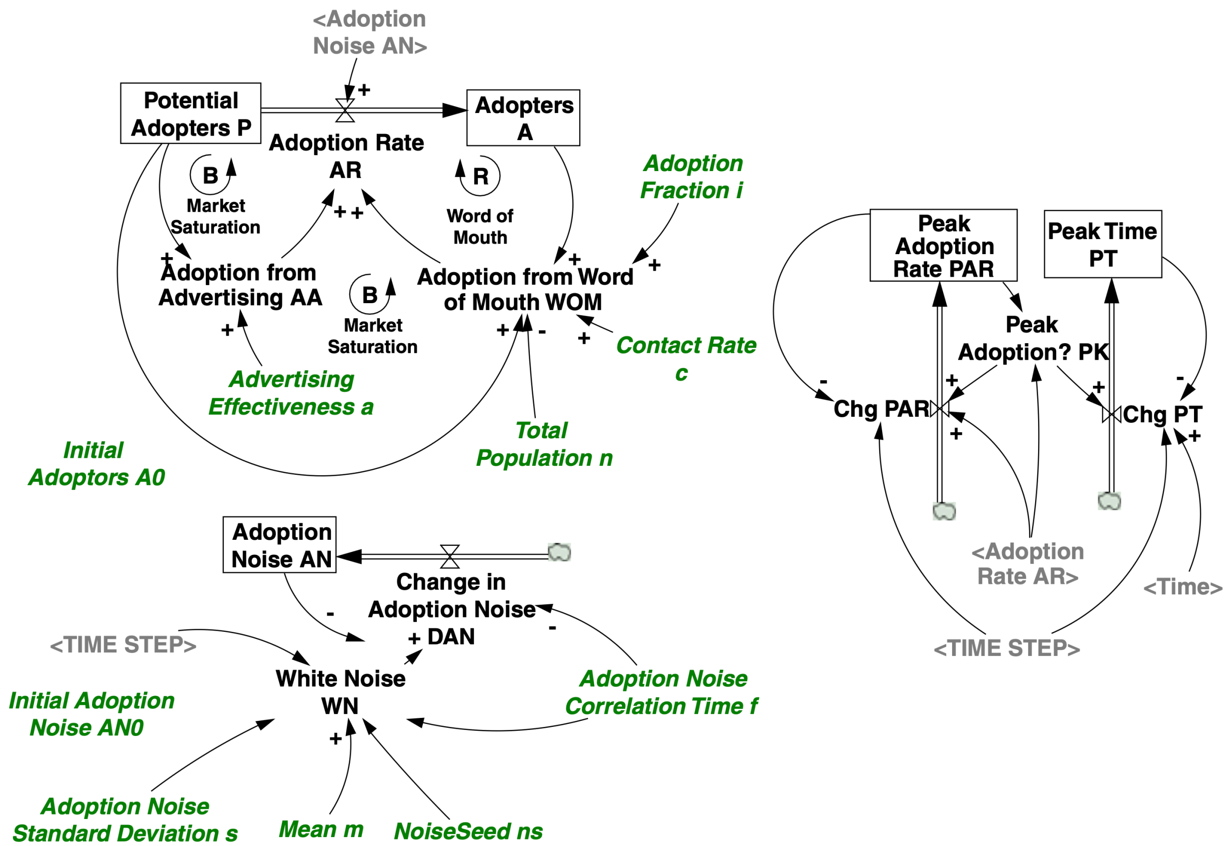


Figure 9.1: Bass Diffusion Model, from (Rahmandad & Sterman, 2012, Supplementary Materials).

Model: bass	model githash: db9472a
Symbol short names	symbol definitions
$s_1 == v[\text{Adopters A}]$	$s_1[t] == c_7 + \int_{c_8}^{t} f_1[ti]\, dti$
$s_2 == v[\text{Adoption Noise AN}]$	$s_2[t] == \int_{c_8}^{t} f_2[ti]\, dti + v_7[t]$
$s_3 == v[\text{Peak Adoption Rate PAR}]$	$s_3[t] == \int_{c_8}^{t} f_3[ti]\, dti$
$s_4 == v[\text{Peak Time PT}]$	$s_4[t] == \int_{c_8}^{t} f_4[ti]\, dti$
$s_5 == v[\text{Potential Adopters P}]$	$s_5[t] == c_{11} + \int_{c_8}^{t} f_5[ti]\, dti - s_1[t]$
$f_1 == v[\text{Adopters A\_flow}]$	$f_1[t] == v_3[t]$
$f_2 == v[\text{Adoption Noise AN\_flow}]$	$f_2[t] == v_4[t]$
$f_3 == v[\text{Peak Adoption Rate PAR\_flow}]$	$f_3[t] == v_5[t]$
$f_4 == v[\text{Peak Time PT\_flow}]$	$f_4[t] == v_6[t]$
$f_5 == v[\text{Potential Adopters P\_flow}]$	$f_5[t] == -v_3[t]$
$v_1 == v[\text{Adoption from Advertising AA}]$	$v_1[t] == c_4\, s_5[t]$
$v_2 == v[\text{Adoption from Word of Mouth WOM}]$	$v_2[t] == \frac{c_1\, c_5\, s_1[t]\, s_5[t]}{c_{11}}$
$v_3 == v[\text{Adoption Rate AR}]$	$v_3[t] == \text{Max}[0, 1 + s_2[t]]\, (v_1[t] + v_2[t])$
$v_4 == v[\text{Change in Adoption Noise DAN}]$	$v_4[t] == \frac{-s_2[t] + v_{12}[t]}{c_2}$
$v_5 == v[\text{Chg PAR}]$	$v_5[t] == \frac{(-s_3[t] + v_3[t])\, v_8[t]}{v_{11}[t]}$
$v_6 == v[\text{Chg PT}]$	$v_6[t] == \frac{v_8[t]\, (-s_4[t] + v_{10}[t])}{v_{11}[t]}$
$v_7 == v[\text{Initial Adoption Noise AN0}]$	$v_7[t] == c_9 + c_3\, v[\text{RANDOM NORMAL}][-10, 10, 0, 1, c_{10}]$
$v_8 == v[\text{"Peak Adoption? PK"}]$	$v_8[t] == \left(\begin{cases} 1 & v_3[t] > s_3[t] \\ 0 & \text{True} \end{cases} \right)$
$v_9 == v[\text{SAVEPER}]$	$v_9[t] == v_{11}[t]$
$v_{10} == v[\text{Time}]$	$v_{10}[t] == t$
$v_{11} == v[\text{TIME STEP}]$	$v_{11}[t] == 0.0078125$
$v_{12} == v[\text{White Noise WN}]$	$v_{12}[t] == c_9 + \left(\frac{c_2\, c_3^2 \left(2 - \frac{v_{11}[t]}{c_2}\right)}{v_{11}[t]} \right)^{0.5} v[\text{RANDOM NORMAL}][-10, 10, 0, 1, c_{10}]$
$c_1 == v[\text{Adoption Fraction i}]$	$c_1 == 0.015$
$c_2 == v[\text{Adoption Noise Correlation Time f}]$	$c_2 == 2$
$c_3 == v[\text{Adoption Noise Standard Deviation s}]$	$c_3 == 0.1$
$c_4 == v[\text{Advertising Effectiveness a}]$	$c_4 == 0.01$
$c_5 == v[\text{Contact Rate c}]$	$c_5 == 100$
$c_6 == v[\text{FINAL TIME}]$	$c_6 == 25$
$c_7 == v[\text{Initial Adoptors A0}]$	$c_7 == 1$
$c_8 == v[\text{INITIAL TIME}]$	$c_8 == 0$
$c_9 == v[\text{Mean m}]$	$c_9 == 0$
$c_{10} == v[\text{NoiseSeed ns}]$	$c_{10} == 1$
$c_{11} == v[\text{Total Population n}]$	$c_{11} == 1. \times 10^6$

Figure 9.2: Internal mathematical representation of Bass Diffusion Model from (Rahmandad & Sterman, 2012, Supplementary Materials)

Table 9.1

model:bass; model githash:db9472a

Assumptions: $s_1 \geq 0 \,\&\&\, s_3 \geq c_7 \,\&\&\, s_4 \geq 0 \,\&\&\, s_5 \geq 0 \,\&\&\, -10 \leq rn[-10, 10] \leq 10 \,\&\&\, v_{11} > 0 \,\&\&\, v_8 \geq 0 \,\&\&\, c_1 > 0 \,\&\&\, c_2 > 0 \,\&\&\, c_3 > 0 \,\&\&\, c_4 > 0 \,\&\&\, c_5 > 0 \,\&\&\, c_6 > 0 \,\&\&\, c_7 > 0 \,\&\&\, c_8 > 0 \,\&\&\, c_9 > 0 \,\&\&\, c_{10} > 0 \,\&\&\, c_{11} > 0$

variable	Extrap?	∂	SGN	>0	>=0	==0	!=0	<=0	<0
$v_1 == c_4\, s_5$	True	$\partial_{s_5} v_1 == c_4$	+	True	True	False	True	False	False
$v_2 == \frac{c_1\, c_5\, s_1\, s_5}{c_{11}}$	True	$\partial_{s_1} v_2 == \frac{c_1\, c_5\, s_5}{c_{11}}$	+	$s_5 > 0$	True	$s_5 == 0$	$s_5 \neq 0$	$s_5 \leq 0$	False
	True	$\partial_{s_5} v_2 == \frac{c_1\, c_5\, s_1}{c_{11}}$	+	$s_1 > 0$	True	$s_1 == 0$	$s_1 \neq 0$	$s_1 \leq 0$	False
$v_3 == (v_1 + v_2)\ \mathrm{Max}[0,\, 1 + s_2]$	True	$\partial_{s_2} v_3 == (v_1 + v_2) \left(\begin{cases} 0 & s_2 < -1 \\ 1 & s_2 > -1 \\ ¿ & \mathrm{True} \end{cases} \right)$	±	$s_2 > -1 \,\&\&\, v_1 + v_2 > 0$	$1 + s_2 \leq 0 \vert\vert v_1 + v_2 \geq 0$	$1 + s_2 \leq 0 \vert\vert v_1 + v_2 == 0$	$s_2 > -1 \,\&\&\, v_1 + v_2 \neq 0$	$1 + s_2 \leq 0 \vert\vert v_1 + v_2 \leq 0$	$s_2 > -1 \,\&\&\, v_1 + v_2 < 0$
	True	$\partial_{v_1} v_3 == \mathrm{Max}[0,\, 1 + s_2]$	+	$s_2 > -1$	True	$s_2 \leq -1$	$s_2 > -1$	$s_2 \leq -1$	False
	True	$\partial_{v_2} v_3 == \mathrm{Max}[0,\, 1 + s_2]$	+	$s_2 > -1$	True	$s_2 \leq -1$	$s_2 > -1$	$s_2 \leq -1$	False
$v_4 == \frac{-s_2 + v_{12}}{c_2}$	True	$\partial_{s_2} v_4 == -\frac{1}{c_2}$	−	False	False	False	True	True	True
	True	$\partial_{v_{12}} v_4 == \frac{1}{c_2}$	+	True	True	False	True	False	False
$v_5 == \frac{(-s_3 + v_3)\, v_8}{v_{11}}$	True	$\partial_{s_3} v_5 == -\frac{v_8}{v_{11}}$	−	False	$v_8 == 0$	$v_8 == 0$	$v_8 \neq 0$	True	$v_8 > 0$
	True	$\partial_{v_3} v_5 == \frac{v_8}{v_{11}}$	+	$v_8 > 0$	True	$v_8 == 0$	$v_8 \neq 0$	$v_8 == 0$	False
	True	$\partial_{v_8} v_5 == \frac{-s_3 + v_3}{v_{11}}$	±	$v_3 > s_3$	$s_3 \leq v_3$	$s_3 == v_3$	$s_3 \neq v_3$	$s_3 \geq v_3$	$v_3 < s_3$
	True	$\partial_{v_{11}} v_5 == -\frac{(-s_3 + v_3)\, v_8}{{v_{11}}^2}$	±	$v_3 < s_3 \,\&\&\, v_8 > 0$	$s_3 \geq v_3 \vert\vert v_8 \leq 0$	$s_3 == v_3 \vert\vert v_8 == 0$	$s_3 \neq v_3 \,\&\&\, v_8 \neq 0$	$s_3 \leq v_3 \vert\vert v_8 \leq 0$	$v_3 > s_3 \,\&\&\, v_8 > 0$
$v_6 == \frac{v_8\, (-s_4 + v_{10})}{v_{11}}$	True	$\partial_{v_8} v_6 == \frac{-s_4 + v_{10}}{v_{11}}$	±	$v_{10} > s_4$	$s_4 \leq v_{10}$	$s_4 == v_{10}$	$s_4 \neq v_{10}$	$s_4 \geq v_{10}$	$v_{10} < s_4$
	True	$\partial_{s_4} v_6 == -\frac{v_8}{v_{11}}$	−	False	$v_8 == 0$	$v_8 == 0$	$v_8 \neq 0$	True	$v_8 > 0$
	True	$\partial_{v_{10}} v_6 == \frac{v_8}{v_{11}}$	+	$v_8 > 0$	True	$v_8 == 0$	$v_8 \neq 0$	$v_8 == 0$	False
	True	$\partial_{v_{11}} v_6 == -\frac{v_8\, (-s_4 + v_{10})}{{v_{11}}^2}$	±	$v_{10} < s_4 \,\&\&\, v_8 > 0$	$s_4 \geq v_{10} \vert\vert v_8 \leq 0$	$s_4 == v_{10} \vert\vert v_8 == 0$	$s_4 \neq v_{10} \,\&\&\, v_8 \neq 0$	$s_4 \leq v_{10} \vert\vert v_8 \leq 0$	$v_8 > 0 \,\&\&\, v_{10} > s_4$
$v_8 == \left(\begin{cases} 1 & v_3 > s_3 \\ 0 & \mathrm{True} \end{cases} \right)$	True	$\partial_{v_3} v_8 == 0$	Z	False	True	True	False	True	False
	True	$\partial_{s_3} v_8 == 0$	Z	False	True	True	False	True	False
$v_9 == v_{11}$	True	$\partial_{v_{11}} v_9 == 1$	+	True	True	False	True	False	False
$v_{12} == \left(c_9 + \sqrt{\frac{c_2\, {c_3}^2 \left(2 - \frac{v_{11}}{c_2}\right)}{v_{11}}}\right) rn[-10, 10]$	True	$\partial_{v_{11}} v_{12} == \frac{\left(-\frac{{c_3}^2}{v_{11}} - \frac{c_2\, {c_3}^2 \left(2 - \frac{v_{11}}{c_2}\right)}{{v_{11}}^2}\right) rn[-10, 10]}{2 \sqrt{\frac{c_2\, {c_3}^2 \left(2 - \frac{v_{11}}{c_2}\right)}{v_{11}}}}$	±	$rn[-10, 10] < 0$	$rn[-10, 10] \leq 0$	$rn[-10, 10] == 0$	$rn[-10, 10] \neq 0$	$rn[-10, 10] \geq 0$	$rn[-10, 10] > 0$

9.4 Method

9.4.1 Import Vensim Model

The Vensim Bass Diffusion Model (Figure 9.2) ((Rahmandad & Sterman, 2012), 2012 Supplementary Materials) is imported using the methods defined in SD Model Import (chapter 8). Table functions are further processed as defined in Table Function Special Processing (subsection 9.4.2).

9.4.2 Table Function Special Processing

Tables in Vensim are a numerical construct specifying a first-order interpolation where the argument is within the domain of the table, and zero-order extrapolation (typically with warning messages issued) for arguments outside the domain of the table: i.e. the first value extends to $-\infty$, and the last value extends to $+\infty$.

For numerical model execution, tables are converted to native numerical interpolation functions in Mathematica (our implementation language). The resulting function has numerical semantics identical to Vensim for interpolation, but will fail with a message if used for extrapolation. If extrapolation semantics are required, the interpolation table is padded with values at $-\infty$ and $+\infty$ that will cause zero-order behavior (matching Vensim specifications) on extrapolation without generating runtime warning messages. Instances of extrapolation are useful to identify because they indicate possible weaknesses in the model specification. Note that extrapolation does not change the Image of the lookup, but it does extend the domain to $(-\infty, +\infty)(\text{aka}\mathbb{R})$.

For symbolic analysis, tables are converted to a piecewise representation. Simple Urban table example (Figure 9.3) shows the table “Effect of Land Availability on Housing Construction” from a simple urban model (Ghaffarzadegan et al., 2011). Note all approximate numbers have been converted to exact rational form. This is because approximate numbers are a numerical concept, rational numbers are an analytic construct that facilitate symbolic manipulations. Note also that we provide both interpolation form, which will reduce to Indeterminate if the domain is violated, and an extrapolation form that is a total function that extends the first value towards $-\infty$ and the last value towards $+\infty$.

$$l_5[t] == \text{interpolate}\left[\left\{\{0,1\}, \left\{\frac{1}{10}, 1\right\}, \ll 8 \gg, \{1,0\}\right\}\right][v_{13}[t]]$$

No Extrapolation:

$$l_5[t] == \left(\begin{cases} 1 & 0 \leq v_{13}[t] \leq \frac{3}{5} \\ \frac{339}{100} - \frac{17\, v_{13}[t]}{5} & \frac{4}{5} < v_{13}[t] \leq \frac{9}{10} \\ \frac{11}{4} - \frac{13\, v_{13}[t]}{5} & \frac{7}{10} < v_{13}[t] \leq \frac{4}{5} \\ \frac{1}{50}\,(71 - 35\, v_{13}[t]) & \frac{3}{5} < v_{13}[t] \leq \frac{7}{10} \\ -\frac{33}{10}\,(-1 + v_{13}[t]) & \frac{9}{10} < v_{13}[t] \leq 1 \\ \text{Indeterminate} & \text{True} \end{cases}\right)$$

Extrapolation:

$$l_5[t] == \left(\begin{cases} 1 & 5\, v_{13}[t] \leq 3 \\ \frac{339}{100} - \frac{17\, v_{13}[t]}{5} & \frac{4}{5} < v_{13}[t] \leq \frac{9}{10} \\ \frac{11}{4} - \frac{13\, v_{13}[t]}{5} & \frac{7}{10} < v_{13}[t] \leq \frac{4}{5} \\ \frac{1}{50}\,(71 - 35\, v_{13}[t]) & \frac{3}{5} < v_{13}[t] \leq \frac{7}{10} \\ -\frac{33}{10}\,(-1 + v_{13}[t]) & \frac{9}{10} < v_{13}[t] \leq 1 \\ 0 & \text{True} \end{cases}\right)$$

Figure 9.3

9.4.3 Partial Derivatives

Partial derivatives (∂) are computed symbolically using the native capabilities of Wolfram Mathematica (Wolfram Research, 2023b). A simple example is shown in Partial Derivative Example (Figure 9.4).

Partials are propagated through stocks by using mixed partials (Weisstein, 2008). Partials through macros (e.g. SMOOTH3) are either computed exactly, or simply asserted based on prior analysis of the behavior of the macro.

$$v_5[t] == \frac{(-s_3[t]+v_3[t])\, v_8[t]}{v_{11}[t]}$$

$$\partial_{s_3[t]} v_5[t] == -\frac{v_8[t]}{v_{11}[t]}$$

$$\partial_{v_3[t]} v_5[t] == \frac{v_8[t]}{v_{11}[t]}$$

$$\partial_{v_8[t]} v_5[t] == \frac{-s_3[t]+v_3[t]}{v_{11}[t]}$$

$$\partial_{v_{11}[t]} v_5[t] == -\frac{(-s_3[t]+v_3[t])\, v_8[t]}{v_{11}[t]^2}$$

Figure 9.4

9.4.4 Reduction

We define the reduction of an inequality as the space for which the inequality is satisfied. For example, the reduction of $x > 0 \wedge x^2 + y^2 < 1$ is $0 < x < 1 \wedge -\sqrt(1-x^2) < y < \sqrt(1-x^2)$. Any value of x and y that satisfies the reduced set of inequalities will, by definition, satisfy the original inequality. The crux of our method is reducing inequalities of the partials ($\partial > 0, \partial \geq 0, \partial = 0, \partial \neq 0, \partial \leq 0, \partial < 0$), to the space over which the reduction is satisfied always ($\forall$), and satisfied sometimes ($\exists$). The resulting logical expressions will each be either True, or False, or conditioned on the values of variables. These combinations are compared through a rubric to determine, for each of the polarities, if it is always true, never true, or sometimes true. If it is always > 0 then the polarity must be "+". If it is sometimes > 0 and sometimes < 0, then the polarity reverses under some conditions and is ambiguous $\pm$.

Any inequality can in theory be reduced, but in practice the computation can fail from resource or patience exhaustion. We have found that branch cuts induced by tables with many piecewise elements can cause such practical failures.

Even if the equation can be reduced, the results may consume many pages of manuscript.

Discontinuities, typically between segments in a piecewise function, cause many additional terms in the constraint equations. We generally discard such constraints by asserting the exclusions into the assumptions, because they can usually be ignored for practical purposes. Singularities, typically caused by divide by zero, can also induce additional terms. We note these as possible weaknesses in the model, but do not remove them because we usually can't prove the condition will or will not occur.

9.4.5 Rubric

The rubric is used to convert the reduced partials into polarities. The partial is reduced for all possible conditions ($\partial > 0, \partial \geq 0, \partial = 0, \partial \neq 0, \partial \leq 0, \partial < 0$), each of which will reduce to True, False, or a conditional equation in terms of the independent variables of the partial under which the condition is satisfied. True indicates the condition MUST be satisfied for all values of the independent variables in the partial. False indicates it CAN NOT be satisfied for any values of the independent variables. A conditional equation indicates the condition CAN be satisfied by certain values of the independent variables.

The rubric is interpreted to indicate that the polarity is as indicated in the left column if the reduction of the partial for all conditions matches the indicated row. The $\forall$ columns match True or False in the reduced partial. The $\exists$columns w/ True match a conditional reduction in the reduced partial.

For example, if the $\forall > 0$ condition is True, then the polarity must be positive and can not be zero or negative. If on the other hand the $\forall > 0$ is False, and the $\exists > 0$ is True, then the polarity could be positive but may also be zero and/or negative.

We reduce for all conditions, though only three ($\partial > 0, \partial = 0, \partial < 0$) are required. We do this as a redundancy check on the rather complicated and intricate reduction algorithm. Any combinations of results other than those in the rubric indicate an inconsistent result and so a failure of the reduction algorithm. Failures are a result of implementation errors, not mathematical in origin.

Table 9.2

conditions	$\forall_\delta\ \delta > 0$	$\forall_\delta\ \delta \geq 0$	$\forall_\delta\ \delta == 0$	$\forall_\delta\ \delta \leq 0$	$\forall_\delta\ \delta < 0$	$\exists_\delta\ \delta > 0$	$\exists_\delta\ \delta \geq 0$	$\exists_\delta\ \delta == 0$	$\exists_\delta\ \delta \leq 0$	$\exists_\delta\ \delta < 0$
$\delta > 0$	True	True	False	False	False	True	True	False	False	False
$\delta == 0$	False	True	True	True	False	False	True	True	True	False
$\delta < 0$	False	False	False	True	True	False	False	False	True	True
$\delta > 0 \Vert \delta == 0$	False	True	False	False	False	True	True	True	True	False
$\delta > 0 \Vert \delta < 0$	False	False	False	False	False	True	True	False	True	True
$\delta == 0 \Vert \delta < 0$	False	False	False	True	False	False	True	True	True	True
$\delta > 0 \Vert \delta == 0 \Vert \delta < 0$	False	False	False	False	False	True	True	True	True	True

9.5 Results

9.5.1 Imported Model

The imported model is shown in Bass Diffusion Model (Figure 9.1) and Bass Diffusion Model Math (Figure 9.2).

9.5.2 Analysis of Model

A link-polarity analysis of the Bass Diffusion model is shown in Bass Polarity Analysis Results (Table 9.3). The columns in that table are interpreted as follows:

Variable The short name of the variable together with its definition. (The original name can be found in the model import; cf. Imported model (**??**) for this example.)

∂ The partial derivative of the variable with respect to each independent variable and stock appearing in its definition.

SGN The polarity of the link, under the assumptions in row 2, as it would be marked on the System-Dynamics diagram, augmented with additional conditions of interest:

- $+$ SGN always ≥ 0.
- $-$ SGN always ≤ 0.
- Z SGN identically 0.
- $\pm$ SGN may take either sign.

$>$ 0 Conditions under which the polarity is > 0. True means always positive; False means never positive; otherwise a mathematical condition (subject to the stated assumptions) is supplied.

$=$ 0 Analogue of the previous column for equality to 0.

$<$ 0 Analogue of the previous column for strict negativity.

A comparison of polarities as assigned by our method, and those assigned by the author of the model, are shown in Bass Polarity Differences (Figure 9.5) marked in red. These are discussed later.

The partial derivatives are computed from an algebraic representation of each variable definition. Several translations were found to be necessary for this particular model, suggesting that exercising our approach on a number of models (future work) will be necessary before it is ready for use by system analysts.

1. The VENSIM construct `MAX`, a numerical operator, was rewritten as a piece- wise mathematical function. Such conversions may reveal discontinuities in the partials; here a discontinuity occurs at $s_2 = 1$, giving an indeterminate derivative.

2. The model employs the special function `RANDOM NORMAL`, which generates clipped normal variates. For reduction we automatically assume that the variate lies within the clip region, $[-10, 10]$ in this case.

3. All constants were constrained to be strictly positive (> 0), a reasonable assumption given the model semantics. Assigning the exact numeric values would weaken the analysis, because the consequences of altering those constants would be hidden.

Our computed polarities are overlaid on the published polarities in Bass Polarity Differences (Figure 9.5). Three discrepancies between the published labels and those obtained by our algorithm were observed:

$$\partial_{s_2} v_3, \qquad \partial_{v_8} v_5, \qquad \partial_{v_8} v_6.$$

The reasons are:

1. $\partial_{s_2} v_3$: Bass Polarity Analysis Results (Table 9.3) shows this loop is positive, as labelled, *iff* $v_1 > v_2$. Because we did not assert that condition, we mark the link as $\pm$.

2. $\partial_{v_8} v_5$: The loop appears to have been labelled according to the polarity it exhibits when it is "active" (defined *ad hoc* here as contributing a non-zero flow to s_3 via f_3) under the condition $v_8 = 1$ and $v_3 \geq s_3$. Our definition of link polarity does not account for such semantics and therefore assigns $\pm$, depending only on the relationship between v_3 and s_3.

3. $\partial_{v_8} v_6$: Analogous to $\partial_{v_8} v_5$.

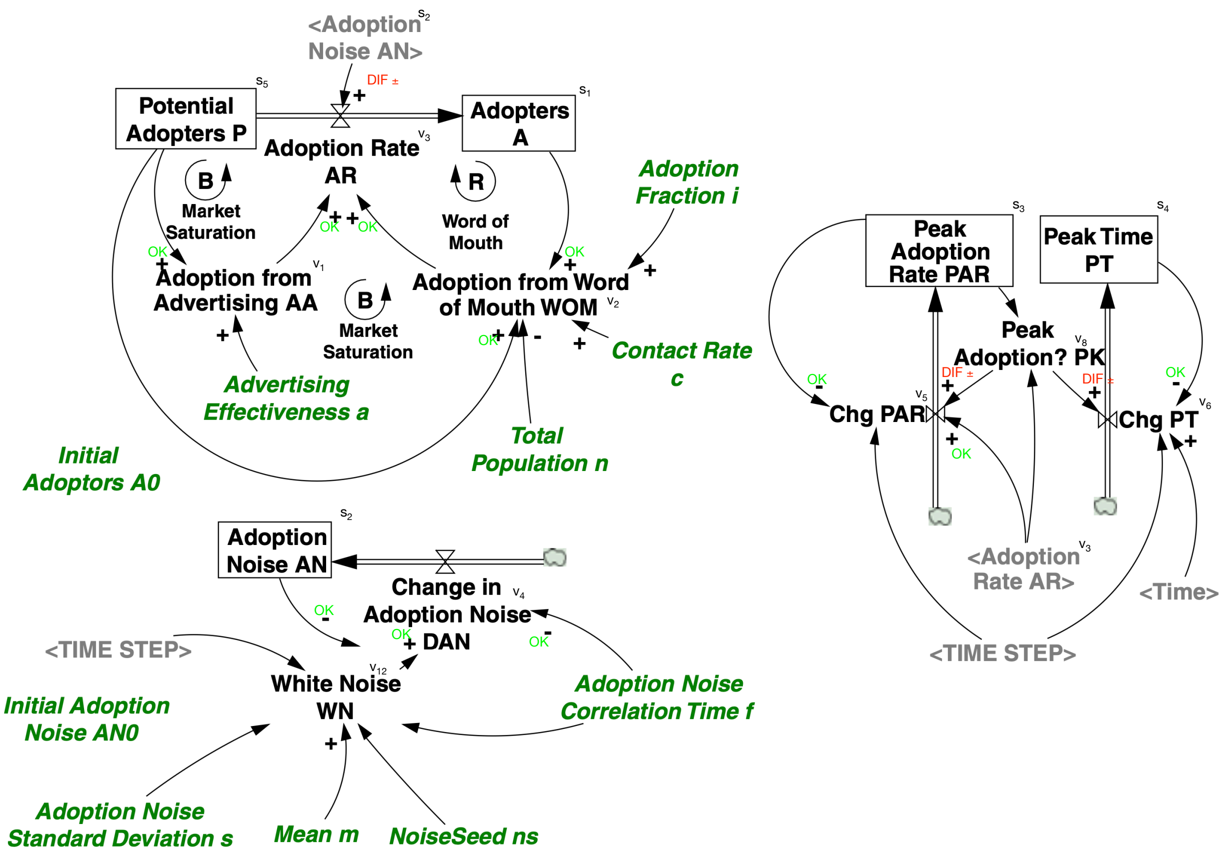


Figure 9.5: Bass Diffusion Model from (Rahmandad & Sterman, 2012, Supplementary Materials). Green indicates polarities that matched our results. Red indicates conflicts with our results.

Table 9.3: Bass Diffusion Model, analytic conditions of polarity reversals.

Model: BassStochastic.mdl

Model githash: db9472a

Assumptions: $s_1 \geq 0 \,\&\&\, s_3 \geq c_7 \,\&\&\, s_4 \geq 0 \,\&\&\, s_5 \geq 0 \,\&\&\, -10 \leq rn[-10, 10] \leq 10 \,\&\&\, v_{11} > 0 \,\&\&\, v_8 \geq 0 \,\&\&\, c_1 > 0 \,\&\&\, c_2 > 0 \,\&\&\, c_3 > 0 \,\&\&\, c_4 > 0 \,\&\&\, c_5 > 0 \,\&\&\, c_6 > 0 \,\&\&\, c_7 > 0 \,\&\&\, c_8 > 0 \,\&\&\, c_9 > 0 \,\&\&\, c_{10} > 0 \,\&\&\, c_{11} > 0$

Variable	Partial	SGN	>0	>=0	==0	!=0	<=0	<0
$v_1[t] == c_4\, s_5[t]$	$\partial_{s_5[t]} v_1[t] == c_4$	+	$\partial > 0 \rrbracket$ True	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket$ False	$\partial \neq 0 \rrbracket$ True	$\partial \leq 0 \rrbracket$ False	$\partial < 0 \rrbracket$ False
$v_2[t] == \frac{c_1 c_5 s_1[t]\, s_5[t]}{c_{11}}$	$\partial_{s_1[t]} v_2[t] == \frac{c_1 c_5 s_5[t]}{c_{11}}$	+	$\partial > 0 \rrbracket\ s_5[t] > 0$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ s_5[t] == 0$	$\partial \neq 0 \rrbracket\ s_5[t] \neq 0$	$\partial \leq 0 \rrbracket\ s_5[t] \leq 0$	$\partial < 0 \rrbracket$ False
$v_2[t] == \frac{c_1 c_5 s_1[t]\, s_5[t]}{c_{11}}$	$\partial_{s_5[t]} v_2[t] == \frac{c_1 c_5 s_1[t]}{c_{11}}$	+	$\partial > 0 \rrbracket\ s_1[t] > 0$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ s_1[t] == 0$	$\partial \neq 0 \rrbracket\ s_1[t] \neq 0$	$\partial \leq 0 \rrbracket\ s_1[t] \leq 0$	$\partial < 0 \rrbracket$ False
$v_3[t] == \mathrm{Max}[0, 1 + s_2[t]]\, (v_1[t] + v_2[t])$	$\partial_{s_2[t]} v_3[t] == \left(\begin{cases} 0 & s_2[t] < -1 \\ 1 & s_2[t] > -1 \\ ¿ & \mathrm{True} \end{cases}\right) (v_1[t] + v_2[t])$	±	$\partial > 0 \rrbracket\ s_2[t] > -1 \,\&\&\, v_1[t] + v_2[t] > 0$	$\partial \geq 0 \rrbracket\ 1 + s_2[t] \leq 0 \,\vert\vert\, v_1[t] + v_2[t] \geq 0$	$\partial == 0 \rrbracket\ 1 + s_2[t] \leq 0 \,\vert\vert\, v_1[t] + v_2[t] == 0$	$\partial \neq 0 \rrbracket\ s_2[t] > -1 \,\&\&\, v_1[t] + v_2[t] \neq 0$	$\partial \leq 0 \rrbracket\ 1 + s_2[t] \leq 0 \,\vert\vert\, v_1[t] + v_2[t] \leq 0$	$\partial < 0 \rrbracket\ s_2[t] > -1 \,\&\&\, v_1[t] + v_2[t] < 0$
$v_3[t] == \mathrm{Max}[0, 1 + s_2[t]]\, (v_1[t] + v_2[t])$	$\partial_{v_1[t]} v_3[t] == \mathrm{Max}[0, 1 + s_2[t]]$	+	$\partial > 0 \rrbracket\ s_2[t] > -1$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ s_2[t] \leq -1$	$\partial \neq 0 \rrbracket\ s_2[t] > -1$	$\partial \leq 0 \rrbracket\ s_2[t] \leq -1$	$\partial < 0 \rrbracket$ False
$v_3[t] == \mathrm{Max}[0, 1 + s_2[t]]\, (v_1[t] + v_2[t])$	$\partial_{v_2[t]} v_3[t] == \mathrm{Max}[0, 1 + s_2[t]]$	+	$\partial > 0 \rrbracket\ s_2[t] > -1$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ s_2[t] \leq -1$	$\partial \neq 0 \rrbracket\ s_2[t] > -1$	$\partial \leq 0 \rrbracket\ s_2[t] \leq -1$	$\partial < 0 \rrbracket$ False
$v_4[t] == \frac{-s_2[t] + v_{12}[t]}{c_2}$	$\partial_{s_2[t]} v_4[t] == -\frac{1}{c_2}$	−	$\partial > 0 \rrbracket$ False	$\partial \geq 0 \rrbracket$ False	$\partial == 0 \rrbracket$ False	$\partial \neq 0 \rrbracket$ True	$\partial \leq 0 \rrbracket$ True	$\partial < 0 \rrbracket$ True
$v_4[t] == \frac{-s_2[t] + v_{12}[t]}{c_2}$	$\partial_{v_{12}[t]} v_4[t] == \frac{1}{c_2}$	+	$\partial > 0 \rrbracket$ True	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket$ False	$\partial \neq 0 \rrbracket$ True	$\partial \leq 0 \rrbracket$ False	$\partial < 0 \rrbracket$ False
$v_5[t] == ((-s_3[t] + v_3[t])\, v_8[t]) / v_{11}[t]$	$\partial_{s_3[t]} v_5[t] == -\frac{v_8[t]}{v_{11}[t]}$	−	$\partial > 0 \rrbracket$ False	$\partial \geq 0 \rrbracket\ v_8[t] == 0$	$\partial == 0 \rrbracket\ v_8[t] == 0$	$\partial \neq 0 \rrbracket\ v_8[t] \neq 0$	$\partial \leq 0 \rrbracket$ True	$\partial < 0 \rrbracket\ v_8[t] > 0$
$v_5[t] == ((-s_3[t] + v_3[t])\, v_8[t]) / v_{11}[t]$	$\partial_{v_3[t]} v_5[t] == \frac{v_8[t]}{v_{11}[t]}$	+	$\partial > 0 \rrbracket\ v_8[t] > 0$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ v_8[t] == 0$	$\partial \neq 0 \rrbracket\ v_8[t] \neq 0$	$\partial \leq 0 \rrbracket\ v_8[t] == 0$	$\partial < 0 \rrbracket$ False
$v_5[t] == ((-s_3[t] + v_3[t])\, v_8[t]) / v_{11}[t]$	$\partial_{v_8[t]} v_5[t] == \frac{-s_3[t] + v_3[t]}{v_{11}[t]}$	±	$\partial > 0 \rrbracket\ v_3[t] > s_3[t]$	$\partial \geq 0 \rrbracket\ s_3[t] \leq v_3[t]$	$\partial == 0 \rrbracket\ s_3[t] == v_3[t]$	$\partial \neq 0 \rrbracket\ s_3[t] \neq v_3[t]$	$\partial \leq 0 \rrbracket\ s_3[t] \geq v_3[t]$	$\partial < 0 \rrbracket\ v_3[t] < s_3[t]$
$v_5[t] == ((-s_3[t] + v_3[t])\, v_8[t]) / v_{11}[t]$	$\partial_{v_{11}[t]} v_5[t] == -\frac{(-s_3[t] + v_3[t])\, v_8[t]}{v_{11}[t]^2}$	±	$\partial > 0 \rrbracket\ v_3[t] < s_3[t] \,\&\&\, v_8[t] > 0$	$\partial \geq 0 \rrbracket\ s_3[t] \geq v_3[t] \,\vert\vert\, v_8[t] \leq 0$	$\partial == 0 \rrbracket\ s_3[t] == v_3[t] \,\vert\vert\, v_8[t] == 0$	$\partial \neq 0 \rrbracket\ s_3[t] \neq v_3[t] \,\&\&\, v_8[t] \neq 0$	$\partial \leq 0 \rrbracket\ s_3[t] \leq v_3[t] \,\vert\vert\, v_8[t] \leq 0$	$\partial < 0 \rrbracket\ v_3[t] > s_3[t] \,\&\&\, v_8[t] > 0$
$v_6[t] == (v_8[t]\, (-s_4[t] + v_{10}[t])) / v_{11}[t]$	$\partial_{v_8[t]} v_6[t] == \frac{-s_4[t] + v_{10}[t]}{v_{11}[t]}$	±	$\partial > 0 \rrbracket\ v_{10}[t] > s_4[t]$	$\partial \geq 0 \rrbracket\ s_4[t] \leq v_{10}[t]$	$\partial == 0 \rrbracket\ s_4[t] == v_{10}[t]$	$\partial \neq 0 \rrbracket\ s_4[t] \neq v_{10}[t]$	$\partial \leq 0 \rrbracket\ s_4[t] \geq v_{10}[t]$	$\partial < 0 \rrbracket\ v_{10}[t] < s_4[t]$
$v_6[t] == (v_8[t]\, (-s_4[t] + v_{10}[t])) / v_{11}[t]$	$\partial_{s_4[t]} v_6[t] == -\frac{v_8[t]}{v_{11}[t]}$	−	$\partial > 0 \rrbracket$ False	$\partial \geq 0 \rrbracket\ v_8[t] == 0$	$\partial == 0 \rrbracket\ v_8[t] == 0$	$\partial \neq 0 \rrbracket\ v_8[t] \neq 0$	$\partial \leq 0 \rrbracket$ True	$\partial < 0 \rrbracket\ v_8[t] > 0$
$v_6[t] == (v_8[t]\, (-s_4[t] + v_{10}[t])) / v_{11}[t]$	$\partial_{v_{10}[t]} v_6[t] == \frac{v_8[t]}{v_{11}[t]}$	+	$\partial > 0 \rrbracket\ v_8[t] > 0$	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket\ v_8[t] == 0$	$\partial \neq 0 \rrbracket\ v_8[t] \neq 0$	$\partial \leq 0 \rrbracket\ v_8[t] == 0$	$\partial < 0 \rrbracket$ False
$v_6[t] == (v_8[t]\, (-s_4[t] + v_{10}[t])) / v_{11}[t]$	$\partial_{v_{11}[t]} v_6[t] == -\frac{v_8[t]\, (-s_4[t] + v_{10}[t])}{v_{11}[t]^2}$	±	$\partial > 0 \rrbracket\ v_{10}[t] < s_4[t] \,\&\&\, v_8[t] > 0$	$\partial \geq 0 \rrbracket\ s_4[t] \geq v_{10}[t] \,\vert\vert\, v_8[t] \leq 0$	$\partial == 0 \rrbracket\ s_4[t] == v_{10}[t] \,\vert\vert\, v_8[t] == 0$	$\partial \neq 0 \rrbracket\ s_4[t] \neq v_{10}[t] \,\&\&\, v_8[t] \neq 0$	$\partial \leq 0 \rrbracket\ s_4[t] \leq v_{10}[t] \,\vert\vert\, v_8[t] \leq 0$	$\partial < 0 \rrbracket\ v_8[t] > 0 \,\&\&\, v_{10}[t] > s_4[t]$
$v_8[t] == \left(\begin{cases} 1 & v_3 \\ & s \\ 0 & \mathrm{Tr} \end{cases}\right.$	$\partial_{v_3[t]} v_8[t] == 0$	Z	$\partial > 0 \rrbracket$ False	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket$ True	$\partial \neq 0 \rrbracket$ False	$\partial \leq 0 \rrbracket$ True	$\partial < 0 \rrbracket$ False
$v_8[t] == \left(\begin{cases} 1 & v_3 \\ & s \\ 0 & \mathrm{Tr} \end{cases}\right.$	$\partial_{s_3[t]} v_8[t] == 0$	Z	$\partial > 0 \rrbracket$ False	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket$ True	$\partial \neq 0 \rrbracket$ False	$\partial \leq 0 \rrbracket$ True	$\partial < 0 \rrbracket$ False
$v_9[t] == v_{11}[t]$	$\partial_{v_{11}[t]} v_9[t] == 1$	+	$\partial > 0 \rrbracket$ True	$\partial \geq 0 \rrbracket$ True	$\partial == 0 \rrbracket$ False	$\partial \neq 0 \rrbracket$ True	$\partial \leq 0 \rrbracket$ False	$\partial < 0 \rrbracket$ False
$v_{12}[t] == c_9 + rn[-10, 10] \sqrt{\left(\left(c_2 c_3^2 \left(2 - \frac{v_{11}[t]}{c_2}\right)\right) / v_{11}[t]\right)}$	$\partial_{v_{11}[t]} v_{12}[t] == \left(rn[-10, 10] \left(-\frac{c_3^2}{v_{11}[t]} - \frac{c_2 c_3^2 \left(2 - \frac{v_{11}[t]}{c_2}\right)}{v_{11}[t]^2}\right)\right) / \left(2 \sqrt{\frac{c_2 c_3^2 \left(2 - \frac{v_{11}[t]}{c_2}\right)}{v_{11}[t]}}\right)$	±	$\partial > 0 \rrbracket\ rn[-10, 10] < 0$	$\partial \geq 0 \rrbracket\ rn[-10, 10] \leq 0$	$\partial == 0 \rrbracket\ rn[-10, 10] == 0$	$\partial \neq 0 \rrbracket\ rn[-10, 10] \neq 0$	$\partial \leq 0 \rrbracket\ rn[-10, 10] \geq 0$	$\partial < 0 \rrbracket\ rn[-10, 10] > 0$

9.6 Discussion

We have developed an automated method of identifying ambiguous polarities and determining link polarities. The method is analytic: it analyzes the mathematical structure of the model; it does not execute the model. It therefore will identify all conditions of polarity reversal, which is not possible using conventional execution-oriented methods.

A detailed analysis of a Bass Diffusion model was presented. We found that the modelers removed some polarity ambiguities identified by our method by using knowledge of nearby variables. When we added those as assumed constraints, our tool matched the assigned polarities exactly. Our limited investigation suggests future work to develop a definition of polarity that formally utilizes information from nearby connected variables, and perhaps a catalog of any other methods of polarity assignment found in our corpus of 900 SD models.

We found via informal experiments applied to our corpus of SD models that many polarity ambiguities are resolved when there constraints on the range of values that might be assigned to constants, which can be used to determine the range of variables. System Dynamics modeling software does not generally support such assertions. It would be nice if they did.

Ad-hoc methods were required in order to prepare the model for analysis. It is future work to develop a catalog of such methods and implement them in a form suitable to automated analysis.

As a preliminary experiment in scalability, we applied the method to about 900 models and spot checked the assignments (where the source model has polarity labels) and found the polarities determined by our method method compare quite well with the labeled polarities. Future work is to formalize this statement via large-scale systematic testing.

The method could extend to loop polarity constraints using at least two techniques: The chain rule could be applied, though propagating the constraints may prove challenging; or the path could be rewritten into a single equation by repeated variable substitution ((J. Sterman, 2000) §6.2.5 "Auxiliary Variables") then using the methods herein on the resulting equation. The latter method would also reduce vulnerability to “phantom loop” effects

((Mojtahedzadeh, 2011)) and would detect when multiple ambiguous link polarities in a path resolve to an unambiguous polarity.

We computed the domain and Image of all tables, and discovered that adding that information to the assumed constraints reduced the false-positive rate of identification of ambiguous polarities. Future work is to consider this an extension of the SGN() definition of polarity, because there is no judgement required: The table by definition can't violate its image. Other future work would be to use this information to identify latent violations of table domains. A table instance might, for example, have a strictly positive derivative that is simply an artifact of a dataset rather than a structural feature of the phenomenon, implying that the dataset might change and consequently cause polarity ambiguity.

We found that it may not be necessary to expand well-known macros, because the polarity behavior can often be pre-computed. Future work is to explore this. Beyond macros, it may also be useful to identify molecules (Hines, 1996) and associate them with pre-computed properties. Future work is to explore this. The computational crux of our method is the reductions of the partial derivative relational operators to their conditions of satisfaction. We found in a scan of many models that some reductions require specific tuning to avoid combinatorial explosions that consume more memory or cpu time than is practically available. We found that it is often necessary to remove singularities and discontinuities from the domain (this can be done automatically) and specific rewrites may be required as well. Future work is to bring this work to the point where this does not often happen in practice.

10 Equations of the corpus

10.1 Summary

In this study, we extracted and cataloged variable equations from 1,104 System Dynamics (SD) models, ranking the most frequently used functional forms and revealing the equation structures most relied upon by modelers. We found that the 58,882 equations used in the SD models are characterized by 3825 expression patterns, only 40 of which represent 80% of the equations.

This indicates that modelers frequently rely on a small "vocabulary," while occasionally resorting to a far broader set. In contrast to prior focus on higher-level loops and feedback structures, we systematically investigated expression-level building blocks, which offer an atomic lens into the scaffolding beneath complex models.

We employed an entropy-based measure to assess the degree of predictability in SD models, revealing how standard or novel their formulations may be.

These findings provide empirical evidence of which patterns are widely adopted, offering data-driven guidance for more efficient model design. By integrating observed equation frequencies into an automated construction process, one could accelerate convergence on plausible solutions and avoid pursuing rarer, less-proven configurations. Earlier work on Molecules and other SD archetypes introduced canonical patterns but did not systematically analyze large numbers of models to quantify usage. Our large-scale approach closes that gap. We speculate that usage data can refine and streamline model building. We anticipate these insights will aid the design of AI-augmented SD model synthesis and analysis.

10.2 Introduction

Introduction

We have compiled a reference corpus of over 1,000 System Dynamics (SD) models, sourced from academic journals, academic and professional consulting artifacts, and community repositories. Within these models, we identified about 59,000 separate variable expressions, which we determined to be represented by about 3000 mathematical expression patterns. The top 40 of those patterns (less than 2% of the patterns) match 80% of the expressions (Patterns Pareto Chart (clipped)), suggesting that modelers tend to work from a small expression vocabulary but will resort to a much larger vocabulary

Our main objective was to extract and catalog a set of "expression-level building blocks" that frequently appear in practice, guided by the observation that SD research has historically lacked systematic, data-driven investigations of how individual variables are typically defined. Barlas, for example argues that "Since structure-oriented behavior tests combine the strength of structural orientation with the advantage of being quantifiable, they seem to be the most promising direction for research on model validation"((Barlas, 1996)). Our work highlights frequently used expression patterns, suggesting that the greater of the validation effort should probably focus on the less probable (less frequently used and so less community experience with them) expressions. Although prior work often highlights higher-level feedback structures and loops and dynamics behavior, our current focus on atomic variable expressions offers a novel lens for investigating the scaffolding beneath complex models, thereby supporting more reproducible modeling methods.

Finally, we applied an entropy-based measure to evaluate the predictability or variety of expression usage across models in the corpus. This metric provides insight into how standard or innovative a model might be. Our future plans involve using these findings as a starting point for exploring how particular expressions combine into larger feedback loops and how best to validate or communicate these structures: a topic (Barlas, 1996) argues is crucial for the evolution of SD practice. By supplying a quantitative account of expression-level building blocks, we aim to stimulate a culture of open data, shared tools, and systematic approaches in SD research and education, echoing calls for greater rigor articulated in both early foundational work (J. W. Forrester, 1961) and more recent discussions about the field's trajectory (J. W. Forrester, 2007).

Sterman (J. Sterman, 2018) highlights how present-day modeling can benefit from "big data," modern machine learning, MCMC methods, network structures, and advanced computational architectures. Similarly, our research leverages advanced symbolic mathematics, pattern recognition, and corpus-analytic approaches not typically found in traditional SD practice. By studying 59,000 variable expressions across 1,000 models, we are, in effect, bringing large-scale data science into System Dynamics. Doing so not only helps identify recurring "best practices" or surprising outliers in the algebraic formulations, but may also reveal opportunities for better standards and tools, aligning directly with Sterman's argument that SD should always adopt new methodologies and remain open to methods outside its traditional toolkit.

Our work may be useful to the SD community by indicating the mathematical forms that are common in practice and hence might be first considered for use during model development; it may be useful to educators by suggesting "edge case" equations that are probably best avoided by beginning modelers; and it may be useful to researchers by forming a basis for rational probability-based choice of equation structures during automated evolutionary model construction.

10.3 Method

10.3.1 Overview

We determined equation pattern frequencies basically as follows:

1. Import and parse all of the models. SD Model Import (P26) describes the import

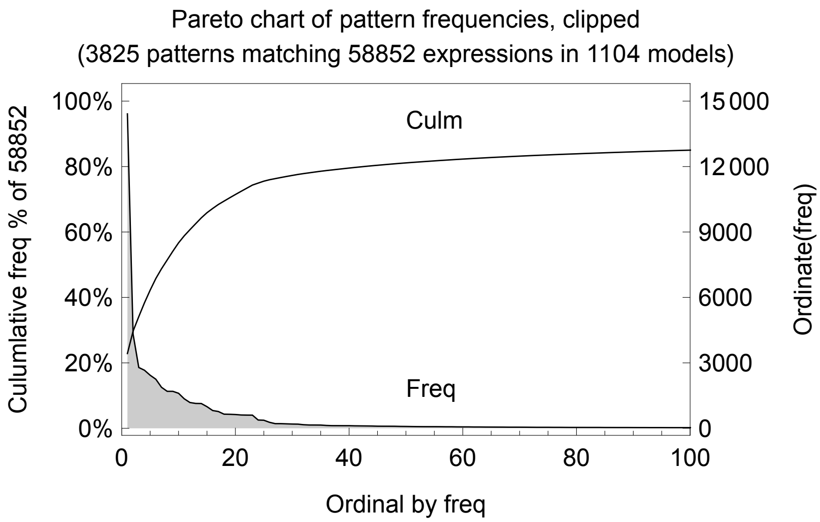


Figure 10.1

process and mathematical representation of the models.

2. Convert each expression to a pattern. The pattern will match the expression and others with the same structure. We generally arrange the patterns so that other expressions that differ by variable names and constant values will match the pattern, though we make some exception. In particular, we generally seek exact matches on certain numbers such as -1, 0, and 1, because we see those are often used where it would not make sense to match other numbers. This process can lead to many patterns that are equivalent. That is dealt with in later steps. Patterns describes patterns and the matching process.

3. Each pattern is assigned an index, e.g. 1, 2, ... Then each expression is tested against each pattern, and assigned the pattern index of the first matching pattern (this is why it is OK to have many patterns that are functionally identical)

4. We now have a list of all expressions in the corpus, and a pattern ID assigned to each expression, where all expressions with the same pattern ID are deemed structurally identical.

10.3.2 SD Model Import

Vensim ".mdl" files were translated into wolfram language syntax using a recursive descent parser algorithm (Aho et al., 1986) with arbitrary look ahead. Two passes were used: The first converted from Vensim syntax and semantics into wolfram language syntax and Vensim semantics; the second converted the first pass to wolfram language semantics. Experimentation was sometimes necessary to determine details of the syntax or semantics.

Only the mathematical essentials were imported: Units, comments and graphical elements were ignored. The imported model can be integrated numerically, manipulated symbolically, and analyzed graphically.

It is future work to implement array variables.

The Vensim variable name character set and structure is much more permissive than those in Wolfram Language, so variables were represented as constant functions with string parameter, e.g. $v[\text{"\$\$} \textit{variable} \quad \textit{name}\text{"}]$.

We give symbols short names, so that the mathematical structure is not obscured. s_i (or si if we are lazy) is used for stocks, v_i for auxiliary variables, l_i for tables (the more obvious "t" is already used to represent simulation time), f_i for stock flows (these are defined as the integrand of the stock. We make these distinct because it is sometimes useful to have a concrete symbol for the flow of a stock), and c_i for constants, where a constant is defined as an auxiliary variable that is assigned a constant numerical value.

Tables in Vensim are a numerical construct specifying a first-order interpolation where the argument is within the domain of the table, and zero-order extrapolation (typically with warning messages issued) for arguments outside the domain of the table: i.e. the first value extends to $-\infty$, and the last value extends to $+\infty$.

For numerical model execution, tables are converted to native numerical interpolation functions in Mathematica. The resulting function has numerical semantics identical to Vensim for interpolation, but will fail with a message if used for extrapolation. If extrapolation semantics are required, the interpolation table is padded with values at $-\infty$ and $+\infty$ that will cause zero-order behavior on extrapolation without generating runtime warning messages. It is useful to identify instances of extrapolation because they indicate possible weaknesses in the model specification. Note that extrapolation does not change the Image of the lookup, but it does extend the domain to $i\infty$ and $+\infty$.

For symbolic analysis, tables are converted to a piecewise representation and all approximate numerical constants in the model are converted to exact rational form. This is because approximate numbers are a numerical concept, rational numbers are an analytic construct that facilitate symbolic manipulations. Note also that we provide both interpolation form, which will reduce to Indeterminate if the domain is violated, and an extrapolation form that is a total function that extends the first value towards $-\infty$ and the last value towards $+\infty$.

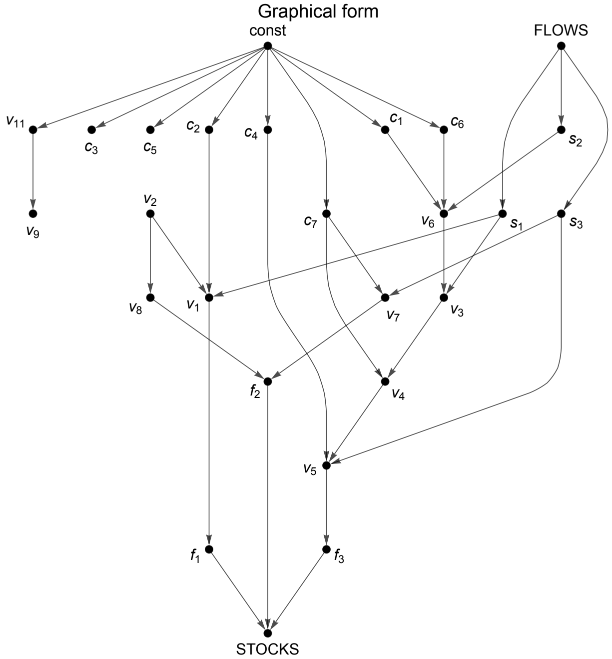
A model may be expressed as a directed graph by constructing a directed edge $v_i \to v_j$ for each independent variable v_j in the equation for variable v_i.

A model may be expressed as a directed acyclic graph by removing all edges $f_i \to s_i$ from the directed graph, which breaks all loops between a flow and its stock.

The imported mathematical and graphical representation of the Inventory Workforce Model is shown in Figure: Inventory Workforce Model

(a)

Graphical form



(b)

Integral form

$s_1[t] == \int_{c_5}^{t} f_1[ti]\, dti + v_2[c_5]$

$s_2[t] == 100 + \int_{c_5}^{t} f_2[ti]\, dti$

$s_3[t] == 50 + \int_{c_5}^{t} f_3[ti]\, dti$

$f_1[t] == v_1[t]$

$f_2[t] == v_7[t] - v_8[t]$

$f_3[t] == v_5[t]$

$v_1[t] == \frac{-s_1[t]+v_2[t]}{c_2}$

$v_2[t] == 100 + \left(\begin{cases} 0 & t < 5 \\ 10 & \text{True} \end{cases} \right)$

$v_3[t] == s_1[t] + v_6[t]$

$v_4[t] == \frac{v_3[t]}{c_7}$

$v_5[t] == \frac{-s_3[t]+v_4[t]}{c_4}$

$v_6[t] == \frac{c_1-s_2[t]}{c_6}$

$v_7[t] == c_7\, s_3[t]$

$v_8[t] == v_2[t]$

$v_9[t] == v_{11}[t]$

$v_{10}[t] == t$

$v_{11}[t] == 0.01$

$c_1 == 100$

$c_2 == 2$

$c_3 == 30$

$c_4 == 4$

$c_5 == 0$

$c_6 == 2$

$c_7 == 2$

$c_1 == v[\text{desired inventory}]$

$c_2 == v[\text{expec time}]$

$c_3 == v[\text{FINAL TIME}]$

$c_4 == v[\text{hring time}]$

$c_5 == v[\text{INITIAL TIME}]$

$c_6 == v[\text{inventory correction time}]$

$c_7 == v[\text{productivity}]$

$f_1 == v[\text{expected demand\_flow}]$

$f_2 == v[\text{Inventory\_flow}]$

$f_3 == v[\text{Work Force\_flow}]$

$s_1 == v[\text{expected demand}]$

$s_2 == v[\text{Inventory}]$

$s_3 == v[\text{Work Force}]$

$v_1 == v[\text{chng in exp dem}]$

$v_2 == v[\text{demand}]$

$v_3 == v[\text{desired production}]$

$v_4 == v[\text{desired work force}]$

$v_5 == v[\text{hring laying off}]$

$v_6 == v[\text{inventoryry corrections}]$

$v_7 == v[\text{production}]$

$v_8 == v[\text{sales}]$

$v_9 == v[\text{SAVEPER}]$

$v_{10} == v[\text{Time}]$

$v_{11} == v[\text{TIME STEP}]$

(c)

Differential form

$s_1'[\tau] == \frac{1}{2} \left(100 + \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) - s_1[\tau]\right)$

$s_2'[\tau] == -100 - \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) + 2\, s_3[\tau]$

$s_3'[\tau] == \frac{1}{4} \left(\frac{1}{2} \left(s_1[\tau] + \frac{1}{2} (100 - s_2[\tau])\right) - s_3[\tau]\right)$

$s_1[0] == 100$

$s_2[0] == 100$

$s_3[0] == 50$

Model: InventoryWorkforce.mdl githash 1449b3b
Stock initialization dependencies elided.
Loops in (a) broken at f_i->s_i.
Long names replaced with short names.
Implicit time variable not shown.

Figure 10.2

10.3.3 Patterns

We use patterns to indicate where a specific element in a particular expression is to be generalized for the purpose of defining "like" expressions. Most of the patterns are fairly straightforward. For example, an integer is matched by "_Integer". A real number is matched by "_Real". A rational is matched by "_Rational". Any type of number is matched by "_?NumberQ", meaning "If the predicate function named NumberQ returns true, match the pattern else don't", where NumberQ does the obvious thing. A set of literal symbols, e.g. a, b, or c, is matched with "a|b|c". Any pattern can be given a name, which forces any other pattern in the expression with the same name to match the input symbol value. For example, "a_+a_" will match "2+2" or "$c_{12} + c_{12}$" but will not match "2+3".

Our choice of pattern details indicates our decisions about what constitutes equivalent expressions. In particular, we choose that if the input expression contains multiple instances of a given variable, then the expression values matching those slots must match exactly. We choose to do this so that an input expression of e.g. "(c1-v1)/c1", which is probably measuring relative change, will not match "(c1-v1)/c2", which probably had a different intent. The pattern in this case would be something like "(c1_-v1_)/c1_". We reuse the variable names as pattern names. The pattern names have no semantics other than they either match other pattern names or they don't, so this is an OK approach.

We also choose to exclude some symbols from generalization. In particular, we treat the integers "0", "1", and "-1" as distinct from any other number forms, on the assumption that expressions like (1-x) are special and do not generalize to "3.2-x". We did not treat common mathematical and physical constants as distinct, though arguably we should have.

We chose to not differentiate between constants ("c"), variables ("v"), and stocks ("s") in expressions.

You will see many instances of e.g. "Indexed[c,12]" and the like in the expressions and patterns. This is simply because we represent constants, variables, stocks, flows, and tables using a concise subscripted form (we also maintain maps between that form and the programmer-assigned variable name), and you are seeing the machinery that creates the subscripted display output. For example, "c_{12}" is "Indexed[c,12]". It follows that a pattern to match any terminal that isn't a literal is "Indexed[c|v|s|f|l,name_Integer]".

An example is presented in Inventory Workforce model expressions and patterns. The full syntax and semantics of patterns and their algorithms, along with tutorial materials, can be seen at https://reference.wolfram.com/language/guide/Patterns.html

Table 10.1: Equations and Patterns of Inventory Workforce Model (Mojtahedzadeh, n.d., supplementry material). "Expr" is expression in imported model. "Pat of Expr" is the pattern derived from the "Expr" that matches related expressions. "Pat ID" is the index of the computed pattern. "First matching ID" is the pattern with least ID that matches the expression: This is necessary because several expressions will often generate equivalent patterns. For example, row 22, a single variable, also matches the pattern of row 15.

row	Expr	Pat of Expr	Pat ID	First ID matching expr
1	0	0	1	1
2	t	t	2	2
3	0.01	const0001_?NumberQ	3	3
4	2	const0001_?NumberQ	3	3
5	2	const0001_?NumberQ	3	3
6	2	const0001_?NumberQ	3	3
7	4	const0001_?NumberQ	3	3
8	30	const0001_?NumberQ	3	3
9	100	const0001_?NumberQ	3	3
10	$100 + \left(\begin{cases} 0 & t < 5 \\ 10 & \text{True} \end{cases}\right)$	$\text{const0003\_?NumberQ} + \left(\begin{cases} 0 & t < \text{const0001\_?NumberQ} \\ \text{const0002\_?NumberQ} & \text{True} \end{cases}\right)$	4	4
11	$100 + \int_{c_5}^{t} f_2[ti]\, dti$	$\text{const0001\_?NumberQ} + \int_{(\text{kc5}:s \mid v \mid c \mid f \mid l)_{\text{c5\_Integer}}}^{t} (\text{kf2}: s \mid v \mid c \mid f \mid l)_{\text{f2\_Integer}}[ti]\, dti$	5	5
12	$50 + \int_{c_5}^{t} f_3[ti]\, dti$	$\text{const0001\_?NumberQ} + \int_{(\text{kc5}:s \mid v \mid c \mid f \mid l)_{\text{c5\_Integer}}}^{t} (\text{kf3}: s \mid v \mid c \mid f \mid l)_{\text{f3\_Integer}}[ti]\, dti$	6	5
13	$\frac{c_1 - s_2[t]}{c_6}$	$\frac{(\text{kc1}:s \mid v \mid c \mid f \mid l)_{\text{c1\_Integer}} - (\text{ks2}:s \mid v \mid c \mid f \mid l)_{\text{s2\_Integer}}[t]}{(\text{kc6}:s \mid v \mid c \mid f \mid l)_{\text{c6\_Integer}}}$	7	7
14	$c_7\, s_3[t]$	$(\text{kc7}: s \mid v \mid c \mid f \mid l)_{\text{c7\_Integer}}\ (\text{ks3}: s \mid v \mid c \mid f \mid l)_{\text{s3\_Integer}}[t]$	8	8
15	$v_1[t]$	$(\text{kv1}: s \mid v \mid c \mid f \mid l)_{\text{v1\_Integer}}[t]$	9	9
16	$v_{11}[t]$	$(\text{kv11}: s \mid v \mid c \mid f \mid l)_{\text{v11\_Integer}}[t]$	10	9
17	$v_2[t]$	$(\text{kv2}: s \mid v \mid c \mid f \mid l)_{\text{v2\_Integer}}[t]$	11	9
18	$\frac{-s_1[t] + v_2[t]}{c_2}$	$\frac{-(\text{ks1}:s \mid v \mid c \mid f \mid l)_{\text{s1\_Integer}}[t] + (\text{kv2}:s \mid v \mid c \mid f \mid l)_{\text{v2\_Integer}}[t]}{(\text{kc2}:s \mid v \mid c \mid f \mid l)_{\text{c2\_Integer}}}$	12	12
19	$\int_{c_5}^{t} f_1[ti]\, dti + v_2[c_5]$	$\int_{(\text{kc5}:s \mid v \mid c \mid f \mid l)_{\text{c5\_Integer}}}^{t} (\text{kf1}: s \mid v \mid c \mid f \mid l)_{\text{f1\_Integer}}[ti]\, dti + (\text{kv2}: s \mid v \mid c \mid f \mid l)_{\text{v2\_Integer}}\left[(\text{kc5}: s \mid v \mid c \mid f \mid l)_{\text{c5\_Integer}}\right]$	13	13
20	$\frac{v_3[t]}{c_7}$	$\frac{(\text{kv3}:s \mid v \mid c \mid f \mid l)_{\text{v3\_Integer}}[t]}{(\text{kc7}:s \mid v \mid c \mid f \mid l)_{\text{c7\_Integer}}}$	14	14
21	$\frac{-s_3[t] + v_4[t]}{c_4}$	$\frac{-(\text{ks3}:s \mid v \mid c \mid f \mid l)_{\text{s3\_Integer}}[t] + (\text{kv4}:s \mid v \mid c \mid f \mid l)_{\text{v4\_Integer}}[t]}{(\text{kc4}:s \mid v \mid c \mid f \mid l)_{\text{c4\_Integer}}}$	15	12
22	$v_5[t]$	$(\text{kv5}: s \mid v \mid c \mid f \mid l)_{\text{v5\_Integer}}[t]$	16	9
23	$s_1[t] + v_6[t]$	$(\text{ks1}: s \mid v \mid c \mid f \mid l)_{\text{s1\_Integer}}[t] + (\text{kv6}: s \mid v \mid c \mid f \mid l)_{\text{v6\_Integer}}[t]$	17	17
24	$v_7[t] - v_8[t]$	$(\text{kv7}: s \mid v \mid c \mid f \mid l)_{\text{v7\_Integer}}[t] - (\text{kv8}: s \mid v \mid c \mid f \mid l)_{\text{v8\_Integer}}[t]$	18	18

10.4 Results

10.4.1 Occurance Rate of Equations

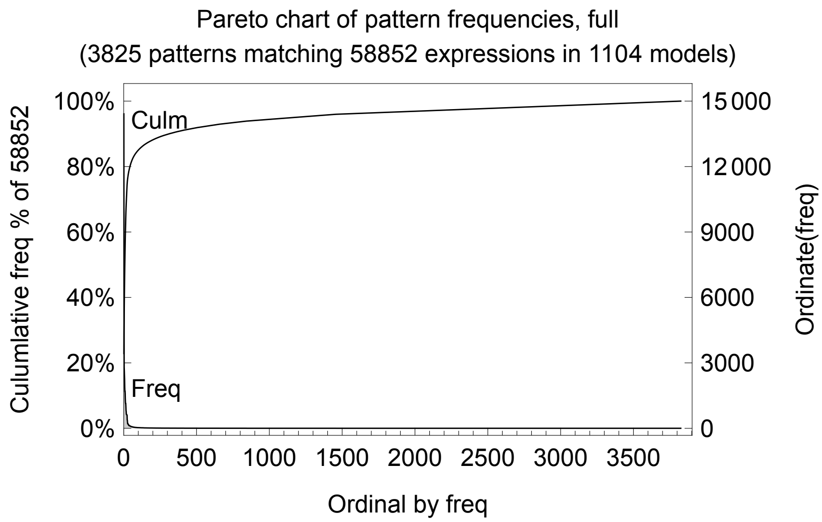
In this section we present the most and least frequently occurring of the $\tilde{3}500$ expression patterns we found within the $\tilde{5}8$K expressions in our corpus of $\tilde{1}100$ SD models.

The results are presented in Table of most and least frequently occurring patterns. We present only the two ends of the data table, as the full table consists of about 3500 rows.

We also present the results in the form of Pareto charts (Juran, 1951). Pareto chart combines a bar chart, which represents the frequency or magnitude of different categories (in our case frequency of use of particular patterns), with a cumulative line graph, which illustrates the cumulative contribution of each category(aka pattern) in descending order. The full chart is in Patterns Pareto Chart (full) (Figure 10.3). It is rather difficult to discern the details of that chart, so we also provide a clipped version Patterns Pareto Chart (clipped) (Figure 10.1) that gives more visibility into the behavior of the high prevalence patterns.

Table 10.2: Table of most and least frequently occurring patterns. "Pat Idx" uniquely identifies the pattern. "Pat Freq" is the number of times the pattern was matched in our corpus of about 58K expressions from about 1K models. "Pattern" is the pattern. "Typical matching expr" is a random matching expression from the corpus. The table shows only the first few most frequent patterns, and the last few infrequent patterns. Note the patterns are not always exclusive. Idx 2 and 3 also would match idx 1556, for example. See body of text for the purpose of this overlap.

Pat idx	Pat freq	Pattern	Example of matching expression
1556	n[13 453]	const0001_ ? NumberQ → 1556	100
5808	n[4028]	$(kl1 : s \mid v \mid c \mid f \mid l)_{l1\_Integer}[t] \to 5808$	$v_{13}[t]$
3	n[2603]	$1 \to 3$	1
5809	n[2485]	$(kc1 : s \mid v \mid c \mid f \mid l)_{c1\_Integer}\ (kl1 : s \mid v \mid c \mid f \mid l)_{l1\_Integer}[t] \to 5809$	$c_1\, v_7[t]$
6917	n[2271]	$-(kl1 : s \mid v \mid c \mid f \mid l)_{l1\_Integer}[t] + (ks1 : s \mid v \mid c \mid f \mid l)_{s1\_Integer}[t] \to 6917$	$v_{12}[t] - v_{15}[t]$
2	n[2101]	$0 \to 2$	0
3676	n[1755]	$\int_{(kc10:s \mid v \mid c \mid f \mid l)_{c10\_Integer}}^{t} (kf1 : s \mid v \mid c \mid f \mid l)_{f1\_Integer}[ti]\, dti \to 3676$	$\int_{c_5}^{t} f_3[ti]\, dti$
5810	n[1583]	$\frac{(kl1:s \mid v \mid c \mid f \mid l)_{l1\_Integer}[t]}{(kc12:s \mid v \mid c \mid f \mid l)_{c12\_Integer}} \to 5810$	$\frac{v_{20}[t]}{c_4}$
3684	n[1582]	$const0001\_ ? NumberQ + \int_{(kc10:s \mid v \mid c \mid f \mid l)_{c10\_Integer}}^{t} (kf1 : s \mid v \mid c \mid f \mid l)_{f1\_Integer}[ti]\, dti \to 3684$	$66\,666.7 + \int_{c_{16}}^{t} f_4[ti]\, dti$
5962	n[1499]	$(kl4 : s \mid v \mid c \mid f \mid l)_{l4\_Integer}[t]\ (kl10 : s \mid v \mid c \mid f \mid l)_{l10\_Integer}[t] \to 5962$	$v_6[t]\, v_7[t]$
3325	n[1263]	_InterpolatingFunction[arg] → 3325	InterpolatingFunction[≪1≫][arg]
4	n[1104]	$t \to 4$	t
3678	n[1065]	$(kc11 : s \mid v \mid c \mid f \mid l)_{c11\_Integer} + \int_{(kc10:s \mid v \mid c \mid f \mid l)_{c10\_Integer}}^{t} (kf1 : s \mid v \mid c \mid f \mid l)_{f1\_Integer}[ti]\, dti \to 3678$	$c_4 + \int_{c_6}^{t} f_3[ti]\, dti$
6256	n[1061]	$\frac{(kl21:s \mid v \mid c \mid f \mid l)_{l21\_Integer}[t]}{(kl20:s \mid v \mid c \mid f \mid l)_{l20\_Integer}[t]} \to 6256$	$\frac{s_1[t]}{s_2[t]}$
≪snip≫	≪snip≫	≪snip≫	≪snip≫
751	n[1]	$Max\Big[0, Min\Big[\big((kc2 : s \mid v \mid c \mid f \mid l)_{c2\_Integer} - (ks2 : s \mid v \mid c \mid f \mid l)_{s2\_Integer}[t] + (ks3 : s \mid v \mid c \mid f \mid l)_{s3\_Integer}[t]\big) / (kc9 : s \mid v \mid c \mid f \mid l)_{c9\_Integer},\ (kv12 : s \mid v \mid c \mid f \mid l)_{v12\_Integer}[t]\Big]\Big] \to 751$	$Max\Big[0, Min\Big[\frac{c_2 - s_2[t] + s_3[t]}{c_9}, v_{12}[t]\Big]\Big]$
749	n[1]	$Max\Big[0, Min\Big[\frac{(ks2:s \mid v \mid c \mid f \mid l)_{s2\_Integer}[t]}{(kc9:s \mid v \mid c \mid f \mid l)_{c9\_Integer}}, \frac{(ks3:s \mid v \mid c \mid f \mid l)_{s3\_Integer}[t]}{(kc9:s \mid v \mid c \mid f \mid l)_{c9\_Integer}}\Big]\Big] \to 749$	$Max\Big[0, Min\Big[\frac{s_2[t]}{c_9}, \frac{s_3[t]}{c_9}\Big]\Big]$
748	n[1]	$Max\Big[0, Min\Big[\frac{(kc31:s \mid v \mid c \mid f \mid l)_{c31\_Integer}\ (ks14:s \mid v \mid c \mid f \mid l)_{s14\_Integer}[t]}{(kc10:s \mid v \mid c \mid f \mid l)_{c10\_Integer}},\ (kv36 : s \mid v \mid c \mid f \mid l)_{v36\_Integer}[t]\Big]\Big] \to 748$	$Max\Big[0, Min\Big[\frac{c_{31}\, s_{14}[t]}{c_{10}}, v_{36}[t]\Big]\Big]$
549	n[1]	$(kc2 : s \mid v \mid c \mid f \mid l)_{c2\_Integer}\ (kc29 : s \mid v \mid c \mid f \mid l)_{c29\_Integer}\ (kc44 : s \mid v \mid c \mid f \mid l)_{c44\_Integer} + \frac{(kc2:s \mid v \mid c \mid f \mid l)_{c2\_Integer}\ (kc44:s \mid v \mid c \mid f \mid l)_{c44\_Integer}\ (kc59:s \mid v \mid c \mid f \mid l)_{c59\_Integer}}{(kc43:s \mid v \mid c \mid f \mid l)_{c43\_Integer}} \to 549$	$c_2\, c_{29}\, c_{44} + \frac{c_2\, c_{44}\, c_{59}}{c_{43}}$
484	n[1]	$\big((kc27 : s \mid v \mid c \mid f \mid l)_{c27\_Integer}\ (kc44 : s \mid v \mid c \mid f \mid l)_{c44\_Integer} + \big(1 - (kc44 : s \mid v \mid c \mid f \mid l)_{c44\_Integer}\big)\ (kc47 : s \mid v \mid c \mid f \mid l)_{c47\_Integer}\big) / (kc4 : s \mid v \mid c \mid f \mid l)_{c4\_Integer} \to 484$	$\frac{c_{27}\, c_{44} + (1 - c_{44})\, c_{47}}{c_4}$



Pareto chart of pattern frequencies, full
(3825 patterns matching 58852 expressions in 1104 models)
Culm
Freq
100%
80%
60%
40%
20%
0%
Culumlative freq % of 58852
15 000
12 000
9000
6000
3000
0
Ordinate(freq)
0
500
1000
1500
2000
2500
3000
3500
Ordinal by freq

Figure 10.3

10.4.2 Entropy of models by Equations

Entropy in the context of document analysis is a measure of the average information content or unpredictability of symbols (or, in our case, expression patterns) within a text (in our case, a SD model), calculated using symbol probabilities derived from a reference corpus (in our case, our model corpus). In information theory terms, each symbol (e.g., character, word, or token, or pattern) is considered a random variable with a certain probability of occurring. These probabilities are first estimated from a sufficiently large and representative reference corpus, which captures typical language patterns. The entropy H of a given document with respect to those probabilities can be computed using Shannon's formula:

$$H = -\sum_{i} p_i \log_2 p_i,$$

where p_i is the probability of the i-th symbol. Higher entropy values indicate a greater degree of unpredictability or surprise in the distribution of symbols, meaning the document's content diverges more from the patterns observed in the reference corpus.

By grounding the calculation in a reference corpus, we quantify how "expected" or "conventional" a new text is. If the new document's symbol frequencies closely match those observed in the corpus, its entropy will be relatively lower, suggesting it largely follows the established patterns of symbol usage. Conversely, if the document's symbols appear in frequencies that significantly deviate from those in the corpus, the entropy will be higher, indicating increased novelty or less predictability. This measure can be used in various linguistic, computational, and data analytics applications, such as text classification, detecting anomalies, or assessing linguistic complexity.

From a practical standpoint, when the entropy is computed using the base-2 logarithm, it directly corresponds to the minimum average number of bits required to encode each symbol in the text under an idealized coding scheme. In other words, if one possesses the exact symbol probability distribution, which can be derived from the reference corpus, for both the compression (encoding) and the decompression (decoding) processes, then the document's entropy provides the theoretical lower bound on its compressibility. A higher entropy value indicates that more bits are needed for each symbol, implying that the document is relatively less compressible. Conversely, a lower entropy suggests that fewer bits per symbol may be sufficient, reflecting greater redundancy or predictability in the document's symbol distribution.

In Information Theory (what we are doing here), Entropy is commonly measured in *bits*, *nats*, or *hartleys* (also referred to as *bans*), depending on the base of the logarithm employed in Shannon's formula, $H = -\sum p_i \log p_i$. When the base-2 logarithm ($\log_2$) is used, the units are bits, reflecting the binary nature of most digital systems. If one substitutes the natural logarithm (ln), the resulting units are nats, and if the common logarithm ($\log_{10}$) is chosen, the entropy is expressed in hartleys. In thermodynamics.

Conversion between these units follows straightforward logarithmic identities. For instance, to convert from bits to nats, one uses the fact that $\ln(x) = \log_2(x)\ln(2)$. Consequently,

$$1\,\text{bit} \approx 0.693\,\text{nats}, \quad \text{and} \quad 1\,\text{nat} \approx 1.4427\,\text{bits}.$$

Similarly, conversion to or from hartleys relies on $\log_{10}(x) = \frac{\ln(x)}{\ln(10)}$. These relationships allow for changing the base of the entropy calculation without loss, thereby accommodating different conventions or analytical requirements in various scientific and engineering domains.

Table 10.3: Equation Entropy of selected models. Entropy is computed with respect to our model corpus. Only models that are generally available and have especially high or low entropy are presented.

Equation Entropy of selected models in corpus

Rank	File	Entropy(Nats)
1	sdproceedings/2006/proceed/supp/405.mdl	138.662
2	sdproceedings_unzipped/sdproceedings/2013/proceed/supp/S1191/CSU Chico Master Plan 03182013.mdl	57.2002
3	sdproceedings/2015/proceed/supp/S1263.mdl	56.4014
4	sdproceedings/2007/proceed/supp/372.mdl	52.8651
5	sdproceedings/2022/supp/S1308.mdl	45.781
6	sdproceedings/2017/proceed/supp/S1356.mdl	44.0776
7	sdproceedings/2019/proceed/supp/S2127.mdl	42.6541
8	sdproceedings/2004/SDS_2004/SUP_MAT/352.mdl	41.8767
9	sdproceedings_unzipped/sdproceedings/2015/proceed/supp/S1391/SDSC_Institutional_Corruption/US_Campaign_Finance_and_Institutional_Corruption_in Public_Policy.mdl	41.8509
10	sdproceedings/2010/proceed/supp/S1301.mdl	41.4464
11	sdproceedings/2015/proceed/supp/S1285.mdl	40.9494
12	sdproceedings/2015/proceed/supp/S1306.mdl	38.2852
13	sdproceedings_unzipped/sdproceedings/2016/proceed/supp/S1321/Paper Support/ICSDS Coordination Model.mdl	37.8297
14	sdproceedings/2011/proceed/supp/S1267.mdl	35.9261
15	sdproceedings/2021/supp/S1179.mdl	35.5224
≪snip≫	≪snip≫	≪snip≫
547	sdproceedings_unzipped/sdproceedings/2004/SDS_2004/SUP_MAT/403/Key CLDs/Figure03.mdl	1.74943
548	sdproceedings_unzipped/sdproceedings/2010/proceed/supp/S1241/Causal Loop 1.mdl	1.74728
549	sdproceedings_unzipped/sdproceedings/2018/proceed/supp/S2225/Supporting service.mdl	1.73991
550	sdproceedings_unzipped/sdproceedings/2007/proceed/supp/209/Dynamic_Experiments_2007/argument_CLD.mdl	1.73891
551	sdproceedings_unzipped/sdproceedings/2014/proceed/supp/S1447/TOC_ConflictArchetype.mdl	1.72901
552	sdproceedings_unzipped/sdproceedings/2011/proceed/supp/S1411/Cusal Diagrams/Supply.mdl	1.72511
553	sdproceedings_unzipped/sdproceedings/2018/proceed/supp/S2350/Supporting Materials/Vensim Models/SimplifiedRoomTemperatureModel.mdl	1.66991
554	sdproceedings_unzipped/sdproceedings/2004/SDS_2004/SUP_MAT/403/Key CLDs/Figure01.mdl	1.6594
555	sdproceedings_unzipped/sdproceedings/2022/supp/S1335/Appendix 3 CLD biodiversity & environment - VF.mdl	1.65531
556	sdproceedings_unzipped/sdproceedings/2018/proceed/supp/S2162/1 sucess to successful archetype-Paper2162-Policies for University Management.mdl	1.65531
557	sdproceedings_unzipped/sdproceedings/2018/proceed/supp/S2115/pulse.mdl	1.63543
558	sdproceedings_unzipped/sdproceedings/2018/proceed/supp/S2350/Supporting Materials/Vensim Models/SimplifiedInventoryModel.mdl	1.61395
559	sdproceedings_unzipped/sdproceedings/2004/SDS_2004/SUP_MAT/403/Key CLDs/Figure02.mdl	1.59407
560	sdproceedings_unzipped/sdproceedings/2017/proceed/supp/S1271/Limits to Growth.mdl	1.5513
561	sdproceedings_unzipped/sdproceedings/2005/proceed/supp/226/model.mdl	1.34695

10.5 Discussion

Our work is based on the premise that the structure of individual equations is interesting. But from a mathematical perspective, this seems a rather dubious claim. After all, there are an infinite number of equations that express any given relationship. In the case of a system dynamics model, the math can always be rewritten so the model has only stocks and a single equation representing the integrand of the stock. In the case of system dynamics, the variables are interesting. We represent the structure of the model not as a mathematical curve-fitting exercise, but to express the structure of the system of interest into the variables, so we can gain insights into the causes of the behavior and the impact of changes to the structure on the behavior. The basis of our premise, then, is that the modeler intended to manifest the structure of the SOI into the variables and so they are immutable. Our premise is thus supportable: The mathematical patterns directly capture the intent of the modeler, and are thus quite interesting.

That said, we did not treat mathematical and physical constants as special, so e.g. $\pi v12^2$ and $c42v12^2$ will match the same pattern, but probably represent different modeler intent. So we have violated our own premise and basically rewritten the equation.

We found numerous instances of "Min(0,...)", "Min(100,Max(0, xxx)" and the like. This pattern is often used to clip the results of a table lookup, or to clip the argument to a table lookup. These are clearly computational artifacts, not representations of the structure of the SOI. It may be appropriate to reduce these to the structural intent they represent, perhaps by assuming the table has the proper domain and image.

Which brings us to tables. We examined many of the tables in our corpus, and found that a good fraction of them seem to be encodings of "swish" sorts of behavior, such as the logistic sigmoid. What is the intent of the table? Should it be represented by a behavior, e.g. (Glick & Duhon, 1994)? A logistic sigmoid function? Data points? It may be reasonable to classify the behaviors represented by the tables, and then incorporate the probabilities into the entropy calculations. There are pros and cons of using real-world data vs mathematical simplification ("curve fit") of that data. We believe there is a place for each.

What if we replaced each equation in a model with its pattern? The result would not be executable, but would form a "skeleton" or "meta-model" or "pattern abstraction" of the model. Would models that are equivalent or similar in this pattern space have interesting shared properties? Does this suggest another way of categorizing models in a corpus? Another way of identifying "molecules" or evolving models?

Are low-probability variables suggestive of areas requiring refinement or additional verification?

11 Generalized Loop Sets

11.1 Summary

The "Generalized Independent Loop Sets" (GILS) algorithm improves upon KILS, SILS, and CILS, while maintaining their independence property and applicability in loop dominance analysis. GILS employs a stricter independence definition, requiring modification of only a single link gain to alter a single loop gain, potentiating investigation of gain-related model behavior. Furthermore and uniquely, GILS enumerates loop sets, offering a more comprehensive analysis through multiple perspectives.

GILS is readily computed as an optimization problem, using Integer Linear Programming (ILP) and existing optimization libraries. This automated approach facilitates integration of GILS into existing Systems Dynamics tools. Custom and/or approximate algorithms could improve performance.

We computed GILS on $\sim$ 678 models in our test corpus, with $\sim$ 13 fails, all due to timeout (no incorrect results), providing evidence that the algorithm is fairly capable. A fully executable example is provided in Supplementary Materials.

11.2 Introduction

Kampmann Independent Loop Set (KILS) (Kampmann, 2012), Shortest Independent Loop Set (SILS) (Huang et al., 2012; Oliva, 2004), and Complete Independent Loop Sets (CILS) (Middleton & Motamed, 2024) enhance model analysis by isolating loop subsets for independent analysis. Each selects special subsets, all maximally independent in some sense, of the loops for isolated consideration. The use of the subsets is well described in the aforementioned literature, and compared in (Hayward & Roach, 2024).

Existing algorithms have the property that the loop independence is weak: it will usually be necessary to adjust many link gains to change exactly one loop gain. It would therefore be cumbersome to manipulate link gains in order to understand model behavior.

Moreover, none of the cited algorithms can enumerate independent subsets, of which there are probably many. The user must accept the single set that is presented, rather than select the best from amongst the several possible independent subsets present in most models.

We present "Generalized Independent Loop Sets" (GILS): (1) An exact algorithm for generating loop sets that are independent by a stricter definition that entails the weaker definition: Each member of the set has one link that is in no other member of the set. The gain of each loop in the set can be independently adjusted via the gain of that single link, without altering the gains of any other loops in the loop set; and (2) independent loop sets can be enumerated, allowing the modeler to find and exploit loop sets most applicable to the problem at hand. An example of this form of independence is presented in (Figure 11.1).

GILS is implemented as an optimization problem, specifically to maximize the number of loops in the ILS while constrained by the independence requirement. Libraries for solving

Method	Linearly Independent?	Exact Maximum Length	Links to change a single loop gain	Enumerates all independent loop sets	Polynomial execution time	Execution Time	Algorithm Type
KILS	Yes(1)	Usually	>=1	No	Yes	Fastest	Graph
SILS	Yes(1)	Usually	>=1	No	Yes	Fastest	Graph
CILS	Yes(1)	Usually	>=1	No	Yes	Fastest	Graph
GILS	Yes(2)	Yes	==1	Yes	No	Reasonable(3)	ILP

Table 11.1: GILS is exact. ILS, SILS, and CILS are approximate. GILS provides stricter independence and enumerates exactly and all independent sets in order of decreasing length.
(1) KILS, SILS, CILS, GILS all return loop sets with the same linearity property: Any combination of loop gains can be achieved by changing some combination of link gains in the independent set. Changing exactly one gain will usually require changing several link gains across several loops.
(2) GILS generates sets with the unique property that each loop has a link that changes the gain of only that loop.
(3) If the GILS methods are applied to a subset of the model loops by selecting the shorter loops in the set, as is done in CILS and SILS, it will run substantially faster but may return approximate results just as CILS and SILS do. Probably not the same approximate result.

p\l	l_{1}	l_{9}	l_{17}	l_{25}	l_{3}	l_{11}	l_{19}	l_{27}	l_{6}	l_{29}	l_{15}	l_{24}	l_{7}	l_{30}	l_{21}	l_{16}	l_{2}	l_{10}	l_{18}	l_{26}	l_{4}	l_{12}	l_{20}	l_{28}	l_{13}	l_{8}	l_{22}	l_{31}	l_{5}	l_{14}	l_{23}
p_{1}	1	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	·	·	·
p_{2}	·	1	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	·	·
p_{3}	·	·	1	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	·
p_{4}	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·
p_{5}	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	1	·	·	·	·	·	·	·	·	·	·
p_{6}	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	1	·	·	·	·	·	·	·	·	·
p_{7}	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	1	·	·	·	·	·	·	·	·
p_{8}	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	1	·	·	·	·	·	·	·
p_{9}	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	·	1	1	·	·	1	1	·	·	1	·	·	·	·	·	·
p_{11}	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	1	·	·	1	1	·	·	1	·	1	·	·	·	·	·
p_{12}	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	1	1	·	·	1	1	·	·	·	1	·	·	·	·
p_{14}	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	·	·	·	1	1	·	·	1	1	·	·	·	1	·	·	·
p_{17}	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	·	1	1	1	·	1	1	1	·	1	·	1	·	·	·	·
p_{19}	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	·	1	1	·	1	1	1	·	1	1	1	·	·	·	·	·
p_{20}	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	1	·	1	1	1	·	1	1	·	1	·	1	·	·	·
p_{22}	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	1	·	1	1	1	·	1	1	1	·	·	1	1	·	·	·

Figure 11.1: GILS applied to Multi-agent Diffusion (Huang et al., 2012) n==4. Rows and columns permuted to accentuate loop independence. p_i and l_i are the path ID and link ID per nomenclature of (Middleton & Motamed, 2024). Diagonal 1's (accented) are unique to each path in the set and indicate the link that sets the gain of exactly that path. The empty columns (framed) are links in the model that are not in the ILS because the ILP was configured to maximize set size where each loop has a unique link, which could not cover all links in the model. Full working example in Supplementary Material. It is future work to explore the utility of constraint variations that balance link coverage, set size, and measure of loop independence.

such optimization problems are available, so the custom code from problem to solution is relatively short and so relatively inexpensive and reliable. The problem setup is independent of the solution algorithm, so custom performant algorithms may be easily integrated as desired.

GILS independent loop sets are enumerated by repeatedly executing the GILS algorithm, each with all prior solutions excluded. The result is an enumeration of ILS's in decreasing length.

The properties of KILS, SILS, CILS, and GILS are summarized in Summary Table (Table 11.1).

11.3 Method

11.3.1 Definitions

A loop, or cycle, is defined as a path that does not repeat a vertex or edge other than the starting and ending vertices, and the starting and ending vertices are the same.

"independent loop set" or ILS is defined to mean that the the gain of a single loop in the set can be changed without changing the gain of any other loop in the set. The crux is in how many link gains must be changed in order to change exactly one loop gain.

11.3.2 SD Model Import

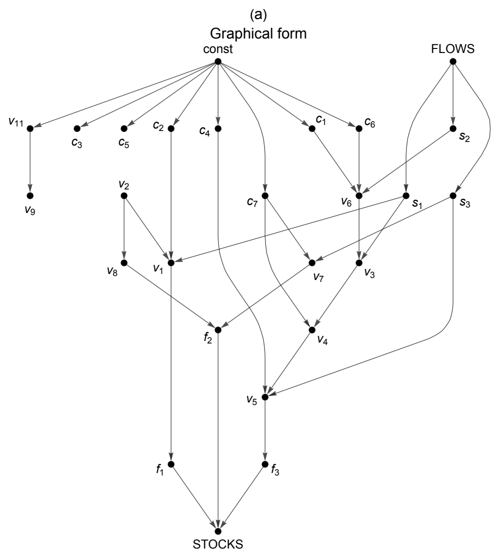
Vensim ".mdl" files were translated into wolfram language using a recursive descent parser algorithm aho1986 with arbitrary look ahead also written in Wolfram Language.

Only the mathematical essentials were imported: Units, comments and graphical elements were ignored. The imported model can be integrated numerically, manipulated symbolically, and analyzed graphically.

We give symbols short names, so that the mathematical structure is not obscured. si for stocks, v_i for auxiliary variables, l_i for tables, f_i for stock flows.

A model is expressed as a directed graph by constructing a directed edge $v_i \to v_j$ for each independent variable v_j in the equation for variable v_i, or as a directed acyclic graph by removing all edges $f_i \to s_i$ from the directed graph to break all loops between a flow and its stock.

The imported mathematical and graphical representation of the Inventory Workforce Model is shown in fig Inventory Workforce Model (Figure 11.2). The algorithms in this paper are applied to graphs of that form.



(b)

Integral form

$s_1[t] == \int_{c_5}^{t} f_1[ti]\, dti + v_2[c_5]$

$s_2[t] == 100 + \int_{c_5}^{t} f_2[ti]\, dti$

$s_3[t] == 50 + \int_{c_5}^{t} f_3[ti]\, dti$

$f_1[t] == v_1[t]$

$f_2[t] == v_7[t] - v_8[t]$

$f_3[t] == v_5[t]$

$v_1[t] == \frac{-s_1[t]+v_2[t]}{c_2}$

$v_2[t] == 100 + \left(\begin{cases} 0 & t < 5 \\ 10 & \text{True} \end{cases} \right)$

$v_3[t] == s_1[t] + v_6[t]$

$v_4[t] == \frac{v_3[t]}{c_7}$

$v_5[t] == \frac{-s_3[t]+v_4[t]}{c_4}$

$v_6[t] == \frac{c_1-s_2[t]}{c_6}$

$v_7[t] == c_7\, s_3[t]$

$v_8[t] == v_2[t]$

$v_9[t] == v_{11}[t]$

$v_{10}[t] == t$

$v_{11}[t] == 0.01$

$c_1 == 100$

$c_2 == 2$

$c_3 == 30$

$c_4 == 4$

$c_5 == 0$

$c_6 == 2$

$c_7 == 2$

$c_1 ==$ v[desired inventory]

$c_2 ==$ v[expec time]

$c_3 ==$ v[FINAL TIME]

$c_4 ==$ v[hring time]

$c_5 ==$ v[INITIAL TIME]

$c_6 ==$ v[inventory correction time]

$c_7 ==$ v[productivity]

$f_1 ==$ v[expected demand_flow]

$f_2 ==$ v[Inventory_flow]

$f_3 ==$ v[Work Force_flow]

$s_1 ==$ v[expected demand]

$s_2 ==$ v[Inventory]

$s_3 ==$ v[Work Force]

$v_1 ==$ v[chng in exp dem]

$v_2 ==$ v[demand]

$v_3 ==$ v[desired production]

$v_4 ==$ v[desired work force]

$v_5 ==$ v[hring laying off]

$v_6 ==$ v[inventoryry corrections]

$v_7 ==$ v[production]

$v_8 ==$ v[sales]

$v_9 ==$ v[SAVEPER]

$v_{10} ==$ v[Time]

$v_{11} ==$ v[TIME STEP]

(c)

Differential form

$s_1'[\tau] == \frac{1}{2} \left(100 + \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) - s_1[\tau]\right)$

$s_2'[\tau] == -100 - \left(\begin{cases} 0 & \tau < 5 \\ 10 & \text{True} \end{cases} \right) + 2\, s_3[\tau]$

$s_3'[\tau] == \frac{1}{4} \left(\frac{1}{2} \left(s_1[\tau] + \frac{1}{2} (100 - s_2[\tau])\right) - s_3[\tau]\right)$

$s_1[0] == 100$

$s_2[0] == 100$

$s_3[0] == 50$

```
Model: InventoryWorkforce.mdl githash 1449b3b
Stock initialization dependencies elided.
Loops in (a) broken at flow->stock.
Long names replaced with short names.
Implicit time variable not shown.
```

Figure 11.2: Internal mathematical and graphical representation of Inventory Workforce Model (Mojtahedzadeh, n.d., supplementry material). GILS is applied to the graphical form (a).

11.3.3 GILS

The setup of GILS as an optimization problem to be solved using ILP solvers such as https://www.mosek.com follows.

Solving an ILP is computationally hard: In the worst case basically every combination of candidate loops must be tested, leading to an upper bound on execution time exponential in candidate loop count. This means that our approach could in some cases fail due to time or memory constraint violations, but practical experience in many industries suggests that the failure frequency drops rapidly under a characteristic problem size. We validate this assertion by computing some GLS subsets on a corpus of quite a few models, as shown in Results.

We transform the model graph by adding a unique link identification number to each link in the graph, l_i, then apply the following algorithm.

We represent the set of paths for which the largest ILP as $p_1, p_2, ...$, each p_i 1 if the path is in the ILP, 0 if it is not.

The optimization problem is to maximize $\sum_i p_i$.

The links in the set of paths are represented as variables $l_i, l_2, ...$. A link variable is the number of paths using that link for which p_i is 1. In other words, it is the number of paths in the candidate ILS that use the indicated link.

The variables $z_1, z_2, ...$ are 1 if the corresponding $l_i = 1$, else zero.

The main setup is shown in ILS ILP Setup (Figure 11.3). The constraints for p_i ILP pi implies (Figure 11.5) and l_i ILP zk equiv (Figure 11.4) were linearized using the big-M method(Griva et al., 2009, § 5.4.2).

Solutions were enumerated by adding, once solutions S is found, a constraint $\sum_{i \in S} p_i \neq |S|$

Supplementary materials provide complete executable examples of setup and execution.

Maximize $\sum_i p_i$	Objective function
Subject To:	
$z_k = \begin{cases} 1 & l_k = 1 \\ 0 & l_k \neq 1 \end{cases}$	Implementation below ILP zk equiv (Figure 11.4)
$p_i \rightarrow \sum_{k \in \mathcal{L}_i} z_k = 1$	Implementation below ILP pi implies (Figure 11.5)
Where:	
$\mathcal{L}_i$	Links in p_i
$p_i \in \mathbb{Z}_{\{0,1\}}$	1 if path i is in ILS else 0
$l_j \in \mathbb{Z}_{\{0,1\}}$	number of paths in the candidate ILS that use the indicated link
$z_j \in \mathbb{Z}_{\{0,1\}}$	1 if the corresponding $l_i = 1$, else zero

Figure 11.3: Overview of GILS setup for ILP solution for ILS. p_i and z_k are abstract, details below.

Implement $z_k = \begin{cases} 1 & l_k = 1 \\ 0 & l_k \neq 1 \end{cases}$

One of these for each l_k in ILP setup

$(l_k | z_k | \mathrm{zz}_k) \in \mathbb{Z}$

$0 \leq l_k$

$0 \leq \mathrm{zz}_k \leq 1$

$0 \leq z_k \leq 1$

$-M(1 - z_k) \leq l_k - 1 \leq M(1 - z_k)$

$-z_k - (M+1)\mathrm{zz}_k + 1 \leq l_k - 1 \leq z_k + (M+1)(1 - \mathrm{zz}_k) - 1$

Figure 11.4: ILP "macro" to implement $z_k = \begin{cases} 1 & l_k = 1 \\ 0 & l_k \neq 1 \end{cases}$

Implement Implication using big-M method.

IF $p_i = 1$, FORCE $x_i = b$

One of these for each p_i in ILP setup

$(p_i | \mathrm{zpi1}_i | \mathrm{zpi2}_i | x_i | b) \in \mathbb{Z}$

$0 \leq p_i \leq 1$

$0 \leq \mathrm{zpi1}_i \leq 1$

$0 \leq \mathrm{zpi2}_i \leq 1$

$-M \leq x_i \leq M$

$b = 1$

$b - M(1 - \mathrm{zpi1}_i) \leq x_i \leq b - \mathrm{zpi2}_i + 1$

$2p_i \leq \mathrm{zpi1}_i + \mathrm{zpi2}_i$

Figure 11.5: ILP "macro" to implement IF $p_i = 1$, FORCE $x_i = b$

11.4 Results

The results were computed on an Apple M3 Max 2023 laptop, using the MOSEK Optimization Suite (ApS, 2019) with no special features of the library utilized. The library was interfaced via Wolfram Mathematica (Wolfram Research, 2023b). We used the MOSEK library, rather than native Mathematica optimization capabilities, to ensure that computations could be repeated by others by using the readily available commercial optimization library.

Corpus models were tested by extracting their graphical representation and computing the ILP setup from the cycles of the graph. Auxiliary variable mathematical representations were not required.

GILS loop sets were computed for 678 models from our model corpus. Models with ≤ 70 loops were tested, as we arbitrarily bounded the computations to 60 seconds and determined experimentally that models with more than 70 loops were likely to exceed our (arbitrary) threshold. About 85% of the models in our corpus have $le70$ loops Corpus loop quantiles (Figure 11.6).

Loop counts are a rather abstract measure of model size. We correlate link count with loop count in Corpus vertices v loops scatter plot (Figure 11.7).

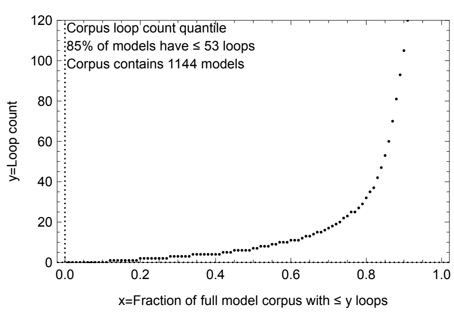


Figure 11.6: Model corpus loop counts.

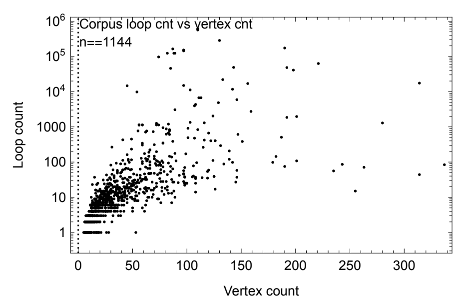


Figure 11.7: Corpus variables v loops scatter plot.

A typical output is shown in .

A plot of loop count vs GILS execution time is shown in GILS loops vs execution time (Figure 11.8).

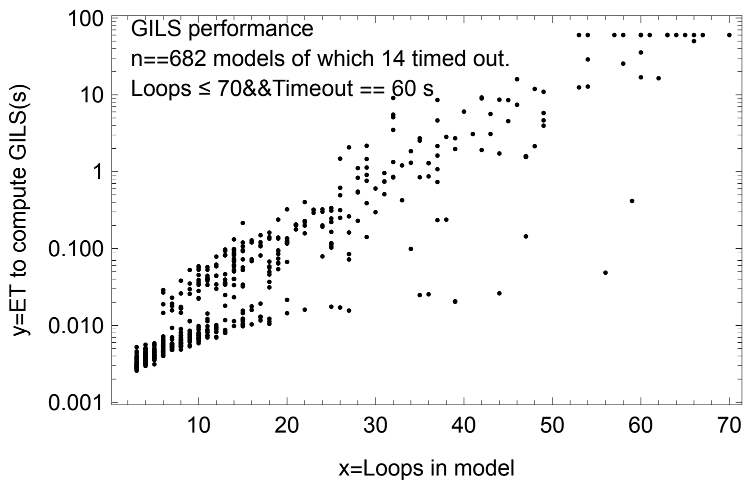


Figure 11.8: Execution time and success rate of GILS calculation. x is loops in model, y is calculation time in seconds. Computation time appears roughly exponential in model loop count. Points at y==60 seconds are timeout failures.

Of the 678 models for which we attempted to compute the GILS, 665 were successfully computed. The remaining 13 models all failed due to exceeding our 60 second timeout. None failed due to incorrect results or other algorithmic failure. Memory consumption was not formally measured, but total process size appeared to be under 30 GBytes at all times.

No library tuning was performed. It is likely that performance could be increased by tuning the library and tuning the formulation of the ILP setup.

We do not know if our corpus is representative of the global SD model population, but suspect its close but slightly biased towards low loop counts as it includes a number of small test or demonstration models. In this case we counted models that we could parse and execute. We have a few hundred more models that we can not parse and execute.

11.5 Discussion

GILS has four important properties not provided by KILS, SILS, or CILS:

- The gains of individual loops can be adjusted independently by a single link gain per loop, rather than many. The analyst will therefore find it easier to experiment with modified loop gains, as only a single link gain need be changed in order to change a single loop gain;
- GILS enumerates independent loop sets, so the analyst can view the model from several perspectives and therefore gain more insights into its loop behavior;
- and GILS accommodates several extensions to the basic definition of loop independence (e.g. exactly one independent gain, at least one independent gain, a bounded number of independent gains, paired independent gains of opposite polarity) within the same framework, so the analyst can generate more loop sets and so again gain more insight into model behavior.
- GILS separates the problem statement from the problem solution, and so it could be easily extended to utilize fast (but approximate) heuristic algorithms that relax the requirement for largest independent sets, without the need to modify other aspects of the framework or the usage patterns of the analyst.

The properties of KILS, SILS, CILS, and GILS are summarized in the Summary Table

We have demonstrated a new algorithm GILS that computes a stricter type of maximal set of independent loops: In this case we define independence to mean that each loop in the set has a link that is not in any other member of the set, which can be used to independently set the gain of that loop. Prior algorithms require solution of simultaneous equations to find the many links to change a single loop gain, and altering those many link gains will likely propagate into many other loops outside of the ILS.

This new type of ILS can be used with existing loop dominance algorithms, and has the added benefit that once a loop is identified for further study, a link is identified by which the gain of that loop, and only that loop (in the loop set), can be modified, perhaps to examine leverage points in the dominant loop.

(Kampmann, 2012) noted that full control of the gain of a loop may require manipulation of a positive link and a negative link, because there may not be any links that can flip polarities. Indeed, it is considered bad practice to have links that can reverse polarities. It is future work to extend GILS to identify pairs of links of opposing polarities that are unique to one loop in the ILS. KILS, SILS, and CILS do not have this capability.

Prior algorithms require several link gains to change in order to change the gain of exactly one loop in the ILS. GILS requires only one link gain to be changed. Since the ILS is not all of the loops in the model, it is likely that those gain changes will propagate into loops outside of the ILS. It is future work to contrast the model behavior impact due to gain changes using the two approaches.

Variations on GILS, all within the ILP optimization framework, could identify various types of subsets: Loops differing by exactly one link, at least one link, two links, and so on. It is future work to implement and explore the utility of this capability.

GILS as implemented herein is slower than CILS. This is the trade to achieve exact rather than approximate results and to use well-tested library code rather than custom algorithms. However, it can compute the GILS for models up to about 50 loops in a few seconds (GILS loops vs execution time), which may represent around 80% percent of the model population(Corpus loop quantiles) assuming our corpus statistics are reasonably representative of the model population as a whole.

KILS, SILS, and CILS, identify one independent loop set. But there are generally many independent loop sets. Is there a reason to believe that one loop set is as good as another as far as loop dominance analysis is concerned? Our work was partially motivated by the assumption that there may be insights to be had by examining other loop sets, but we have not supported this assumption. It is future work to examine the dominance statistics across all basis sets, and perhaps develop refined algorithms for selecting independent sets that will provide maximum incremental gain in insight per additional loop set analyzed.

There are numerous heuristics algorithms for solving ILP problems, and so the GILS problem setup used herein can be solved approximately and presumably more quickly. We do not explore that option here.

Loop count is substantially increased by the use of chaining macros such as DELAY3, and loops with several such macros multiply exponentially due to the many combinations of paths through the macros. It is future work to apply GILS without expanding the macros (or by "unexpanding" them before applying GILS). This would probably increase the fraction of models than can be analyzed by GILS in a timely fashion.

We believe our methods will be of particular interest to SD researchers and tool developers that implement SILS and CILS, because we open an option to generate multiple loop sets and then perform proprietary filtering to select loop sets of particular analytical power. Our methods will be of interest to general practitioners (once the computations are made easy to apply) because the practice of loop dominance analysis will likely provide more insights as a consequence of having multiple loop sets to consider. The ability to isolate loop gain changes to a single link will likely improve gain-change model analysis techniques.

12 AI Outperforms 43 grads developing CLD of Janis Groupthink

12.1 Summary

Augmented large language models (AugLLMs), transformers equipped with external tools for math, web search, and structured reasoning, can extract feedback-loop structure directly from prose. To test this claim, ChatGPT was prompted to answer a graduate question on Janis's theory of groupthink in three distinct personas: a "mid-B student," an "over-achiever," and a "polymath." Each persona produced an unconstrained narrative that the pipeline converted into a Causal Loop Diagram (CLD), then into an Attributed Directed Graph (ADG) for quantitative analysis.

The study compared these AI-generated diagrams with 43 human-produced CLDs and an instructor reference model, ranking them with a Borda aggregation of multiple loop-count metrics. Results showed that the over-achiever persona matched near-expert human performance, the mid-B persona clustered around the class median, and the polymath persona generated the most comprehensive diagram of all. Because the model operated in isolated sessions without access to student work, the ability to calibrate output complexity solely from the role prompt is notable.

Managing this multi-stage workflow required a formal Pipeline Algebra (PA). PA treats every step: prompt expansion, chat interaction, JSON extraction, graph attribution, and statistical scoring, as a typed morphism that can be composed, inspected, and rerun. Category-theoretic map and comap operators enforce data contracts and eliminate hidden state, providing both reproducibility and scalability that traditional orchestration frameworks struggle to guarantee.

Together, the empirical results and the PA framework demonstrate two key advances. First, frontier language models can now automate high-quality causal extraction from unstructured text, lowering the barrier to systems thinking. Second, a mathematically disciplined workflow is essential for trustworthy deployment of such models, enabling analysts and educators to move from narrative sources to CLDs-and eventually to quantitative System Dynamics models-with far less manual effort.

NOTE: This work has been revised and accepted for publication at IS2025.

12.2 Abstract

We explore the capacity of transformer-based LLMs augmented on the service side with e.g. math tool use, web search, and internal and/or interactive scaffolding (AugLLM) to generate and analyze Causal Loop Diagrams (CLDs) from unstructured natural text. Using ChatGPT, an augmented reflective generative pre-trained transformer, we address a graduate test question on groupthink, a social theory of group decision-making as written by Janis (1982), for which we have 43 student responses. We prompted the model in three distinct roles: "over-achieving student," "mid-B student," and "polymath." Each role elicited varying levels of causal abstraction, with the initial output unconstrained in format to ensure maximum computational resources are allocated to CLD elucidation. The responses were internalized into Annotated Directed Graphs for analysis.

We compare the AI-generated CLDs against 43 human-produced diagrams and a reference model using Borda aggregation of loop counts. Our findings indicate that the AugLLM adjusts its performance to match the role prompt, producing near-expert-level diagrams in the over-achiever mode and more measured outputs in the mid-B mode, while the polymath mode yields the most comprehensive diagrams, far exceeding the prompt minimum requirements. The AugLLM in roles of quiz taker had no access to the works of the other students, so this outcome is somewhat remarkable.

We introduce a novel, compositional pipeline algebra to manage this multi-stage process. PA orchestrates chat-based prompt engineering, structured output generation, and higher-order operations. This algebra leverages higher-order operators such as map and comap, to both express and execute the pipeline in a concise, scalable, and reproducible form.

This work demonstrates the potential of advanced LLMs for scalable causal analysis of unstructured text, while our pipeline algebra provides a reproducible framework to express and automate complex, chat-based workflows.

Keywords— System Dynamics, Causal Loop Diagram, Artificial Intelligence, LLM, Groupthink

12.3 Introduction

Humans often struggle with feedback-rich, non-linear systems, aligning with the assertion that "systems thinking is not a natural act" (Valerdi & Rouse, 2010). Even studies of groupthink such as (Park, 2000) and (Schafer & Crichlow, 1996) rely on linear cause-and-effect frameworks that overlook the essential role of feedback loops in this social phenomenon. Applying Systems Thinking to raw textual descriptions to extract structured representations that include feedback (specifically for our purposes the Causal Loop Diagram (Figure 12.1) (M. R. Goodman, 1972)). The CLD (section 12.7) is vital for understanding such complex phenomena (J. W. Forrester, 1971), (J. D. Sterman, 1994), though the CLD does have some limitations due to its qualitative nature (Richardson, 1986a), (Lane, 2008), (OECD, 2017).

Meanwhile, the transformative potential of the transformer architecture (Vaswani, 2017) has fueled recent advancements in large language models (Brown et al., 2020), (OpenAi, 2023). We use Augmented LLM (AugLLM) as a generic descriptor for advanced generative models (e.g. ChatGPT or Google Gemini) that incorporate additional data-handling and specialized modules on top of their core frontier LLM capabilities. By layering orchestration logic, domain-specific reasoning, chain-of-thought, agentics, and custom workflows

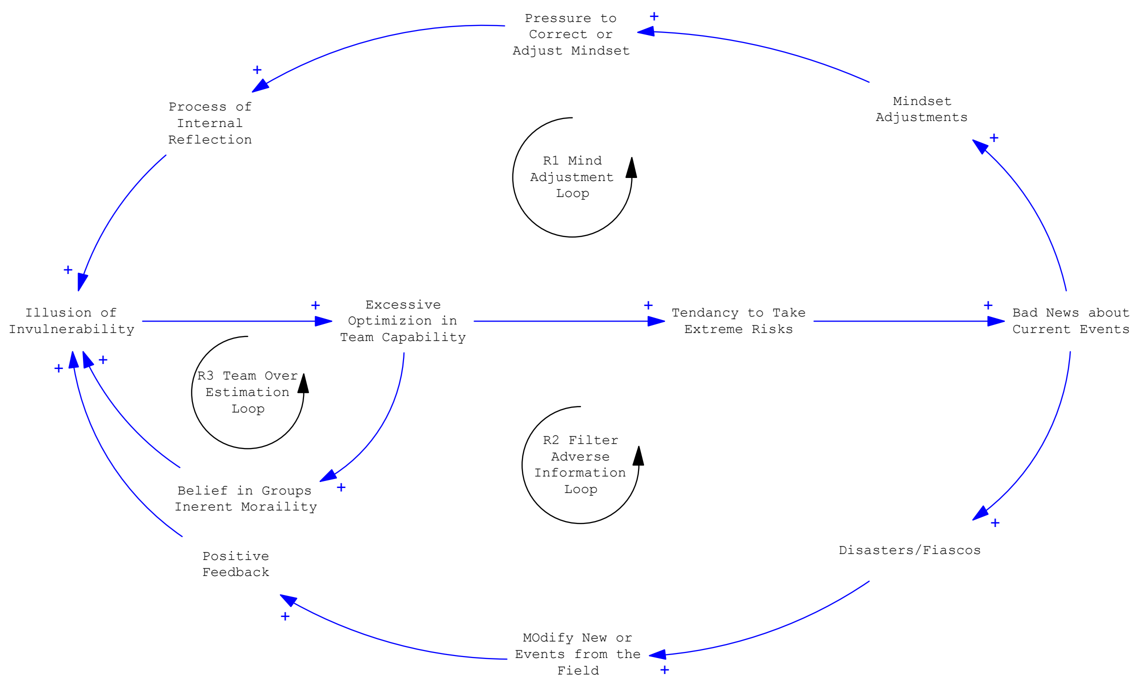


Figure 12.1: Simplified Groupthink Causal Loop Diagram. The CLD is distinguished by the causal links with polarity markings (+/-), labeled Reenforcing (R) or Balancing (B) loops of interest (R1, R2, ...) and rich feedback structure. Janis identifies no negative polarities and hence there are no balancing loops.

onto the base model, these Augmented LLM systems can tackle more complex tasks and offer enhanced reliability and interpretability. The AugLLM eases the job of the user (the "prompter") by taking care of details internally that would otherwise have required significant attention to orchestrate externally.

We demonstrate that the transformer architecture is highly effective at extracting causal relationships from unstructured textual input.

We explore the potential of a CLD/AugLLM synergy by applying these frontier models to Janis's "groupthink" theory (Janis, 1982) (Whyte, 1952), which features abundant causal relationships suitable for AI-driven CLD creation. We also leverage pre-existing student-produced diagrams for comparison, providing a strong basis for testing how well the AI captures feedback loops in social theory.

Specifically, we compare AI-generated CLDs with 43 student-produced diagrams and a reference model, using Borda aggregation of certain loop count measures to quantify the CLD quality. Repetitions over several days capture short-term variations in model outputs and ensure more robust findings.

To systematically modulate response complexity, we prompted the model under different "roles" (e.g., polymath, over-achieving student, mid-B student), systematically ablating AugLLM capability. This approach is conceptually related to persona- or instruction-based prompting (Zhang et al., 2018), (Ouyang et al., 2022), allowing fine-grained control over the AI's depth of reasoning.

We used also the "polymath" role to perform some ancillary tasks, such as developing a comprehensive CLD representation of groupthink and summarizing the groupthink text in order to minimize context window consumption for some tasks.

Generating and evaluating CLDs from raw text involves multiple steps, ranging from prompt engineering and AugLLM interaction to post-processing, diagram annotation, and final analysis. Without a structured workflow, these components can become difficult to manage, debug, explicate, and replicate, especially as more transformations or data sources are integrated. Common ML orchestration tools (e.g., TFX (Baylor et al., n.d.), Airflow, Kubeflow, Palantir Foundry, others) address this need by linking discrete tasks into pipelines. However, they typically rely on loosely defined interfaces, which limits post-hoc compositions and risks hidden data mismatches or inconsistent assumptions between stages, and lack formal explication.

We developed our Pipeline Algebra (PA) using category-theoretic principles (Spivak, 2014) and informed by the "notation is thought" (linguistic determinism) perspective (Iverson, 1980) to enforce rigorous input–output contracts and well specified categories("types") and enhance composability. Although PA draws inspiration from that perspective, it does not rely on it for its core functionality. The resulting framework provides a scalable, safer, and more maintainable workflow for augmented LLM-based CLD creation and manipulation. We employ PA both as a mathematical notation to formally describe pipelines and as an executable language to animate those pipelines.

We represent each CLD as an Attributed Directed Graph (ADG). The ADG extends a conventional directed graph by assigning zero or more attributes to vertices edges and the graph itself. This flexible structure captures the richer semantic information inherent in complex systems modeling. The attributes form part of the overall category and operator structure of PA: We do not use them as ad-hoc convenience annotations.

The results, clearly apparent in student rankings graph (Figure 12.2), indicate that the AI did indeed perform at the requested exemplary level when prompted to do so, and also at a mid-range level when prompted to do so. In this case we show the ranking with respect to the number of loops in the student CLD (a common measure of the information content in a CLD).

We hope that our work encourages more people to apply Systems Thinking (J. Monat et al., 2022) (Stroh, 2015) (Senge, 2006) (Sweeney, 2001) (Richardson, 1991) by using AI to lower the hurdle of developing and comprehending causal loop diagrams either as the qualitative end game, or on the path to development of quantitative System Dynamics models. The AI ability to summarize a large text in CLD form suggests utility in proto-quantitative text summarization.

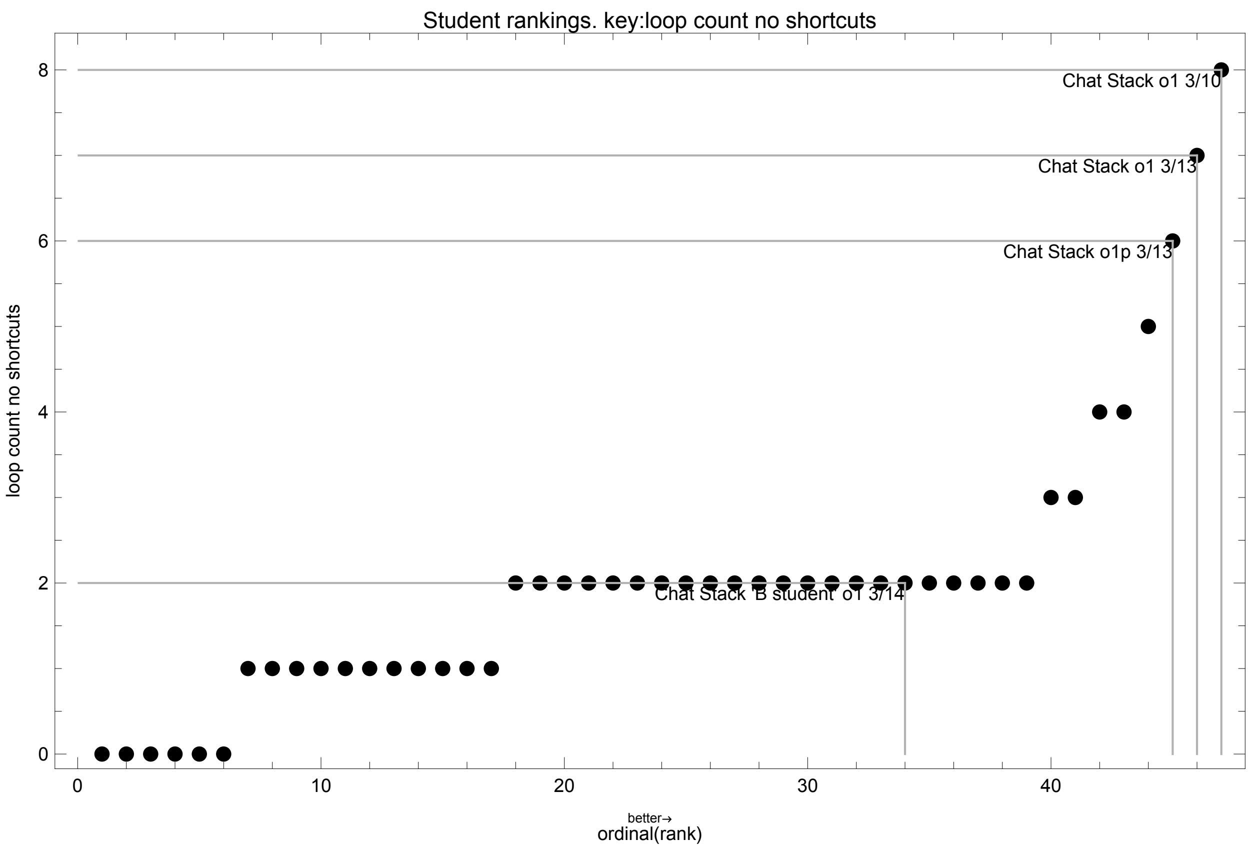


Figure 12.2: Student (live and AI) performance on test question. Ranked by number of loops in delivered CLD. AI used to convert textual responses to graphical form. The AI, in roll of high-acheiving student, took the top scores. There are several because a secondary experiment repeated the prompt over time to note AI stochastics. The AI was also prompted in roll of nominally performing student. It delivered arguably mid-range performance. The test question specified "identify at least two loops".

IRB Statement

A note on methodology: The Institutional Review Board (IRB) at Colorado State University determined that this project does not meet the federal definition of research with human subjects. Therefore, IRB approval was not required. The IRB response is in email from Kamran.Eftekhari_Shahroudi@colostate.edu to kirk.reinholtz@colostate.edu 6/27/2025 8:44AM.

12.4 Method

43 students over three semesters participated in a quiz asking that they identify at least two causal loops in (Janis, 1982, ch 8). A reference CLD was provided by the instructor. AugAI was prompted to answer the quiz question in three distinct roles (effectively as pseudo-students): B-level student; overachieving student; and overachieving professional polymath. The aggregated set of student, reference, and AugAI CLDs were scored on several measures, and ranked by an aggregation of two measures of loop count. The complete processing chain is shown in Morphisms (Figure 12.3).

Cross-contamination was avoided by processing of student responses in anonymous sessions, which are documented to retain no memory of the session after termination of the session and to not use the session for training or other state-exporting purposes. AugAI responses were also generated in individual anonymous sessions. Independence of results was therefore maintained.

When the AugLLM was tasked with answering the quiz question, three roles were specified: An over-achieving student; mid-B student; and professional polymath. As the AugLLM sessions responding to the quiz were isolated from all sessions that touched any student products, the AugLLM responses must establish the performance norms broadly, not with reference to known student responses.

All prompt/response and ancillary processing performed in this study was specified and executed in our pipeline algebra. The details of the algebra are outside the scope of this

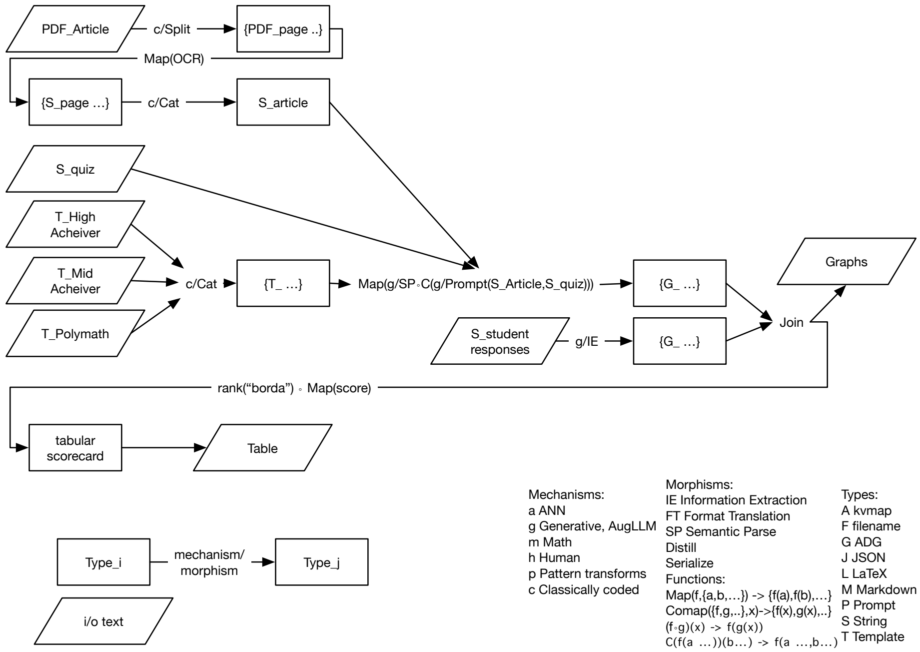


Figure 12.3: The janis ch8 images of pages is OCR'd into text for subsequent processing. The exogenous student responses are textual. The reference CLD is exogenous. The de-identified responses are processed into uniform ADG representation of the response CLDs. AugAI is prompted to answer the quiz question, in three distinct roles (effectively as pseudo-students): B-level student; overachieving student; and overachieving professional polymath. The aggregated set of student, reference, and AugAI CLDs are scored on several measures, and ranked by an aggregation of two measures of loop count.

article. But we provide an overview:

A typical prompt/response specified in pipeline algebra is shown in Pipeline Algebra Example (Equation 12.1). First a template is expanded, resulting in a problem statement. The problem statement is then used as a prompt in an anonymous session, resulting in a LaTeX-formatted response (in this case. JSON and native markdown are also often used). This snippet can be used as a function, or inline in a larger pipeline. The full pipeline used in this study is outlined in Morphisms (Figure 12.3). Rubrics for some high-order functions and naming conventions are also shown.

– NOTE some details elided
– NOTE Iterative prompting used to maximize TPU*s applied to CLD discovery

$$
\begin{aligned}
&(f \circ g)(x) \equiv f(g(x)) \\
&\mathrm{Map}(f)\{a, b, c, \ldots\} \equiv \{f(a), f(b), f(c), \ldots\} \\
&\mathrm{Comap}(\{f, g\})(x) \equiv \{f(x), g(x)\} \\
&\mathrm{Doprompt}(\\
&\qquad \mathrm{Key}(P_{problemstmt}) \\
&\qquad \mathrm{Session} \rightarrow \mathrm{Autonomous}, \\
&\qquad \mathrm{Endpoint} \rightarrow "chatgpt - o1" \\
&) \xrightarrow{\text{insert}} L_{problemstmt} \\
&\circ \mathrm{ExpandTemplate}(\\
&\qquad \mathrm{Key}(T_{base}), \\
&\qquad \{prompt \mapsto \ldots, article \mapsto \mathrm{Key}(myarticle)\}, \\
&) \xrightarrow{\text{insert}} P_{problemstmt} \\
&\circ \mathrm{Set}(\{T_{base} \mapsto \ldots, myarticle \mapsto \ldots, G_{smecld} \mapsto \ldots, F_1 \mapsto \ldots, F_2 \mapsto \ldots\})(\varnothing)
\end{aligned}
\tag{12.1}
$$

NOTES:
1. Hungarian notation used to improve readability:
   $A \mapsto$ kvmap
   $F_{name} \mapsto$ File name
   $G_{name} \mapsto$ ADG
   $J_{name} \mapsto$ JSON string or struct
   $L_{name} \mapsto$ LaTeX
   $M_{name} \mapsto$ Markdown native chat rv
   $P_{name} \mapsto$ prompt
   $T_{name} \mapsto$ Template

An Attributed Directed Graph (ADG) extends the conventional directed graph by assigning zero or more named attributes not only to each vertex and each directed edge, but also to the graph as a whole. These attributes can denote numeric values, string labels, logical flags, or any type of metadata relevant to the application domain (e.g., weights, capacities, probabilities, or version tags). This structure is especially useful in complex systems modeling, where edges and vertices may carry descriptive annotations (such as variable equations in system dynamics) and the overall model may have global properties or parameters (e.g., time stamps or overarching labels). The ADG thus combines the usual graph-theoretic relationships with flexible annotation mechanisms to capture richer semantic information about the system under study.

We use the ADG to represent a CLD and an SD model, among other things. All required information is associated with the ADG as attributes. We do not apply any ad-hoc convenience annotations in our work, though the implementation code does not prevent this.

The ADG is described formally and in the vernacular as follows:

$$
\begin{aligned}
&\text{Let } \mathcal{N} \text{ be a set of named attributes,}\\
&\mathcal{P}(\mathcal{N}) = \{S \mid S \subseteq \mathcal{N}\}\\
&\mathcal{V} \text{ be the set of vertices,}\\
&\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \text{ be the set of directed edges.}\\
&\text{An ADG is a 5-tuple } \mathcal{G} = \big(\mathcal{V}, \mathcal{E}, \phi_V, \phi_E, \phi_G\big),\\
&\phi_V : \mathcal{V} \to \mathcal{P}(\mathcal{N}),\\
&\phi_E : \mathcal{E} \to \mathcal{P}(\mathcal{N}),\\
&\phi_G : \{\bullet\} \to \mathcal{P}(\mathcal{N}),\\
&\text{where}\\
&\quad \phi_V \text{ maps each vertex } v \in \mathcal{V}\\
&\qquad \text{to a subset of named attributes in } \mathcal{N},\\
&\quad \phi_E \text{ maps each edge } e \in \mathcal{E}\\
&\qquad \text{to a subset of named attributes in } \mathcal{N},\\
&\quad \phi_G \text{ assigns a set of global attributes}\\
&\qquad \text{to the graph as a whole (indicated by } \{\bullet\}\text{).}
\end{aligned}
\tag{12.2}
$$

CLD's are scored on the following several criteria (results in student rankings table (Table 12.1)):

- Vertex Count

- Edge Count
- Loop Count
- Count of vertices in loops
- Loop count, triangle redundancies removed

A "triangle redundancy" is a short transitive closure in the CLD. This manifests in the CLD as a structure of a->b, b->c, and a->c. a->c is in some sense redundant: It states an effect already implied by the causal chain. Each such triangle doubles the number of loops traversing "b" vertex, and so these can have an outsized effect on the loop count as well as obscure the key causal structures of the CLD. Longer closures are of course possible and do sometimes appear. We chose to remove only triangles because they appear frequently in student works, are usually but not always uninformative, and greatly distort the loop count.

We define a CLD cycle as a path with no repetitions of vertices or edges other than the starting and ending vertices. The closed path is a loop. Loops are distinct only if they do not contain the same edges (i.e. the starting point is not significant).

The CLDs were ranked using a modified Borda aggregation of the loop count and loop count with triangle redundancies removed. We justify this choice as giving some weight to triangles on grounds that they are often but not always useless redundancies.The Modified Borda aggregation is performed by first ranking on each parameter (1 is lowest, n is highest, ties allowed) and then ranking by the sum of those ranks.

All experiment runs were defined and executed in Pipeline Algebra.

The pipeline was executed several times to refine the prompts, on ChatGPT o1 and ChatGPT o4. All such tests were performed wholly in anonymous sessions so that the AugLLM state was not permanently altered (e.g. did not retain "memories" of the results or prompts). The run for the record described herein was performed o/a 3/13/2025 on ChatGPT o1 in anonymous sessions.

Prompt responses were recorded verbatim so that subsequent queries of exactly the same prompt and session configuration can return exactly the same response (Prompt/response pairs are effectively idempotent) in order to provide repeatability of pipeline execution. In some cases this mechanism is suppressed, e.g. to observe response variance over time and to tune the pipeline.

Some results analysis was encoded into the pipeline. Others were computed using statistical and graphical methods applied to the final state of the pipeline execution.

12.5 Results

The AI, in roll of high-achieving student, took the top scores. There are several because a secondary experiment repeated the prompt over time to note AI stochastics. The AI ,prompted in roll of nominally performing student, delivered mid-range performance. student rankings graph, student rankings table (Table 12.1).

The essential topological structure of all responses (Student, Reference, AugLLM in its several roles, and Comprehensive) are shown in response graph signatures (Figure 12.4).

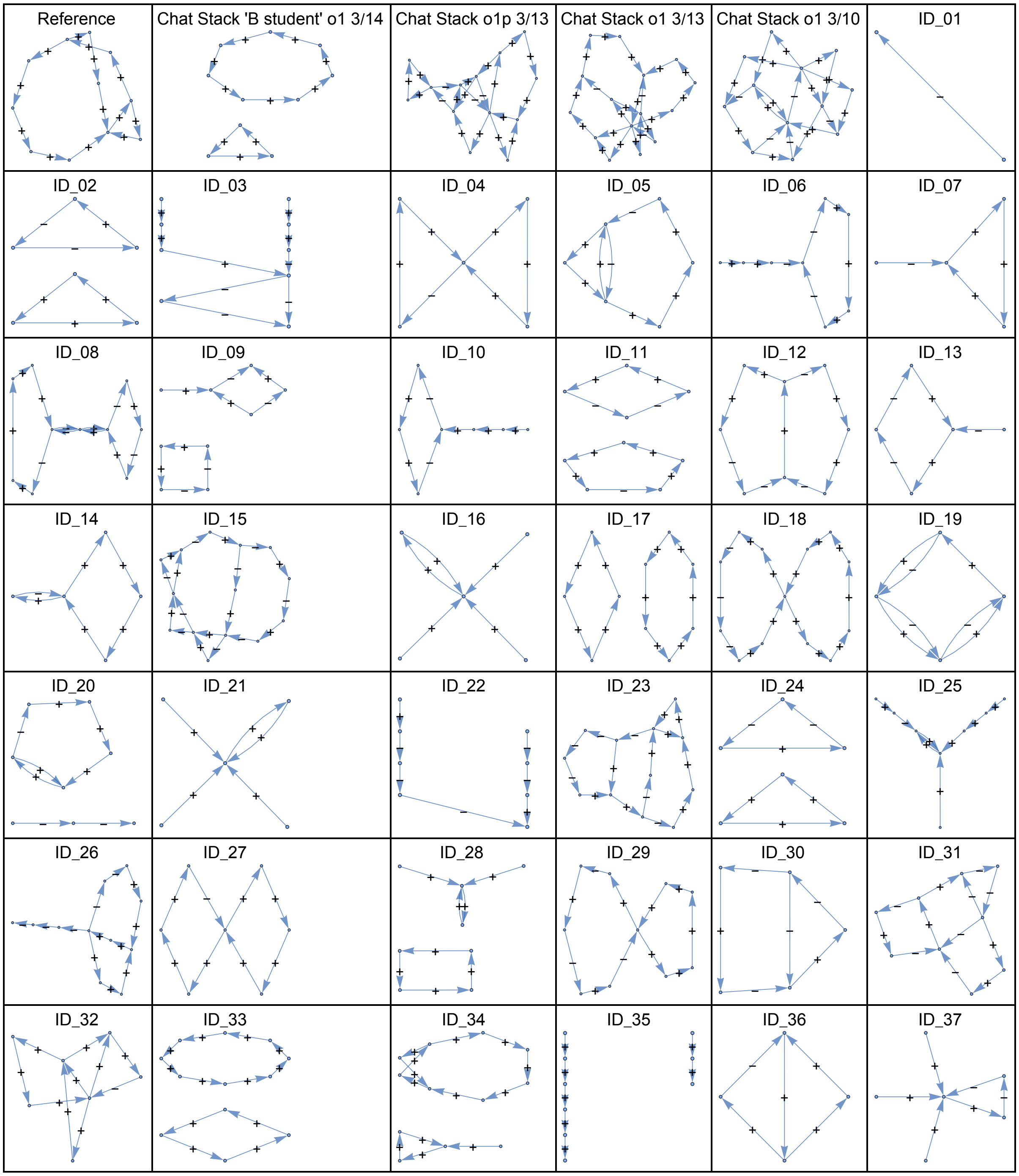


Figure 12.4: All graded CLDs, in no particular order. Vertex labels elided to emphasize topology.

Table 12.1

id	RANK (Borda)	loop count rank	loop count no shortcuts rank	vertex count	edge count	loop count	vertices in cycles count	closure2 count	loop count no shortcuts
Chat Stack o1 3/10	93	46	47	12	20	13	12	2	8
Chat Stack o1p 3/13	90	45	45	13	20	12	13	2	6
Chat Stack o1 3/13	90	44	46	15	21	8	15	1	7
ID_23	86	42	44	12	15	5	12	0	5
ID_31	84	41	43	9	12	4	9	0	4
ID_08	82	40	42	10	13	4	10	0	4
ID_34	79	38	41	12	13	3	11	0	3
ID_32	78	43	35	7	10	5	7	2	2
Reference	77	37	40	11	13	3	11	0	3
ID_42	75	36	39	7	7	2	7	0	2
ID_40	72	34	38	6	7	2	5	0	2
ID_15	72	47	25	14	18	16	14	3	2
ID_36	70	33	37	5	5	2	4	0	2
ID_33	68	32	36	12	12	2	12	0	2
ID_30	65	31	34	5	6	2	5	0	2
ID_29	63	30	33	10	11	2	10	0	2
ID_28	61	29	32	8	8	2	6	0	2
ID_05	60	39	21	6	8	4	6	1	2
ID_27	59	28	31	7	8	2	7	0	2
ID_26	57	27	30	11	12	2	8	0	2
ID_24	55	26	29	6	6	2	6	0	2
ID_20	53	25	28	8	8	2	5	0	2
ID_41	52	35	17	6	7	2	6	1	1
ID_18	51	24	27	13	14	2	13	0	2
ID_17	49	23	26	10	10	2	10	0	2
ID_14	46	22	24	5	6	2	5	0	2
ID_12	44	21	23	8	9	2	8	0	2
ID_11	42	20	22	9	9	2	9	0	2
ID_04	39	19	20	5	6	2	5	0	2
ID_02	37	18	19	6	6	2	6	0	2
Chat Stack 'B student' o1 3/14	35	17	18	10	10	2	10	0	2
ID_37	32	16	16	6	6	1	3	0	1
ID_25	30	15	15	8	8	1	2	0	1
ID_21	28	14	14	5	5	1	2	0	1
ID_19	26	13	13	4	7	1	4	0	1
ID_16	24	12	12	5	5	1	2	0	1
ID_13	22	11	11	5	5	1	4	0	1
ID_10	20	10	10	7	7	1	4	0	1
ID_09	18	9	9	9	9	1	4	0	1
ID_07	16	8	8	4	4	1	3	0	1
ID_06	14	7	7	8	8	1	5	0	1
ID_39	12	6	6	7	6	0	0	0	0
ID_38	10	5	5	10	9	0	0	0	0
ID_35	8	4	4	9	7	0	0	0	0
ID_22	6	3	3	8	7	0	0	0	0
ID_03	4	2	2	10	9	0	0	1	0
ID_01	2	1	1	2	1	0	0	0	0

12.6 Discussion

This study demonstrates how AI, specifically Augmented LLM-based chat stacks, can analyze complex texts and generate high-quality Causal Loop Diagrams (CLDs). By successfully extracting causal relationships from Janis's Groupthink, the AugLLM shows that it can significantly enhance the extraction of causal relationships from unstructured text and represent them as CLDs. This has important implications for systems thinking practitioners, tool developers, and researchers, as it lowers the barriers to creating causal models and provides new ways to explore feedback-driven systems described in unstructured natural language.

We have shown that the LLM can not only perform well in extracting causal relationships from text, but can attenuate its performance to approximate that of a student of specified capabilities, without reference to the corpus of student responses: It can apparently calibrate its performance against its vast knowledge base.

For systems thinking practitioners, this approach simplifies and speeds up the creation of CLDs. Traditionally, developing a CLD requires expertise in identifying causal connections and structuring feedback loops. AI can automate the first steps of this process, allowing practitioners to focus on refining and interpreting models rather than building them from scratch. This is particularly valuable for those working in policy analysis, consulting, and organizational design, where systems thinking is used to navigate complex, interdependent challenges.

For developers of systems thinking and System Dynamics tools, AI-assisted CLD generation could become an important feature. Integrating AI into modeling environments like Vensim, Kumu, or Stella could enable users to input reports, policy documents, or research papers and automatically receive structured causal loop diagrams.

Researchers in systems science and artificial intelligence can also benefit from this work. AI's ability to generate CLDs from text demonstrates new possibilities in natural language processing and causal inference. Additionally, the experiment's secondary test, where the AI was prompted multiple times over several days, provides insight into the stochastic nature of AI-driven analysis. This highlights opportunities for future research into AI consistency and accuracy in causal reasoning tasks.

Beyond systems thinking professionals, this capability has potential applications in education, policy analysis, and interdisciplinary collaboration. Educators could use AI-generated CLDs to help students visualize complex systems, compare AI-generated models to human interpretations, and refine their systems thinking skills. Policy analysts and strategists could leverage AI-generated causal maps for decision-making, and interdisciplinary teams could use them to align their understanding of complex issues.

Overall, this work shows that AI can play a significant role in expanding access to systems thinking methods. By making CLD creation more accessible, AI can help more people engage in causal analysis, foster collaboration, and enhance the exploration of complex systems. As AI technology continues to advance, its impact on systems thinking will likely grow, shaping the way we analyze and address interconnected challenges.

However, several limitations have been identified. First, AugLLM responses are inherently stochastic and subject to change as the underlying models evolve. Second, achieving sufficient aggregate computational effort may require labor-intensive prompt engineering using chaining, iterative and scaffolding methods, each incurring HITL overhead. Third, hallucinations remain a persistent and unpredictable risk. Collectively, these issues pose significant challenges for verification and validation (V&V), and strongly indicate the need for meticulous post-hoc review of all AugLLM-generated artifacts. Fortunately the industry is devoting significant attention to these challenges and so the situation is likely to improve substantially as time goes on. Regardless, we maintain that researchers bear full responsibility for the meticulous verification of all AI-generated content, down to the last detail.

For future work, we hope to explore bridging the gap between qualitative CLDs and quantitative System Dynamics models using AugLLM. We expect it will ease several difficulties, including the identification of "reasonable" equations to assign to the auxiliary variables; identifying integrating (Stock) variables; and determining likely reference behaviors that can be used to evolve the model. We expect AugLLM to be combined with optimization methods such as Evolutionary Computation (Jacob, 2001) and similarity methods referencing a corpus of System Dynamics models to fully bridge the gap. Preliminary informal experiments have been encouraging.

The CLD quality ranking measures utilized in this study are considered reasonable, though unlikely to be optimal. A more rigorous evaluation framework remains an open area for future research. Classical metrics warrant further exploration, including the distribution of loop lengths, the proportion of nodes and edges participating in feedback cycles, cyclomatic complexity from graph theory, and structural properties such as loop connectivity and overlap. In addition, more information-theoretic and distributional approaches may prove valuable e.g. link or loop entropy or entropy computed relative to probability distributions derived from reference corpora.

12.7 The CLD

What is the CLD Causal Loop Diagrams (CLDs) originate from the system dynamics framework developed by Jay Forrester and colleagues at MIT (J. Sterman, 2000). They are conceptual tools designed to visualize and analyze the feedback loops and interdependencies within complex systems. Variables appear as nodes, while directed arrows indicate causal influences. Positive (+) or negative (–) signs show how changes propagate, reinforcing or counteracting one another. Although CLDs do not yield precise quantitative predictions, they help identify structural patterns, highlight leverage points, anticipate unintended consequences, and establish a shared understanding of the broader context. These insights can guide the development of detailed simulation models and inform evidence-based policies, particularly in cases where complex feedback effects drive system behavior.

Why the CLD In the process of selecting a suitable representation method for this work, both concept maps ((Novak, 2010)) and Causal Loop Diagrams ((J. Sterman, 2000)) were considered. Concept maps are well regarded for capturing broad, associative relationships among ideas, making them a strong candidate for initial brainstorming and revealing general conceptual linkages. However, the CLD is particularly focused on how changes propagate via feedback through interconnected components. As the impetus for this work was a wealth of data available due to a class assignment to develop a CLD representing elements of the loops identified by Janis, we selected the CLD for this work. The CLD also furthers our goal of developing an AI-based quantitative path from the CLD to an SD model. (M. R. Goodman, 1974).

Advances in AI, particularly large language models like ChatGPT or Google Gemini, now offer opportunities to integrate CLDs into an iterative, conversational workflow. While CLDs alone provide structural clarity and scientists contribute domain expertise, AI can rapidly test multiple scenarios, identify missing elements, and help reconcile conflicting perspectives by analyzing the relationships depicted in the diagram. For instance, if a policymaker wonders how adjusting subsidies for energy storage might affect carbon emissions and grid reliability, the AI can propose variables, infer causal links, and highlight overlooked feedback loops, ultimately transforming a simple chain of influences into a nuanced, multi-loop diagram. This synergy between human judgment, scientific evidence, and AI-driven exploration enables more refined and transparent analyses than using CLDs or AI in isolation.

However, leveraging CLDs in this way also requires acknowledging certain challenges. Policymakers, scientists, and AI developers must invest time to learn the approach, and ensuring that all stakeholders accurately interpret the diagrams can be demanding. There is also a risk of over-relying on AI-generated insights without sufficient domain-specific scrutiny (Messeri & Crockett, 2024). https://www.nature.com/articles/d41586-025-01067-2 Yet, by using CLDs as a shared language, these risks are to some extend mitigated. Everyone can see why the AI suggests certain interventions, which feedback loops support its reasoning, and where uncertainties lie.

We conclude that Causal Loop Diagrams offer a powerful means of bridging communication gaps between policymakers, scientists, and AI systems. CLDs help decision-makers understand complex phenomena at a systemic level by visualizing feedback loops, clarifying causal pathways, and providing a transparent framework for iterative exploration, When coupled with scientific input and AI-driven analysis, they form a cohesive platform for informed policymaking, ensuring that interventions are guided by evidence, critical reflection, and an appreciation for the intricate feedback effects that shape real-world outcomes.

Formal Representation of CLD A Causal Loop Diagram can be exactly represented as an attributed directed graph (ADG). In the case of CLD, the attributes certainly include the link polarity. We also use various augmented CLD representations, in which case the attribute might include an indication of link delay, magnitude of effect, and other link-specific attributes.

Formally,

Definition (Attributed Directed Graph): An *Attributed Directed Graph* (ADG) is defined as a tuple:

$$G = (V, E, A)$$

where:

- V is a finite set of vertices (nodes).
- $E \subseteq V \times V$ is a set of directed edges, where each edge (u, v) represents a directed relationship from vertex u to vertex v.
- $A : E \to L$ is an attribute function assigning labels or attribute values from some set L to edges. The attribute set L may be multidimensional, containing edge weights, labels, or domain-specific information.

12.7.1 CLD JSON Schema

We often use JSON as the target syntax of AugLLM prompting, as it is reasonably well supported by our chosen LLM and is relatively easy to parse on the client size. It does, however, seem to consume significant LLM resources and so we often use a distinct scaffolding prompt that is tasked specifically to perform the conversion.

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "https://example.com/cld.schema.json",
  "title": "Causal Loop Diagram Schema",
  "description": "A JSON schema for representing a Causal Loop Diagram (CLD), including
      variables and their causal links.",
  "type": "object",
  "properties": {
    "variables": {
      "type": "array",
      "description": "List of variables in the system",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string",
            "minLength": 10,
            "description": "A descriptive name (>= 10 characters) for the variable, e.g.
                'Greenhouse Gas Emissions Rate'"
          },
          "description": {
            "type": "string",
            "description": "An explanation of what the variable is and where it originated"
          },
          "units": {
            "type": "string",
            "description": "Industry-standard unit for the variable or 'DimensionlessUnit' if no
                units apply"
          }
        },
        "required": ["name", "description", "units"]
      }
```

```
    },
    "links": {
      "type": "array",
      "description": "List of causal links connecting the variables",
      "items": {
        "type": "object",
        "properties": {
          "source": {
            "type": "string",
            "description": "The name of the source variable for this link"
          },
          "destination": {
            "type": "string",
            "description": "The name of the destination variable for this link"
          },
          "polarity": {
            "type": "string",
            "enum": ["+", "-","+/-","unknown"],
            "description": "Indicates whether changes in source cause a same-direction (+) or
                opposite-direction (-) change in the destination"
          }
        },
        "required": ["source", "destination", "polarity"]
      }
    }
  },
  "required": ["variables", "links"]
}
```

12.8 Prompting Styles

Persona Persona-based prompting refers to the practice of explicitly assigning the AI model a particular role, expertise, or behavioral profile at the outset of an interaction. By framing the chat stack as, for example, an academic STEM researcher that is expert in mathematics, system dynamics systems thinking, public policy, and causal loop diagrams, we establish a context that directs the model's responses toward a specific style of reasoning and knowledge domain. We may also assign a role, e.g. "You are a scientist performing research on ..." and a voice, e.g. "Write your results in a STEM academic article style...", in order to further focus the attentions of the AI on the task at hand.

Iterative Iterative prompting is a prompting approach that divides complex tasks into a series of smaller, focused steps, each generally receiving its own unit of computational resources. The user provides each step as a distinct prompt and builds on the answers from previous stages. This method fosters comprehensive exploration, incremental refinement, and more accurate final results. Multiple prompts also provide access to a larger aggregate “compute budget,” enabling deeper analysis and reducing the risk of overlooked details.

Scaffolded Scaffolded prompting refers to the use of structured, multi-step prompts that guide a language model through intermediate reasoning stages to solve complex tasks more effectively than single-shot queries. This scaffolding can be externally imposed by the user, through explicit task decomposition, sequential subgoals, or interactive workflows, or internally generated by the model as it organizes its reasoning into a coherent multi-stage response. When applied externally, scaffolded prompting typically increases aggregate computational effort by eliciting multiple inference passes across subtasks. However, it also introduces a greater HITL burden, as users must engineer and interpret each stage of the process. In both forms, scaffolded prompting supports reliability, interpretability, and structured problem-solving by emulating cognitive scaffolding strategies from human learning.

Chain-of-thought Chain-of-thought prompting is a prompting approach that requires the model to articulate the intermediate steps involved in reaching its final answer. This method reveals the logic, assumptions, and inferences behind each conclusion, often leading to more precise and transparent outcomes.

Socratic Socratic prompting is a questioning method that draws on the Socratic tradition. The user poses a sequence of targeted prompts, prompting the model to refine its reasoning, uncover subtle details, or resolve inconsistencies in successive stages. This step-by-step structure aims to clarify insights progressively.

Self-reflection Self-reflection prompting is a style that instructs the model to evaluate and critique its own reasoning before producing a final answer. The model examines potential errors or omissions and considers alternative perspectives, fostering deeper analysis and improving overall reliability in the process.

Self-consistency Self-consistency prompting is defined as issuing the exact same prompt multiple times, gathering the several independent outputs (or "samples"), and then aggregating those outputs, perhaps by majority vote or some synthesis step, to reach a final answer. This helps mitigate the randomness of sampling and can yield more reliable results by leveraging consistent patterns that emerge across multiple responses.

13 LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra

13.1 Introduction

Humans are notoriously poor at reasoning about feedback-dominated, nonlinear phenomena, a limitation captured in the observation that "systems thinking is not a natural act" (Valerdi & Rouse, 2010). Linear, one-way causal narratives routinely obscure the reinforcing and balancing loops that govern real-world behavior (J. W. Forrester, 1971). We have proven adept at creating systems exhibiting such behavior. Unfortunately the real-world behavior is frequently undesirable, but it is self-evident that we are not as adept at predicting and mitigating the undesired behaviors as we are at creating the systems in the first place. In perhaps one of the greatest of vicious loops, we are creating ever more and ever more dangerous such systems.

Systems Thinking (ST) (Kim, 1999; D. H. Meadows & Wright, 2008; J. P. Monat & Gannon, 2015) is a general term for attempting to comprehend the behavior of feedback-laden non-liner systems. But ST is itself hard. Tools have been developed to aid our attempts at ST, in particular the qualitative Causal Loop Diagram and its quantitative relative System Dynamics. but until now most of the tools mechanized the processes, but did not much aid the intellectual effort itself.

Causal Loop Diagrams ((J. Sterman, 2000, ch 5)) elucidate the structure that leads to the feedback-driven behavior of the system, and helps to identify leverage points (J. W. Forrester, 1971; M. R. Goodman, 1972; J. D. Sterman, 1994). Constructing a high quality

CLD, however, is labor-intensive: an expert must review source material, define variables with well formed polarity-aware names, identify and close impactful feedback loops, and justify every link. The required effort and skills in wrangling domain specific, mathematical, and general software tools consequently limits CLD and SD uptake among engineers and policy analysts.

But transformer-based large language models (LLMs) can alter that cost profile. The architecture that (Vaswani, 2017) underpins frontier systems such as GPT-4 (OpenAi, 2023) can extract, correlate, and synthesize content at great scale. When an LLM is paired by its developer (Yao et al., 2022) with retrieval utilities, deterministic calculators, and task breakdown and orchestration logic, it becomes capable of exploiting its hardwired knowledge base as well as searching the web and various databases and even writing code to assist itself in its reasoning. We refer to all of these as "TALM" (Tool-augmented language model) in this paper, which is intended to cover what is also called "Agentic AI", "Agentic LLM", "Composite AI", "Multi-tool CoT" (Chain of Thought), "ReAct Agents", "Toolformer", "LLM+", "agentic tool use", and others.

Growing empirical evidence (Arndt, 2023; Gupta et al., n.d.; Liu & Keith, 2025) indicates that large-language-model architectures such as ChatGPT can autonomously abstract and integrate heterogeneous textual inputs into semi-formal, graph-based knowledge structures, most notably causal loop diagrams and Toulmin-style argument schemata, while achieving accuracy that approaches expert baselines.

Such a system can deliver, for our present purposes, two complementary artifacts: (i) a source-constrained CLD whose links are justified solely by the focal text, and (ii) a knowledge-augmented CLD that extends that skeleton with variables, drivers, and risks obtained from the LLM knowledge base and external (e.g. the public Web) sources.

Automating CLD generation involves multiple coupled stages: text extraction, causal-relation mining, polarity assignment, loop synthesis, diagram layout, knowledge augmentation, validation, and iteration with human domain experts. General-purpose orchestration frameworks such as TFX (Baylor et al., n.d.), Airflow, and Kubeflow connect these steps but rely on loosely specified interfaces that can hide data mismatches and are optimized for execution at scale, not epistemics or pedagogic clarity. We propose that an executable, mathematical style of workflow formalism that enforces type safety, records provenance, guarantees reproducibility, integrates AI, mathematical, and general algorithmics, and communicates ideas in epistemically transparent rather than programmatic terms will improve CLD and SD uptake.

This study introduces Pipeline Algebra (PA), a category-theoretic pipeline workflow notation in which every stage is a typed morphism with explicit domain and codomain (Spivak, 2014). PA is, in effect, statically and strongly typed. All PA operators are capable of idempotency: executing a stage with identical inputs can always yield identical outputs, even when the underlying TALM is stochastic. Types and Morphisms (Figure 13.1) provides an overview of the PA-orchestrated workflow, highlighting the hand-off between deterministic operators, TALM services, and subject-matter-expert review. In the spirit of Iverson's dictum that "Notation is thought" (Iverson, 1980), a PA script functions simultaneously as formal human-readable documentation and executable specification, producing a deterministic audit trail suitable for replication and peer review (Peng, 2011).

CLDs produced by the pipeline are stored as Attributed Directed Graphs (ADGs). An ADG extends the directed graph by attaching formal typed attributes to vertices, edges, and the graph as a whole. Crucially, the same structure can also encode the stock-and-flow constructs of a quantitative System Dynamics (SD) model. Using one uniform representation therefore enables a future, automated evolution from qualitative CLD to fully parameterized SD model while preserving provenance at every step.

A subject-matter expert (SME) remains essential. The TALM supplies breadth, tireless text mining, global knowledge recall, and the ability to connect distant facts, whereas the SME provides depth: contextual judgement, domain heuristics, and strategic intent. Effective diagram construction proceeds iteratively: the SME frames a prompt or partial

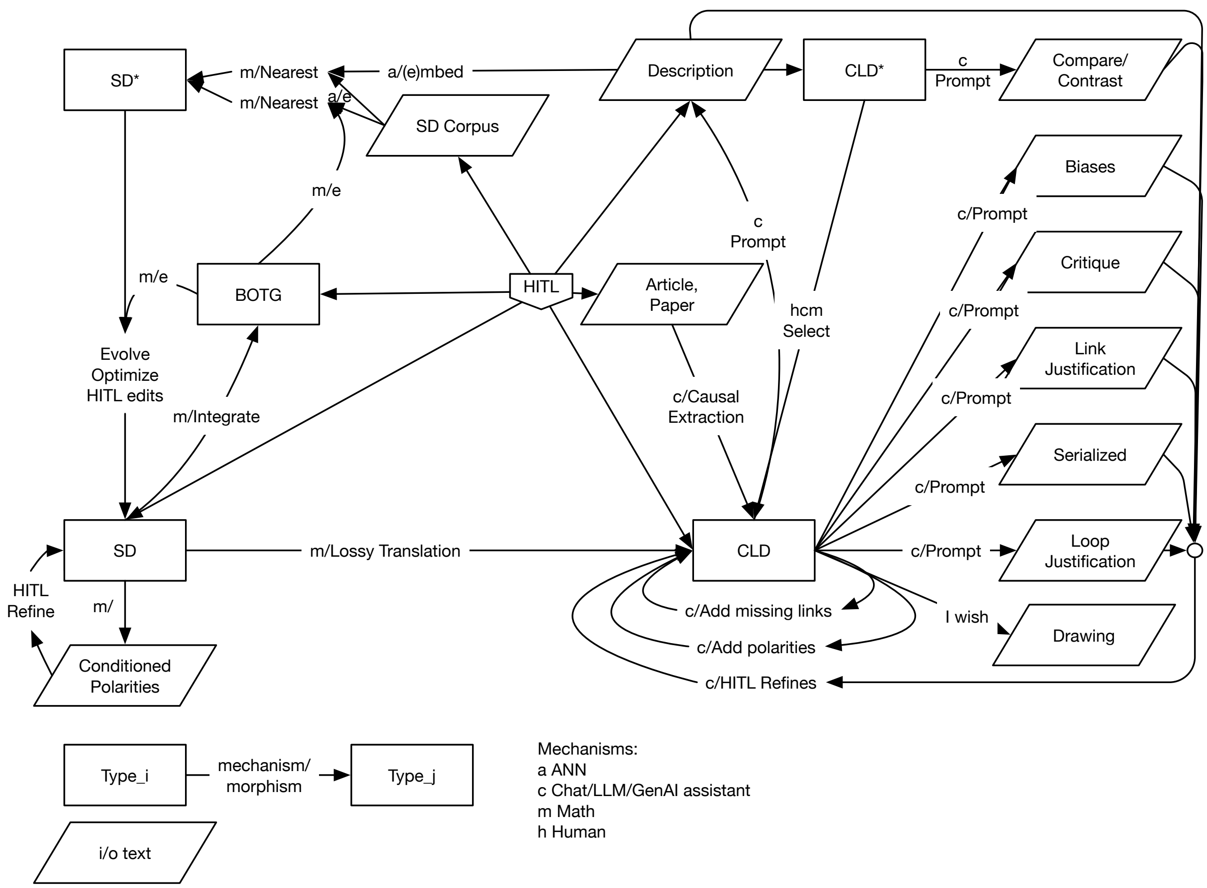


Figure 13.1: Much of our work is described and implemented as morphisms of types. Here we present the major morphisms. Types are usually Associative Arrays or lists thereof. Morphisms add KV pairs, delete keys, or mutate values. Most of the morphisms are further described herein using our Pipeline Algebra. The implementation closely follows the notation. Most morphisms are stateless idempotent operators on AA's or compositions (including high level functions e.g. map, comap, composition, lambdas, curry, meta-algorithmic operators, error handling) thereof. Operators that interact with stochastic services e.g. ChatGPT or Gemini may memoize results so as to provide apparent idempotence.

diagram, the AI expands it, and the SME validates or refines the output and/or the pipeline itself. This division of labor concentrates expert effort on conceptual issues while delegating repetitive extraction and pattern-matching tasks to the machine.

This paper pursues four objectives. First, we formalize Pipeline Algebra and show how its idempotent, typed operators convert a tool ensemble including TALM, symbolic and pattern matching, numeric mathematical capabilities, and local AI and ordinary algorithms into a dependable collaborator. Second, we demonstrate practical utility by generating both source-constrained and knowledge-augmented CLDs for the subject energy-transition case study, achieving turnaround times measured in minutes rather than hours or days. Third, we report an SME evaluation that identifies weaknesses in the CLD that highlight the need for expert evaluation and refinement as well as the value of pipeline improvement based upon SME analysis. Finally, the refinements are applied to the PA pipeline, resulting in an improved CLD and a pipeline that will reflect the SME input into any future instantiations.

Collectively these results support the thesis that PA-orchestrated TALM lowers the effort barrier to feedback-centric reasoning without displacing expert oversight, and lays the groundwork for future automated evolution from CLD to quantitative SD modeling. Future work will provide quantitative accuracy benchmarks against reference diagrams, extend the CLD→SD transformation to full simulation, and test the pipeline across multiple domains and reviewers while developing automated variable-naming and polarity checks. These extensions are essential to establish the general validity and practical scalability of the approach.

13.2 Materials and Methods

The work reported herein is both described in and performed by execution of Pipeline Algebra (subsection 13.2.1)Pipeline Algebra. PA provides an overview of PA, and presents the PA formulation that performed the work we report herein.

13.2.1 Pipeline Algebra

Pipeline Algebra (PA) is a *category-theoretic workflow language* (Spivak, 2014) whose atomic unit is the **typed morphism**: a pure function declared with an explicit domain and codomain. Any morphism may invoke a large-language model, run deterministic code, launch a cloud batch job, or delegate to a human, yet it must honor its type contract: given an input of type τ_1, it yields an output of type τ_2 and it must be a pure function from a computational perspective, hence leaving global compute state untouched. There is a subtlety in this purity: An TALM conversation may take place across several interactions in a session, which obviously modifies global state. We treat this formally as a monadic construct with the session key representing the monad state that is mutated across prompt/response pairs. Comprehensive morphisms are constructed by composing operators using higher-order functional programming operators (Hughes, 1989) such as composition, map, and comap, and functional transformation operators such as symbolic differentiation. Higher-order functions are generally intended to be useful for any application of PA, while atomic operators are expected to be a mix of general and domain-specific capabilities.

PA uses a state-passing style (Moggi, 1991) where each operator has a single argument (an associative array) which contains state that must include the named input instances for the operator, and the return from the operator is the input state with the operator result added to the state. For example, in PA Cheatsheet (Equation 13.1)(5) this is shown explicitly.

Some operators such as Comap create a list of states (one from each comapped function). For notational convenience we assume that the function return values can be merged without issue, and do so automatically within the Comap operator. A richer options is available to Merge explicitly, but that we do not use or further describe that capability here. We use an arrow notation such as follows to indicate the objects (e.g. $F_{\text{vensimfile}}$), categories or types (e.g. F) and morphisms (the arrow).

We use a much denser mathematical form for more complex morphs, shown in PA Cheatsheet (Equation 13.1)

$$
\begin{aligned}
&\text{– NOTE Notional: Details elided or simplified for clarity.}\\
&\text{– Some operator names changed for clarity.}\\
&\text{– Explicit state threading pattern is used.}\\
&\text{– State in Keyword/Value list e.g. } \{c_1 \mapsto 3\}\\
&1 \quad (f \circ g)(x) \equiv f(g(x))\\
&2 \quad \mathrm{Map}(f)\{a, b, c, \ldots\} \equiv \{f(a), f(b), f(c), \ldots\}\\
&3 \quad \mathrm{Comap}(\{f, g\})(x) \equiv \{f(x), g(x)\}\\
&4 \quad \mathrm{f}(\ldots, rv \mapsto \cdot) \equiv f(\ldots) \to \cdot\\
&5 \quad (\mathrm{Plus}(a_1 \mapsto 2, a_2 \mapsto \mathrm{Key}(c_1)) \to p_1)(\{c_1 \mapsto 3\}) \to \{c_1 \mapsto 3, p_1 \mapsto 5\}\\
&\quad F : A \to B \equiv A \underset{\mathrm{F}}{\to} B\\
&\quad \text{– } \{c_1 \mapsto 3\} \text{ is input state. } \{c_1 \mapsto 3, p_1 \mapsto 5\} \text{ is output state.}\\
&6 \quad (\ f)(x_1, x_2) = \mathrm{g}(\mathrm{F} \mapsto \mathrm{x}_1, \mathrm{G} \mapsto \mathrm{x}_2)\\
&\text{– Hungarian notation used to improve readability. It has no formal semantics:}\\
&\quad A \mapsto \text{kvmap for state passing e.g. } \{c_1 \mapsto 3, p_1 \mapsto 5\}\\
&\quad F_{name} \mapsto \text{File name}\\
&\quad G_{name} \mapsto \text{ADG}\\
&\quad J_{name} \mapsto \text{JSON string or struct}\\
&\quad L_{name} \mapsto \text{LaTeX}\\
&\quad M_{name} \mapsto \text{Markdown native chat rv}\\
&\quad P_{name} \mapsto \text{prompt}\\
&\quad S_{name} \mapsto \text{String. May also be J|L|M|P|T}\\
&\quad T_{name} \mapsto \text{Template}
\end{aligned}
\tag{13.1}
$$

We designed PA as a high-density mathematically-oriented notation, on the premise doing so would facilitate epistemic exploration on the premise that "Notation is thought" (Iverson, 1980) and "Civilization advances by extending the number of important operations which we can perform without thinking of them." (Whitehead, 1911). a PA script functions simultaneously as formal, human-readable epistemic documentation, pedagogic exposition of method, and instantiates to generate the the reported results.

The formal, mathematical syntax and semantics of PA facilitates meta-manipulations. Meta-algorithmic methods (Simske, 2013) may be applied to optimize for real-time cost, speed, or accuracy targets.

We speculate that the PA compositional forms are relatively amenable to TALM comprehension and manipulation because the LLM loss rewards first capture n-gram-like continuities (because they are the strongest, most ubiquitous statistical cues in text) and hierarchal queues appear far less often and matter only many tokens later, so the gradient signal that would teach stable recursion (e.g. expression trees) is weaker and noisier. If true, then PA may facilitate novel TALM-based meta-algorithmic methods. It is future work to validate our speculation.

Functions can be composed as seen in PA Cheatsheet (Equation 13.1)(6). Functions behave as substitutions: Any instance of the LHS is, in effect, replaced with the RHS with the argument patterns replaced by their actual values. This can be used to increase notation density but provides no fundamentally new semantics.

PA defines denotational semantics; any runtime that respects the data-dependency DAG and memoization invariants is sound. This facilitates two important design choices: DAG execution (Lee & Parks, 1995; Mac Lane, 1998) (the basis of several large scale open source and commercial workflow systems such as Apache Airflow and Amazon Sagemaker); and memoization (Michie, 1968). DAG execution enables effective cloud-scale execution by identifying the minimal sequencing necessary to insure causality (i.e., "compile" PA into Airflow or Sagemaker); and memoization provides deterministic results in the face of stochastic operators (e.g. an TALM prompt response). Deterministic results greatly simplify scientific verifiability (Peng, 2011) of computed results, and can tactically improve performance by performing make-like (Feldman, 1979) optimizations. It is future work to perform such optimizations.

DAG execution is simpler to implement when the DAG is static and can be determined syntactically. All PA constructs allow this. It sometimes complicates the syntax (e.g. template expansion requires that the interpolated variables are mapped to state variables syntactically (i.e. not looked up at runtime).

Categorical semantics do more than tidy notation. Let $\mathbf{P}$ be the category whose objects are PA data types and whose morphisms are the pure functions we admit, and let $\mathbf{Exec}$ be the category of executable DAG nodes under composition. We define a semantics functor

$$[\![-]\!] : \mathbf{P} \longrightarrow \mathbf{Exec}$$

that maps each atomic operator to its runtime task and preserves identities and composition. Because $[\![-]\!]$ is functorial, any equational law proved in $\mathbf{P}$, for example

$$\mathrm{Map}\, f \;\circ\; \mathrm{Map}\, g \;=\; \mathrm{Map}\,(f \circ g),$$

carries over to the compiled DAG, enabling correctness-preserving optimizations such as fusion and parallel re-ordering. For execution we work in the Kleisli category $\mathbf{Kl}(S)$ of the state monad S, so operator purity is maintained at the semantic level even while session state threads through the computation.

13.2.2 Templates

PA uses a simple template notation, e.g.

$$\begin{aligned} &\mathrm{ExpandTemplate}(\\ &\quad T \mapsto \text{"Hello, `whoever`!"}, \\ &\quad args \mapsto \{whoever \mapsto \mathrm{Key}(a1)\} \\ &)(\{a1 \mapsto \text{"world"}\}) \to P_{prompt} \end{aligned} \tag{13.2}$$

The above will dereference $a1$ from A, yielding the string "world" which is assigned to the template variable "whoever", which is interpolated into the template where the variable is found enclosed in single quotes "'". The result, in the state S_{prompt}, is "Hello, world!".

Expansion is iterated until a fixed point is achieved. Interpolated variables may therefore themselves contain interpolated variables. Loops are fatal.

13.2.3 Causal Loop Diagram

A causal loop diagram (M. Goodman, 1973) (See Notional CLD (Figure 13.2)) is a conceptual diagram made up of variables connected by arrows with polarity signs (+/−) that indicate the direction and nature of causality, forming feedback loops that represent the structure of a dynamic system.

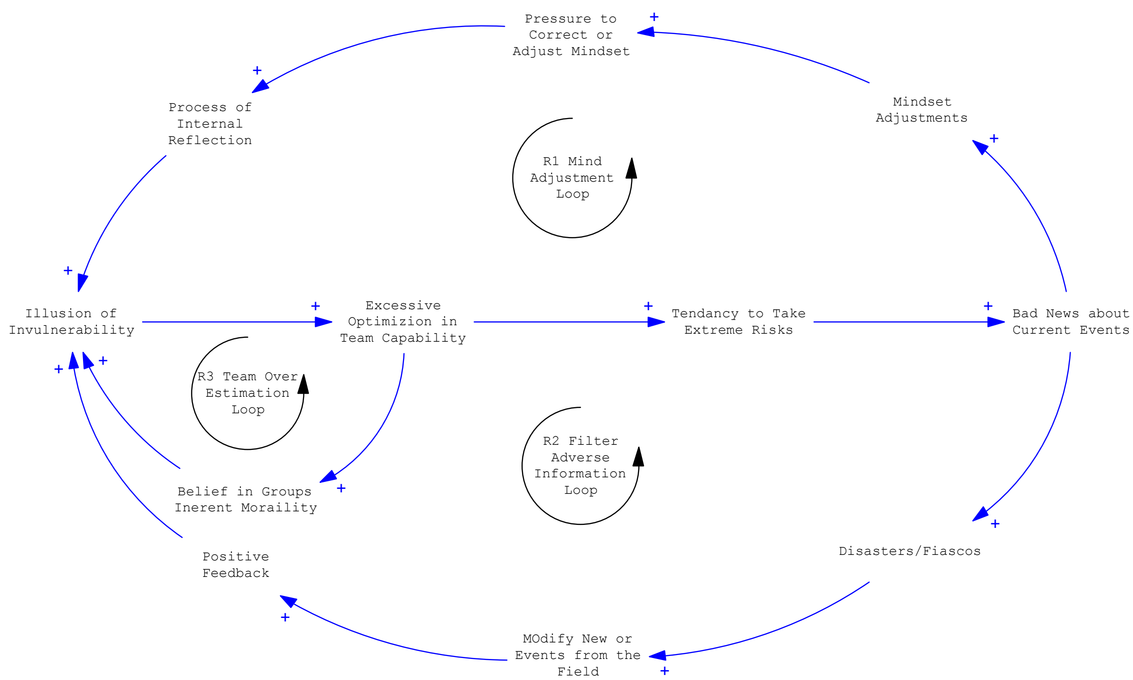


Figure 13.2: Notional Causal Loop Diagram. The CLD is distinguished by the causal links with polarity markings (+/-), labeled Reenforcing (R) or Balancing (B) loops of interest (R1, R2, ...) and rich feedback structure.

We represent a CLD formally as an Attributed Directed Graph (ADG) (subsection 13.2.4), where the variables are vertices and links are directed edges. In the case of CLD, the edge attributes certainly include the link polarity. We also use various augmented CLD representations, in which case the attribute might include an indication of link delay, magnitude of effect, and other link or vertex-specific attributes.

The semantics of the CLD are quite simple: An arrow from variable A to variable B with a polarity sign "+" or "-" indicates that, if the "value" of variable A is incrementally increased, the "value" of B will incrementally either increase ("+") or decrease ("-"), ceteris paribus. "Increased" generally means "towards $+\infty$", not "away from zero". A loop is defined as (B)alancing if it has an odd number of "-" links, because ultimately a disturbing force will tend to be counteracted, else its (R)eenforcing, and will tend to amplify a disturbing force. (See Notional CLD (Figure 13.2))

From a semantic standpoint, each variable in the diagram denotes an attribute or condition that can vary in magnitude (more on this restriction to scalar values later), and each link's polarity determines whether an incremental increase (or decrease) in one variable leads to a corresponding increase (or decrease) in another. Reinforcing loops (Rn) encapsulate processes that accelerate growth or decline by sending changes back through the system in a manner that compounds their initial effect. In contrast, balancing loops (Bn) are characterized by self-correcting tendencies, whereby shifts in one direction eventually produce counteractions that moderate or reverse the original trend. Delays add further complexity by introducing lags between cause and effect, an essential consideration for accurately interpreting feedback phenomena and anticipating system behavior.

Causal loop diagrams are limited because variables without clear scalar algebraic meaning, such as those that aggregate their input, make polarity hard to define, which can blur the distinction between accumulation and direct causation and lead to misinterpretation. For example, while water flow rate into a bucket from a faucet might go up or down, the level of the water in the bucket will never go down (Richardson, 1986b). This is the origin of the "bathtub paradox" (Sweeney & Sterman, 2000).

(Kim, 1995; Kim & Senge, 1994) has good guidelines for manual CLD construction as well as a number of behavioral archetypes. Kim has written a number of useful books on Systems Thinking and use of the CLD.

13.2.4 Attributed Directed Graph (ADG)

In our Pipeline Algebra (PA) Formalism, every model is carried in an *Attributed Directed Graph* (ADG, Hungarian prefix G). An ADG is a directed graph whose vertices, edges, and whole-graph object each possess a statically declared *attribute set*. Attribute types (real, string, Boolean, enumerated, or user-defined domain classes) are registered in the same type system that governs pipeline morphisms; their presence and type correctness can then be mechanically verified. The attribute schema is fixed by the formalism, not supplied ad hoc.

This design permits a single ADG instance to represent multiple modeling layers. For a Causal Loop Diagram (subsection 13.2.3) the attributes record variable names and link polarities; for a System-Dynamics model additional attributes hold equations, units, and parameter values. Because every attribute is formally specified, downstream operators, such as CLD→SD transformers or provenance checkers, can rely on guaranteed field semantics rather than ad-hoc tag parsing. The ADG serves chiefly as an in-memory representation, but it can be losslessly exported to e.g. GraphML (Brandes et al., 2013) should persistent storage or interchange is required.

The ADG is described formally and in the vernacular as shown below:

Let $\mathcal{N}$ be a set of named attributes,

$$\mathcal{P}(\mathcal{N}) = \{S \mid S \subseteq \mathcal{N}\}$$

$\mathcal{V}$ be the set of vertices,

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ be the set of directed edges.

An ADG is a 5-tuple $\mathcal{G} = \big(\mathcal{V}, \mathcal{E}, \phi_V, \phi_E, \phi_G\big)$,

$$\phi_V : \mathcal{V} \to \mathcal{P}(\mathcal{N}),$$

$$\phi_E : \mathcal{E} \to \mathcal{P}(\mathcal{N}),$$

$$\phi_G : \{\bullet\} \to \mathcal{P}(\mathcal{N}), \tag{13.3}$$

where

ϕ_V maps each vertex $v \in \mathcal{V}$

to a subset of named attributes in $\mathcal{N}$,

ϕ_E maps each edge $e \in \mathcal{E}$

to a subset of named attributes in $\mathcal{N}$,

ϕ_G assigns a set of global attributes

to the graph as a whole (indicated by $\{\bullet\}$).

13.2.5 CLD variable naming guidelines

1. **Unambiguous variable phrasing.** Name each variable so that an increase (↑) or decrease (↓) is clear. E.g. *Inventory Level* rather than *Inventory* (Kim, 1995) (Guideline table).

2. **Polarity on every link.** Mark each causal arrow + or − and confirm with the "if … then …" test (J. Sterman, 2000) ch 5, CLD link polarity determination rule (subsection 13.2.6)

3. **Close the feedback loops.** Convert open chains into closed loops to give the diagram

dynamic meaning (Richardson & Pugh, 1981).

4. **Show time delays.** Denote delayed effects explicitly so slow feedback is not mistaken for instantaneous response (J. W. Forrester, 1961; Lane, 2008).

5. **Separate stocks from flows.** Depict accumulations and rates as distinct nodes; conflating them yields the "bathtub" reasoning errors documented by (Sweeney & Sterman, 2000). See also (Kim, 1995).

6. **Avoid "starburst" nodes.** More than about four incoming or outgoing arrows often signals an aggregate that should be decomposed.

7. **Flag exogenous drivers.** Identify variables outside the boundary so the endogenous structure is unambiguous.

8. **Label loops R or B and verify sign counts.** Count negative links: odd $\Rightarrow$ balancing, even $\Rightarrow$ reinforcing.

9. **Check sign consistency around each loop.** A mis-signed link flips an R loop to B or vice-versa (M. R. Goodman, 1974).

10. **Provide a causal narrative.** A short prose explanation surfaces missing links or variables before simulation (Vennix et al., 1996).

11. **Use neutral, quantity-like names.** Avoid evaluative labels that obscure polarity (e.g. *Job Stress*, not *Stressed Employees*) (Kim, 1995).

12. **No bidirectional arrows.** If causation runs both ways, insert an intermediate variable rather than two opposing arrows (Kim, 1995).

13. **Keep each variable conceptually homogeneous.** Do not bundle disparate constructs in one node; split or rename as needed (Richardson, 1997).

14. **Eliminate duplicate or synonym variables.** After drafting, run a *uniqueness check* and merge nodes that are merely different labels for the same concept (Richardson & Pugh, 1981).

We have also noted a few more guidelines are suggested when interacting with AI:

- **Glossary definitions.** Supply a one-sentence definition for every variable because clear, localized semantics help the LLM disambiguate synonyms and anchor embeddings, improving automated vertex alignment and question-answering. .
- **Descriptive multi-word names.** Use self-contained phrases such as *Capacity Factor* rather than abbreviations like *CF* because the richer surface forms boost embedding quality and reduce alias collisions in LLM-based matching.
- **Edge-level provenance.** Store a source sentence, citation, or SME note with every causal link because provenance metadata lets downstream reflection or explanation prompts trace reasoning paths and focus internal or external expert review where confidence is low.
- **Loop narrative tags.** Provide a plain-language, one-line description for each feedback loop because concise narratives help both humans and language models verify sign counts, detect missing links, and generate coherent summaries vennix1996.

13.2.6 CLD link polarity determination rule

Applicability This rule determines the polarity (+ or −) of a direct causal link from a source Variable X to a destination Variable Y ($X \rightarrow Y$) in a Causal Loop Diagram (CLD).

Principle (Ceteris Paribus) To apply the test, consider only the direct influence of X on Y. Mentally change Variable X and determine how Variable Y would respond *directly* to that change, assuming *all other variables in the system remain momentarily constant* (*ceteris paribus*).

Test and Assignment 1. Consider a hypothetical **increase** in Variable X.

- If this increase in X causes Variable Y to **increase**, the link $X \to Y$ is **Positive (+)**.
- If this increase in X causes Variable Y to **decrease**, the link $X \to Y$ is **Negative (–)**.

2. (As a check, the opposite change in X must yield a consistent polarity): Consider a hypothetical **decrease** in Variable X.

- If this decrease in X causes Variable Y to **decrease**, the link $X \to Y$ is **Positive (+)**.
- If this decrease in X causes Variable Y to **increase**, the link $X \to Y$ is **Negative (–)**.

Conclusion The link polarity is Positive (+) if X and Y change in the *same* direction, and Negative (–) if X and Y change in *opposite* directions, based *only* on the direct, isolated impact of X on Y.

13.2.7 Pipeline Specification of work reported herein

The PA morph that computed the results we report upon here is shown below.

HITL interactions are encoded in the same manner as TALM chat interactions: A function Doprompt() is given a model (e.g. "chatgpt -o3" or "sme") to which the prompt, as a string, is sent and from which the response, as a string, is received. The prompt itself is typically a template (Templates (subsection 13.2.2)) but may be a simple string literal.

A few other operators (e.g. ToPDF() may be HITL as well. The PA specification stands, no matter the method of implementation.

$$
\begin{aligned}
&(\\
&\quad \circ\, \text{ExportToVensim}(G \mapsto KeyG_{aicld1}, F \mapsto \text{"/path/to/aicld1.mdl"})\\
&\quad \circ\, \text{CloseSession}()\\
&\quad\quad \circ\, \text{Doprompt}(\mathit{SID} \mapsto \text{Key}(\mathit{SID}_{sme}), F \mapsto \text{"/path/to/aicld1.mdl"})\\
&\quad \circ\, \text{OpenSession}()\\
&\quad \circ\, \text{ExportToVensim}(G \mapsto G_{aicld1}, F \mapsto \text{"/path/to/aicld1.mdl"})\\
&\quad \circ\, \text{CloseSession}(\mathit{SID} \mapsto \mathit{SID}_{mid1})\\
&\quad\quad \text{... SIDmid1 AI Make CLD from Sabstract, Sproblemstmt arrow Gaicld1}\\
&\quad\quad \circ\, \text{Comap}(\\
&\quad\quad\quad \text{Scrape}(\text{"abstract"}, F \mapsto Key(F_{article})) \to S_{abstract},\\
&\quad\quad\quad \text{SIDmid1 AI extract problem statement from file to Sproblemstmt}\\
&\quad\quad)\\
&\quad \circ\, \text{OpenSession}(\mathit{MID} \mapsto \text{Key}(S_{mid1})) \to \mathit{SID}_{mid1}\\
&\quad \circ\, \text{Comap}(\\
&\quad\quad \text{Set}(F_{article} \mapsto \text{"/path/to/smearticle.pdf"})\\
&\quad\quad \text{Set}(F_{smecld} \mapsto \text{"/path/to/smecld.mdl"})\\
&\quad\quad \text{Set}(S_{mid1} \mapsto \text{"chatgpt-o1"})\\
&\quad\quad \text{Set}(S_{mid2} \mapsto \text{"chatgpt-o3"})\\
&\quad\quad \text{Set}(S_{sme} \mapsto \text{"human-sme"})\\
&\quad\quad \text{Set}(S_{author} \mapsto \text{"human-author"})\\
&\quad\quad \text{Set}(S_{jsoncldschema} \mapsto \text{"}\ldots\text{"})\\
&\quad)\\
&)(\{\})
\end{aligned}
\tag{13.4}
$$

Creates the AI CLD given the reference article abstract and ai-extracted problem statement.

$$
\begin{aligned}
&(\\
&\quad \text{Comap}(\\
&\quad\quad \text{ToPDF}(F_{vensim} \mapsto \text{"/path/to/aicld1.mdl"}, F_{PDF} \mapsto \text{"/path/to/aicld1.pdf"}),\\
&\quad\quad \text{ToPDF}(F_{vensim} \mapsto \text{"/path/to/aicld2.mdl"}, F_{PDF} \mapsto \text{"/path/to/aicld2.pdf"})\\
&\quad)\\
&\quad \circ \text{Comap}(\\
&\quad\quad \circ \text{ExportToVensim}(G \mapsto \text{Key}(G_{aicld1}), F \mapsto \text{"/path/to/aicld1.mdl"}),\\
&\quad\quad \circ \text{ExportToVensim}(G \mapsto \text{Key}(G_{aicld2}), F \mapsto \text{"/path/to/aicld2.mdl"})\\
&\quad)\\
&\quad \circ \text{JSONtoADG}(J \mapsto \text{Key}(J_{aicld2})) \to G_{aicld2}\\
&\quad \circ \text{CloseSession}(SID \mapsto SID_{mid2})\\
&\quad\quad \circ \text{Doprompt}(SID \mapsto \text{Key}(SID_{mid2}), \ldots, T \mapsto \ldots \text{export CLD in JSON}) \to J_{aicld2}\\
&\quad\quad \circ \text{Doprompt}(SID \mapsto \text{Key}(SID_{mid2}), \ldots, T \mapsto \ldots \text{for review})\\
&\quad\quad \circ \text{Doprompt}(SID \mapsto \text{Key}(SID_{mid2}), \ldots, T \mapsto \ldots \text{for loops})\\
&\quad\quad \circ \text{Doprompt}(SID \mapsto \text{Key}(SID_{mid2}), \ldots, T \mapsto \ldots \text{for links})\\
&\quad\quad \circ \text{Doprompt}(\\
&\quad\quad\quad SID \mapsto \text{Key}(SID_{mid2}),\\
&\quad\quad\quad args \mapsto \{abstract \mapsto \text{Key}(S_{abstract}), cldguidelines \mapsto \text{Key}(S_{cldguidelines}))\}\\
&\quad\quad\quad T \mapsto \ldots \text{for variables} \ldots\\
&\quad\quad)\\
&\quad \circ \text{OpenSession}(MID \mapsto \text{Key}(S_{mid2})) \to SID_{mid2}\\
&\quad \circ \text{CloseSession}(SID \mapsto \text{Key}(SID_{midauthor})\\
&\quad\quad \circ \text{Doprompt}(SID \mapsto \text{Key}(SID_{midauthor}), \ldots) \to S_{cldguidelines}\\
&\quad \circ \text{OpenSession}(MID \mapsto \text{Key}(S_{midauthor})) \to SID_{midauthor}\\
&) \circ (previouseqn)
\end{aligned}
\tag{13.5}
$$

The above was derived during execution of this work based upon SME feedback, derives an AI-generated given the abstract of the reference article and guidelines on creating a well-formed CLD.

13.3 Results

13.3.1 AI-generated CLD

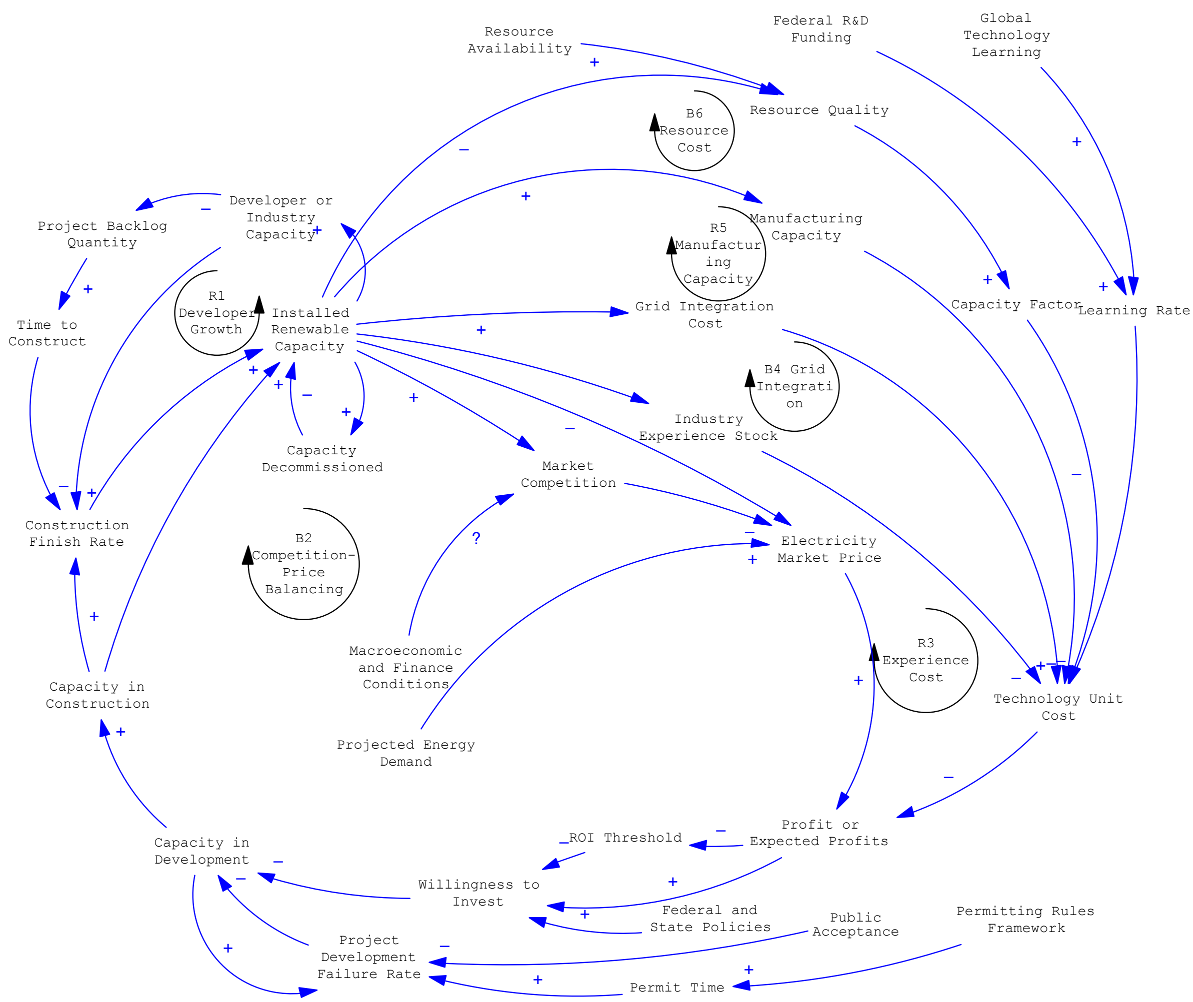


Figure 13.3: AI-generated CLD showing dynamics of deployment of novel energy systems. CLD derived from abstract and AI-derived problem statement of (Lawrence et al., 2025) using PA pipeline shown in the Pipeline Specification of work reported herein (subsection 13.2.7). This CLD is critiqued by the SME in SME Critique of AI-generated CLD (subsection 13.3.2).

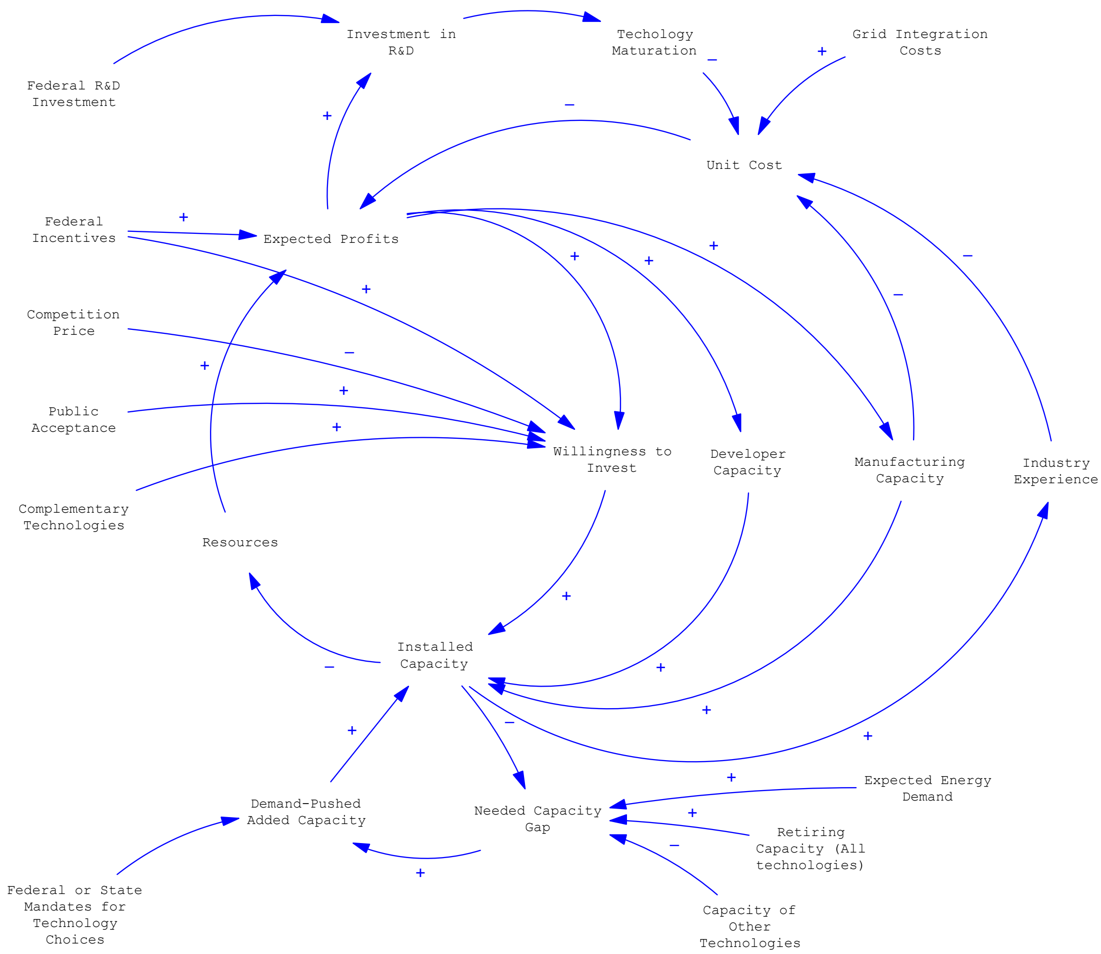


Figure 13.4: SME-generated CLD showing dynamics of deployment of novel energy systems. This is the reference CLD for our work. From Figure 3 of (Lawrence et al., 2025).

13.3.2 SME Critique of AI-generated CLD

The SME evaluated the AI-generated CLD AI-generated CLD (subsection 13.3.1), results shown below. The original SME comment may have been lightly edited. A classification of the errors reported by the SME is shown in Classification of SME-assessed CLD issues Categorization of SME Critiques (subsection 13.3.3).

Quoting the SME:

- **Global Technology Learning** - This is too vague - what does this mean? See the comment for Learning Rate, it applies to this variable as well.
- **Learning Rate** - This is too vague - learning rates are applied to Capital Expenses, O&M expenses, and Capacity Factor. In other words, by improving the technology we make it cheaper to make, cheaper to maintain, and we extract more power (capacity factor).
- **Macroeconomic and Finance Conditions** - This is an odd variables because it can mean anything. The ? instead of +/- shows just that - it is unclear how financial condition will affect the market competition.
- **Projected Energy Demand** - Projected demand does not increase the market price, the market price is the "today" measure, "projected demand" is the possible future measure. The projected demand affects the willingness to invest though.
- **Profit or Expected Profit** - Two profits in the name of this variable. Odd that AI labeled it this way, just a note.
- **ROI Threshold** - Return on Investment threshold is a predetermined desired value, e.g., min 10
- **Federal and State Policies** - This is too vague - policies could push willingness to invest both ways. EPA is a policy and stricter EPA rules will

result in greater portion of declined permit approvals and reduced willingness to invest while tax incentives increase willingness to invest.

- **Permitting Rules Framework** - The permitting rules framework affects the project permit rate which in tern affect the project development failure rate. The permitting rules don't affect permit time, the number of applications does. Lastly, the permit time does not directly affect the project development failure rate. It just takes longer to get from "development" to "construction".
- The SME also noted that significant knowledge is required in order to construct a prompt and to validate the AI response.

13.3.3 Categorization of SME Critiques

We categorized the SME-identified errors/oddities as shown below. The categories were derived based upon the SME analysis, not a priori.

The categorized errors suggest two systemic errors: The AI did not perform well in giving CLD variables well-formed names; and the AI did not perform well in relating link polarities with the semantics implied by the variable name. The pipeline was subsequently improved as shown in the Pipeline Updates in response to SME Critique (subsection 13.3.4).

In response to the categorized SME critique, we added both CLD variable naming guidelines (subsection 13.2.5) and a CLD link polarity determination rule (subsection 13.2.6) to the CLD formulation prompt Pipeline Specification of work reported herein (subsection 13.2.7). The resulting CLD is shown in Phase II CLD Responsive to SME Critique (Figure 13.6). We did not further analyze the results: That is future work.

Table 13.1: Categories for CLD flaws

Code	Category (focus)	Typical symptom
A	Ambiguous / overly broad variable	Name fails to convey a single, well-bounded concept or its sign.
B	Duplicate / syntactically odd label	Internal redundancy or repeated wording in the variable name.
C	Mis-specified causal linkage or polarity	Link direction or sign is inconsistent with domain logic.
D	Redundant / unnecessary element	Adds no new information; duplicates an existing construct.
E	Process-level limitation	Issue lies in the human-AI workflow, not in the diagram content.

Table 13.3: SME-identified issues mapped to category codes

Variable	Code(s)	Rationale
Global Technology Learning	A	Label too vague; concept unclear.
Learning Rate	A	Same vagueness; unspecified which rates apply.
Macroeconomic and Finance Conditions	A,C	Broad label; polarity unknown (“?”).
Projected Energy Demand	C	Incorrect causal path to market price.
Profit or Expected Profit	B	Redundant wording inside the label.
ROI Threshold	D	Adds a second dependency already implied by Expected Profit.
Federal and State Policies	A	Over-general; can push willingness both ways.
Permitting Rules Framework	C,D	Mis-states drivers of permit time and failure; adds extra link.
Prompting / Validation effort	E	Workflow limitation rather than a diagram variable.

13.3.4 Pipeline Updates in response to SME Critique

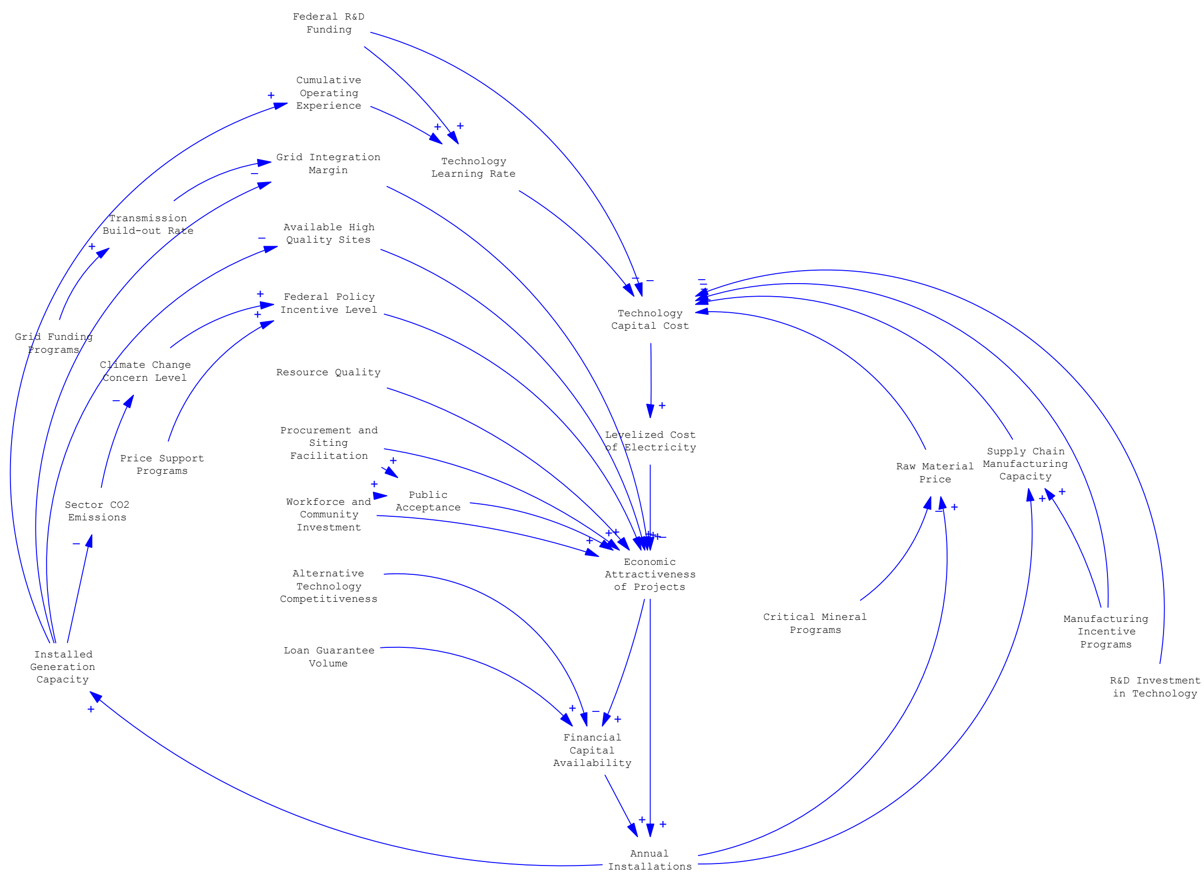


Figure 13.5: AI-generated CLD showing dynamics of deployment of novel energy systems. CLD derived in response to SME Critique of AI-generated CLD (subsection 13.3.2) from abstract of (Lawrence et al., 2025) and CLD guidelines in CLD variable naming guidelines (subsection 13.2.5) using PA pipeline shown Pipeline Specification of work reported herein (subsection 13.2.7)

Figure 13.6

The SME determined that the AI did not perform well in giving CLD variables well-formed names; and the AI did not perform well in relating link polarities with the semantics implied by the variable name. In response to the categorized SME critique, we added both CLD variable naming guidelines (subsection 13.2.5) and a CLD link polarity determination rule (subsection 13.2.6) to the CLD formulation prompt. The pipeline specification Pipeline Specification of work reported herein (subsection 13.2.7) was extended to include the second round of prompting with the improved prompt, so that both the phase I and phase II CLD's can be recreated from the pipeline. The resulting CLD, not further analyzed in this article, shown in Phase II CLD Responsive to SME Critique (Figure 13.6).

13.3.5 AI-generated standard business loops

Table 13.4: Feedback-loop catalog with sign verification. The odd–even negative-link rule is used:

THIS TABLE IS AI-GENERATED			
Loop ID	**Type**	**Core causal path**	**Short narrative**
R1 Learning Curve	R	Annual Installations → Cumulative Operating Experience + → Technology Learning Rate − → Technology Capital Cost + → LCOE − → Economic Attractiveness + → Annual Installations	Experience lowers cost, cost boosts demand, demand adds more experience.
R2 Finance Magnet	R	Economic Attractiveness + → Financial Capital Availability + → Annual Installations → (lower costs & higher returns feed back to Attractiveness)	High returns draw bigger capital pools that fund still more projects.
R3 Supply-Chain Scale-Up	R	Annual Installations + → Supply-Chain Capacity − → Technology Capital Cost + → LCOE − → Economic Attractiveness + → Annual Installations	Factory scale spreads overhead and lowers cost, fueling demand that keeps factories full.
R4 Policy Flywheel	R	Climate Concern + → Policy Incentive Level + → Economic Attractiveness + → Annual Installations − → Sector CO_2 Emissions − → Climate Concern	Public worry drives subsidies; rapid build cuts emissions, sustaining concern long enough for deep penetration.
R5 Price-Support	R	Price Support Programs + → Policy Incentive Level + → Economic Attractiveness + → Annual Installations → (success sustains Price Support)	Fiscal credits directly raise ROI and create political momentum to preserve the credits.

Continued on next page

Table 13.4 – continued

Loop ID	Type	Core causal path	Short narrative
R6 Manufacturing Incentive	R	Manufacturing Incentives + → Supply-Chain Capacity − → Technology Capital Cost + → Economic Attractiveness + → Annual Installations → (utilization validates further incentives)	Production rebates seed domestic plants; lower costs boost volumes that harvest more rebates.
R7 Infrastructure Expansion	R	Grid Funding + → Transmission Build-out Rate + → Grid Margin + → Economic Attractiveness + → Annual Installations → (stakeholders lobby for more Grid Funding)	Public infrastructure spend lifts capacity limits, enabling growth that justifies further spend.
R8 Finance Catalyzer	R	Loan Guarantees + → Financial Capital Availability + → Annual Installations → (portfolio success underwrites new guarantees)	Risk transfer lowers borrowing cost; successful builds validate and expand guarantee programs.
B1 Market Saturation	B	Annual Installations + → Installed Capacity − → Available High-Quality Sites + → Economic Attractiveness − → Annual Installations	Finite prime sites mean economics deteriorate as easy opportunities are used up.
B2 Capacity Constraint	B	Annual Installations + → Installed Capacity − → Grid Margin + → Economic Attractiveness − → Annual Installations	Shared-asset congestion or curtailment risk slows further builds until capacity is expanded.
B3 Complacency	B	Annual Installations − → Sector CO_2 Emissions − → Climate Concern − → Policy Incentive Level − → Economic Attractiveness − → Annual Installations	Early success reduces perceived urgency, eroding policy support and cooling growth.

Continued on next page

Table 13.4 – continued

Loop ID	Type	Core causal path	Short narrative
B5 Commodity Squeeze	B	Annual Installations + → Raw Material Prices + → Technology Capital Cost + → LCOE – → Economic Attractiveness – → Annual Installations	Rapid scale-up tightens commodity supply, raising costs and damping demand unless countered by diversification programs.

13.4 Discussion

The SME Assessment of AI-generated CLD (subsection 13.3.1), especially when distilled into a Categorization of SME Critiques (subsection 13.3.3), strongly suggested two weaknesses in our first experiment: (1) We failed to include rules for a well-formed CLD in our prompt; and (2) by depending on the Extracted Problem Statement, we tacitly used much of the SME labor as input to the algorithms which we claim reduces the need for that labor.

We substantiated these implementation deficiencies rather than fundamental weaknesses with a preliminary experiment. Our method was to (1) develop CLD variable naming guidelines (subsection 13.2.5) with an eye towards utility for both human and AI use; (2) add them to the persona prompt; (3) use iterative prompting to increase probability that sufficient TPU's are allocated to our experiment; and (4) provide only the abstract of (Lawrence et al., 2025) for task definition, rather than the full Extracted Problem Statement.

Our preliminary results, after first prompting for AI-generated variable glossary, then for AI-generated links info, then for AI-generated loop info, and only then for the AI-generated CLD, suggest that the improved prompt substantially improved the quality of the resultant Phase II CLD Responsive to SME Critique (Figure 13.6). It is future work to formally analyze the CLD and to further refine the processes.

In addition, we prompted for AI-generated standard business loops (subsection 13.3.5) embedded in the CLD, which, if future analysis indicates the loops are well formed and significant, would provide further support for our claim that properly orchestrated AI can significantly aid in ST and SD, as well as specifically highlight the capability of the TALM to comprehend the structure and behavior of a CLD. The results strongly suggest this is worthy of further exploration, and suggestive of the potential of explicitly assembling CLD's (or perhaps even SD models) from well-known molecules that embody the standard business loop behaviors on the one hand, or mapping observed behaviors into standard business loops from which the model is inferred on the other hand.

Future Work We believe it would be useful to develop a comprehensive prompt for CLD creation that covers the complete process, perhaps along the lines of (J. Sterman, 2000, ch5) and (Richardson & Pugh, 1981, ch2). This would keep the TALM focused on creating a comprehensive and well-formed CLD. Key points probably include the inputs to the TALM including, reference modes of interest, SOI boundary, and time horizon; and features of a well-formed CLD.

Conduct further case studies (healthcare, supply chain, ecology) to test domain versatility.

Develop comparative and criterion-based methods of CLD evaluation, perhaps based on reconciliation of variable names then using a confusion matrix to compare variables, links, and loops (after lexicographic rotation to normalize) (Kohavi, 1998).

Exercise meta-algorithmic optimization of PA, with particular attention paid to LLM-based approaches. We speculate that the linear compositional structure of PA will enhance the ability of the LLM to comprehend and manipulate the PA.

14 Missing Loops

14.1 Introduction

We define a missing loop as any real-world feedback loop that is absent from a CLD or SD model and, as a consequence, diminishes the fidelity of the model's projections; such omissions occur in two forms: Type I (directional) missing loops introduce systematic one-sided bias, giving forecast errors a non-zero mean, for example, the model persistently over- or under-predicts a variable, whereas Type II (variance or structural) missing loops leave the mean error near zero but inflate error variance, induce autocorrelation, mistime peaks, or create bias that appears only under certain states, so that although the average error cancels out, confidence in timing and variability is misplaced.

"Missing feedback is one of the most common causes of system malfunction" (D. Meadows, 1999), and finding them is a standard element of model validation (J. Sterman, 2000; J. D. Sterman, 1994). Indeed, adding policy resistance loops missing from a simple epidemic model (Rahmandad et al., 2022) reduced its forecast bias to approach state of the art models. Missing stabilization loops leading to Groupthink P43 Janis for IS 2025 Revised (chapter 12) (Kim, 1992, §Escalation)could have caused a nuclear war.

Loop omissions are not necessarily accidental. Indeed, they are often engineered into public polices: Enable Regulatory Capture (Stigler, 1971); provide future opportunities for blame avoidance (Weaver, 1986), or sabotage policy performance for electoral advantage (Hirsch & Kastellec, 2022).

Recent Examples:

California electricity deregulation (2000-01): The market design rules capped retail prices while allowing wholesale bids to clear every 15 minutes, but they omitted a stabilizing loop that would have forced generators to keep reserve margins in line with real demand.

Enron traders gamed this gap, parking megawatts out of state to create artificial shortages, then reselling at scarcity prices: extracting an estimated $1.6 billion and triggering rolling blackouts. (Nix et al., 2021)

Voluntary REDD+ carbon-offset programs: Most forestry offset protocols treat baseline deforestation as an exogenous trend and lack a corrective loop that updates baselines with actual land-use change. A 2023 Science audit of 18 high-profile projects found that 94 % of the credits sold were "hot air," allowing firms to claim carbon neutrality (and, more to the point, to collect premium prices) without real climate benefit. The authors call the missing-baseline loop a critical vulnerability that "invites profiteering." (West et al., 2023).

Blockchain and generative-AI policies often omit crucial feedback loops, externalizing environmental and social costs while concentrating profits(de Vries-Gao, n.d.).

As systems engineers, our responsibility is to construct models that faithfully represent the system of interest (SOI) in both structure and behavior. A model that omits critical feedback loops falls short, so the literature contains a sustained effort to detect such gaps. Traditionally, missing loops are uncovered through manual scrutiny: peer review of causal-loop diagrams (CLDs), boundary-adequacy tests, and group-model-building workshops: All an intensive process documented by Sterman (J. Sterman, 2000, ch.3) and formalized in validation studies by Barlas and colleagues (Barlas, 1996). Automated techniques have begun to appear, notably the mining of process-execution logs to derive provisional CLDs (Pourbafrani & Aalst, 2022) and sparse system-identification methods that learn governing equations directly from data (Brunton et al., 2016); however, these approaches still focus on fitting observed trajectories rather than probing structural completeness. We investigate a new direction: deploying a large-language model (LLM) to analyze model topology for vulnerabilities and likely missing feedbacks, capitalizing on the LLM's broad embedded knowledge of human and policy behavior and its agentic ability to consult external sources.

We present a proof of concept that the LLM can detect both Type I and Type II missing loops, given only a CLD to analyze. We analyze a previously published SME-generated CLD

(Lawrence et al., 2025, Figure 3) that we studied in LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra (chapter 13).

We have argued earlier the advantages of using the CLD in interactions with the LLM. We further note this quote as suggestive: "A recent study by two CERT/CC researchers had employed qualitative system dynamics (i.e. causal loop analysis) for controlling vulnerability ... Before embarking on that, the authors had remarked... we are not aware of any work using system dynamics to explicitly study the threat environment or its impact on system operations..."(Andersen et al., n.d.).

We conclude based on a direct read of our results, AI-identified Type I Missing Loops (subsection 14.3.1) and AI-identified Type II Missing Loops (subsection 14.3.2), that the LLM appears competent at identifying missing loops of both types.

Perhaps the most striking result is that every loop we first uncovered tacitly assumed "rational" behavior. Only when the language model was asked point-blank for bad-actor loops did it produce credible candidates (see AI On Bad Actors and Missing Loops (subsection 14.3.3)). In a follow-up exchange the model explained the omission:

> System-dynamics conventions and the evidentiary landscape nudge analysts (and thus the LLM) to concentrate on well-documented, endogenous feedbacks, implicitly treating sabotage, disinformation, or rent-seeking as exogenous noise. That focus, reinforced by rational-actor assumptions, the scope suggested by the original variables, and a normative reluctance to foreground politically charged dynamics, allowed "bad-actor" loops to be overlooked even though they can strongly shape variance-type errors.
>
> — (Dialog with LLM)

It is future work to determine the degree to which this is true and what to do about it as Systems Engineers.

14.2 Method

The CLD is serialized into two prompts: One to elicit Type I missing loops, another to elicit Type II missing loops. The prompts are evaluated in independent anonymous ChatGPT o3 sessions to prevent cross influences.

The format of prompt responses was not constrained, so as to allow the LLM to devote maximum attention to the prompt. The response markup was later converted to LaTeX for publication.

We noted mid-experiment that the LLM was not considering missing loops that remain latent unless activated by bad actors. We performed ad-hoc queries to explore this.

The results have not been evaluated by a subject-matter expert.

14.3 Results

14.3.1 AI-identified Type I Missing Loops

THIS TEXT IS AI-GENERATED

1. Community-Familiarity Reinforcing Loop
(a) Loop name: Community-Familiarity Reinforcing Loop
(b) Missing link & polarity:

$$\text{Installed Capacity} \xrightarrow{+} \text{Public Acceptance}$$

(c) Short causal story: As wind-turbines, solar farms, or reactors become everyday sights, local familiarity raises Public Acceptance. Greater acceptance boosts Willingness to Invest, reinforcing further additions to Installed Capacity.
(d) Bias direction: Downward bias-omitting the positive loop makes the model under-predict capacity growth and investment.
(e) Real-world examples: Surveys show residents living near wind projects rate the technology more favorably than those with no exposure; the same neighborhood "peer effect" appears in rooftop-solar adoption data.

2. Incentive-Sunset Balancing Loop
(a) Loop name: Incentive-Sunset Balancing Loop
(b) Missing link & polarity:

$$\text{Installed Capacity} \xrightarrow{-} \text{Federal Incentives}$$

(c) Short causal story: When a technology reaches scale, legislators phase out tax credits or grants. Shrinking Federal Incentives lowers Expected Profits and, in turn, curbs Willingness to Invest, slowing new builds.

(d) Bias direction: Upward bias-forecasts overstate capacity and profits by assuming incentives persist indefinitely.
(e) Real-world examples: U.S. wind additions dip each time the Production Tax Credit lapses; IRA clean-power credits begin phasing down once sector-wide emissions targets are met.

3. Grid-Congestion Cost Balancing Loop

(a) Loop name: Grid-Congestion Cost Balancing Loop
(b) Missing link & polarity:

$$\text{Installed Capacity} \xrightarrow{+} \text{Grid Integration Costs}$$

(c) Short causal story: Beyond modest penetration, every new gigawatt of variable renewables drives extra transmission, curtailment and balancing costs. Higher Grid Integration Costs raise Unit Cost, squeeze Expected Profits, and dampen Willingness to Invest.
(d) Bias direction: Upward bias on capacity (and downward bias on costs)-the model ignores a key brake on growth.
(e) Real-world examples: CAISO's solar-curtailment bill and the EU's ballooning transmission upgrades illustrate sharply rising integration costs with higher renewable shares.

4. Complementary-Technology Acceleration Loop

(a) Loop name: Complementary-Technology Acceleration Loop (plausible but rarely modeled)
(b) Missing link & polarity:

$$\text{Installed Capacity} \xrightarrow{+} \text{Complementary Technologies}$$

(c) Short causal story: Large-scale deployment of a core technology (e.g. solar PV) spurs investment in complements (batteries, hydrogen off-takers). Those complements reduce variability and costs, further lifting Willingness to Invest in the core technology-a reinforcing spiral.
(d) Bias direction: Downward bias-forecasts understate how quickly investor confidence and capacity can grow once complements catch up.
(e) Real-world examples: California's utility-scale battery boom closely tracks its solar surge; similar dynamics emerge with heat-pumps plus building-retrofits and EVs plus fast-charging infrastructure.

14.3.2 AI-identified Type II Missing Loops

THIS TEXT IS AI-GENERATED

1. **Supply-Chain Bottleneck Balancing Loop (B1)**

Missing causal link	Polarity
Installed Capacity → Supply-Chain Pressure	+
Supply-Chain Pressure → Component Lead-Time	+
Component Lead-Time → Installed Capacity (rate)	−

Causal story. Rapid build-out strains upstream suppliers, lengthening lead-times and tempering near-term installations. Bias direction. Omitting this damping loop understates variance and shifts peak build years earlier. Examples. 2021-22 polysilicon shortage delaying U.S. PV projects; 2024 wind-turbine component shortages in Brazil.

2. **Capital-Cost (Interest-Rate) Balancing Loop (B2)**

Missing causal link	Polarity
Willingness to Invest → Sectoral Demand for Capital	+
Sectoral Demand for Capital → Cost of Capital	+
Cost of Capital → Willingness to Invest	−

Causal story. A rush of projects competes for limited tax-equity and debt; higher capital costs curb new commitments. Bias direction. Model becomes too smooth, ignoring boom-bust swings tied to macro-financial cycles. Examples. 1980s U.S. nuclear cancellations during double-digit interest rates; 2023 residential-solar slowdown after Fed hikes.

3. **Local-Opposition (NIMBY) Balancing Loop (B3)**

Missing causal link	Polarity
Installed Capacity → Community Disamenities	+
Community Disamenities → Public Acceptance	−

Causal story. Visible or noisy assets erode public acceptance, which then lowers willingness to invest, slowing expansion. Bias direction. Forecasts place capacity too quickly and with insufficient variability, missing multi-year permitting pauses. Examples. Cape Wind (MA) and Ocean Wind 1 (NJ) decade-long delays; 2017-19 German onshore-wind contraction due to local bans.

4. **Skilled-Labor Constraint Balancing Loop (B4 - new, plausible)**

Missing causal link	Polarity
Installed Capacity → Demand for Skilled Labor	+
Demand for Skilled Labor → Labor Wage Rate	+
Labor Wage Rate → Unit Cost	+

Causal story. Surging installations outstrip the supply of certified crews; rising wages lift unit cost and curb new builds via lower profits. Bias direction. Year-to-year volatility and schedule slips are under-predicted; mean capacity may stay accurate once labor catches up. Examples. 2024 U.S. offshore-wind delays from crew shortages; 2023 Australian solar-EPC price spike due to limited HV electricians.

14.3.3 AI On Bad Actors and Missing Loops

THIS TEXT IS AI-GENERATED

Ad-hoc prompt: Show four plausible loops that are missing and are latent unless there are bad actors. Examples are rent-seeking and attempts to sabotage energy system growth using viral social media. Do not limit yourself to that class of bad actors though.

1. **Regulatory-Capture Rent-Seeking (B_R1)**

Missing causal link	**Polarity**
Installed Capacity → Threat to Incumbent Profits	+
Threat to Incumbent Profits → Lobbying Intensity	+
Lobbying Intensity → Regulatory Barriers	+
Regulatory Barriers → Willingness to Invest	−

Causal story. Growing capacity threatens incumbent margins; incumbents fund lobbying that raises permitting or interconnection barriers, curbing investment. Bias if omitted. Deployment is forecast too quickly and with too little variance during intense lobbying periods.

2. **Viral-Disinformation Social-License Loop (B_D1)**

Missing causal link	**Polarity**
Installed Capacity (or Policy Push) → Disinformation Funding	+
Disinformation Funding → False-/Negative-Content Volume	+
False-/Negative-Content Volume → Public Acceptance	−

Causal story. Bad actors seed viral narratives ("kills birds", "grid risk") that erode public acceptance, suppressing willingness to invest. Bias if omitted. Model misses abrupt multi-year slowdowns triggered by coordinated misinformation campaigns.

3. **Critical-Mineral Price Manipulation (B_M1)**

Missing causal link	**Polarity**
Installed Capacity → Demand for Critical Minerals	+
Demand for Critical Minerals → Mineral Prices	+
Mineral Prices → Unit Cost	+

Causal story. Speculators or state-owned suppliers hoard lithium, cobalt, or rare-earths; price spikes lift unit cost and slow new builds. Bias if omitted. Cost volatility and timing of build-out waves linked to commodity cycles are understated.

4. **Cyber-Sabotage Grid-Instability Loop (B_C1)**

Missing causal link	Polarity
Installed Capacity (variable renewables) → Attack Surface	+
Attack Surface → Cyber-Attack Frequency	+
Cyber-Attack Frequency → Grid Integration Costs	+
Grid Integration Costs → Unit Cost	+

Causal story. Expanded inverter/SCADA footprint invites cyber attacks; utilities harden the grid, raising integration costs that flow through unit cost and profits to dampen investment. Bias if omitted. Forecasts ignore cost spikes and investment dips that follow major cyber incidents, understating year-to-year volatility.

Why no bad actor responses?

Ad-hoc prompt: Speculate why your response to this prompt originally did not include any "bad actor" missing loop. Response:

#	**Reason category**	**What probably happened**	**How it skews the search for missing loops**
1	Model-building habit: focus on endogenous, well-intentioned actors	Classical SD practice first maps feedbacks inside the system and treats sabotage as an exogenous shock.	Adversarial feedbacks look like noise rather than missing structure and are therefore ignored, even though they can dominate variance once adoption becomes politically salient.
2	Evidence availability & retrieval bias	Abundant peer-reviewed work quantifies grid bottlenecks, NIMBY effects, learning curves, etc.; rigorous studies of disinformation campaigns are rarer and scattered.	The analyst (or language model) defaults to well-documented loops, under-representing low-visibility dynamics that are harder to quantify but still powerful.
3	Implicit "rational actor" assumption	Energy models often assume incumbents respond via prices or transparent lobbying already captured in "competition" variables. Covert interference lies outside that play-book.	Variance mechanisms that are strategic rather than purely economic are omitted, causing overly smooth diffusion curves.

4	Scope inferred from original CLD variables	The diagram contains no stocks like “Information Environment” or “Lobbying Effort.” When hunting for omissions, we tend to recombine existing variables.	Loops requiring entirely new variables (e.g. “Co-ordinated Disinformation”) are subconsciously penalized despite being allowed by the prompt.
5	Normative blind spot	Clean-energy work is often mission-driven; analysts may shy away from highlighting tactics that could further undermine the transition or appear politically charged.	Bias toward technical or socio-economic loops and away from politically contentious dynamics that generate high uncertainty.

Bottom line: Analysts (and LLMs trained on their corpus) gravitate toward well-behaved, internally motivated feedbacks that are straightforward to parameterize. Adversarial or malicious dynamics are relegated to “shocks,” even though-from a variance-error perspective-they deserve representation as proper balancing loops once the technology becomes a threat.

14.4 Discussion

Discussion

We have shown empirical evidence that the LLM is a powerful aid in surfacing missing loops in a published CLD. We noted an interesting gap: The LLM generally discounts "bad actor" vulnerabilities such as rent seeking, sabotage, or disinformation attacks. It appears biased towards a "rational actor" model of the world. When prompted to explain this apparent bias, the LLM noted among other things a "normative reluctance to foreground politically charged dynamics".

We believe our work shows the LLM is a valuable tool in policy analysis, though some prompt engineering will be required in order to cause stronger consideration of "bad actor" vulnerabilities.

There are several limitations that must be considered: Hallucination, of course, though our method of falsifiable prompting can mitigate this to some extent; The method depends critically on long names and descriptions of the CLD so that the behaviors are well represented in the LLM latent space; the results are probably sensitive to prompts (the rational actor bias being an example of this); and finally, one must consider that the bad actors are themselves using similar approaches to find vulnerabilities in the SOI.

We used a very simple prompting sequence. It is future work to refine the prompting strategy, in particular we expect that some prompting to condition the posterior with considerations of usual and novel vulnerabilities and missing loops would increase the power of the method.

More general future work is to invert the control pattern: At present we execute the PA pipeline locally, and treat the LLM as one of any number of tools available under terminal operators: e.g. there's an operator for prompting, an operator for integrating a model, etc. It appears that the commercial products e.g. ChatGPT have "tools" APIs and agentic frameworks sufficiently powerful that it is time to center control on the agentic LLM, treating my terminal operators as tools within that context. This would simplify application of the LLM CoT and agentic capabilities to developing and executing a pipeline, which would probably make our work more approachable (it's just prompting, not specialized workbench) and more capable. If I were starting my PhD today I would propose this as the technical foundation of my research.

15 Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI

15.1 Summary

Causal–loop diagrams (CLDs) (M. Goodman, 1973) capture qualitative feedback structure, yet turning them into calibrated system–dynamics (SD) models remains (Barbrook-Johnson & Penn, 2022; Richardson, 1997; J. Sterman, 2000) labor-intensive and error prone. We introduce *falsifiable prompting*: a human-in-the-loop protocol that, in this case, forces a large-language model (OpenAI (OpenAi, 2023) **o3**) to generate SD equations whose correctness can be *independently falsified*, thereby exposing and eliminating hallucinations.

The loop is implemented in Pipeline Algebra (chapter 2) and executes as follows: (1) convert a 31-variable CLD of U.S. renewable-capacity diffusion into executable stock–flow code; (2) run the simulation; (3) pass automated validators for dimensional consistency and polarity (Barlas, 1996) preservation; (4) reproduce an eight-point ref mode calibration trajectory; and (5) repeat, with optional human veto, until all tests succeed. Using only the S-curve published by (Lawrence et al., 2025) Figure 12 (0 $\rightarrow$ 335 GW in 40 yr) as ground truth, the loop converges in four AI-driven iterations to a parsimonious one-stock model that aggregates twelve reinforcing loops into a single gain (β). The calibrated model matches the benchmark with Theil's $U = 0.019$; an independent Wolfram Mathematica rerun confirm robustness. Despite its simplicity, the model reproduces the macro-dynamics of Lawrence's five-stock, data-rich formulation, demonstrating that the aggregated β suffices when detailed exogenous streams are unavailable. It does not demonstrate the oscillatory behavior over

the baseline S curve that can be seen in Lawrence figure 12.

Of particular interest was the unprompted choice by the AI to simplify the model to a single stock based on the observation that there was not sufficient ref mode data provided to calibrate a more extensive model.

The pipeline cut model-development from days to hours (in our judgement. Formal time studies have not been conducted), yet preserved scientific rigor through continuous falsification tests.

Portions of this abstract were drafted with AI assistance (OpenAI o3) and curated by the authors.

15.2 Introduction

Causal–loop diagrams (CLDs) have long served as the lingua franca of *systems thinking* (ST), offering a concise way to record how feedback processes drive behavior over time (M. Goodman, 1973). By themselves, however, CLDs are qualitative; only when they are translated into executable, stock–flow formulations can the underlying hypotheses be tested numerically and subjected to sensitivity analysis (J. Sterman, 2000). In other words, the bridge from CLD to a calibrated system–dynamics (SD) model fuses conceptual insight with quantitative rigor, uniting two communities that often work in parallel rather than together.

The payoff is large for both camps. SD modelers *must* pass through the CLD stage to launch any serious simulation effort, while ST practitioners gain three clear benefits from a working SD analog: (i) intuitive loop reasoning is confirmed or refuted by time-series behavior; (ii) idiosyncratic, non-archetypal structures can be explored without forcing them into textbook molds; and (iii) multiple archetypes can be composed and stress-tested in a single coherent run. Unfortunately, the translation remains notoriously painful. It demands fluency in CLD syntax, SD best practice, unit algebra, polarity logic, and statistical calibration: all skills that are orthogonal to most subject-matter expertise and therefore form a high cost-of-entry bottleneck (Barbrook-Johnson & Penn, 2022; Richardson, 1997).

Large-language models (LLMs) offer a tempting shortcut. Recent work shows they can perform analogical structure mapping, leverage latent domain knowledge, and synthesize executable code across disciplines (Wei et al., 2022). Yet unfiltered LLM output is plagued by *hallucinations*(Ji et al., 2023): seemingly plausible but syntactically or semantically invalid statements. For SD this risk is acute: a single unit inconsistency or polarity error can invalidate an entire model.

We therefore propose *falsifiable prompting*, a zero-trust, five-step protocol that forces the LLM to earn credibility through continuous refutation tests. Each AI-generated model must (1) compile and run, (2) pass automated checks for dimensional consistency and loop polarity (Barlas, 1996), (3) reproduce an agreed calibration data set, and (4) restart if any test fails. Only when all hurdles are cleared does human review become optional. The protocol may be encoded as typed morphisms in *Pipeline Algebra* (PA), providing an audit-trail and making future plug-ins, such as edge-AI generators or hybrid motif libraries, straightforward.

To demonstrate the idea we task OpenAI's o3 model with converting a 31-variable CLD of U.S. renewable-capacity diffusion (Lawrence et al., 2025) into an SD model using nothing but an eight-point S-curve (0 335 GW in 40 years) as ground truth. Within four prompt–test cycles the LLM produces a parsimonious one-stock model whose single aggregated gain (β) subsumes twelve feedback loops. Independent reruns reproduce the benchmark with RMSE$\approx$ 8 GW and Theil's $U \approx 0.018$, while unit and polarity tests pass cleanly. Development time falls substantially relative to a manual baseline, yet no scientific safeguards are compromised.

One residual risk is *structural overfit*: an LLM could fabricate opaque algebra that hugs the calibration curve yet embodies no SOI-related feedback logic. (Richardson, 2011) Our prototype focuses on demonstrating the automated gate; systematic human vetting of structural plausibility was *not* performed here beyond enforcing maintenance of the CLD causal structure, but we emphasize that in production use the workflow should conclude with an expert review step to confirm that the machine-generated structure aligns with accepted causal reasoning before it is used for policy insight or forecasting.

The remainder of the paper proceeds as follows. Section Methods (section 15.3) formalizes the falsifiable-prompting loop in PA. Section Results (section 15.4) presents calibration and validation results, including robustness checks. Section Discussion (section 15.5) weighs limitations, such as structural overfit risks, and outlines future work on AI-friendly motif libraries, and concludes that AI, when governed by falsifiable prompting, can reliably automate the CLD SD pipeline without sacrificing rigor. Section Appendicies (section 15.7) includes full prompt/response details, and a Toulmin(Toulmin, 2003) analysis of the reference article (Lawrence et al., 2025) that was used to refine the prompt (Prompt Customization (subsection 15.3.2))

15.2.1 Argument Overview

Argument Overview

1. **From qualitative insight to quantitative test.** Causal–loop diagrams (CLDs) encode hypotheses about feedback (M. Goodman, 1973) structure, whereas system–dynamics (SD) simulations expose those (J. Sterman, 2000) hypotheses to numerical scrutiny. Translating CLDs into SD models therefore welds the conceptual practice of *systems thinking* (ST) to the formal discipline of SD.

2. **Why CLD $\rightarrow$ SD matters for both camps.** *SD* requires it to initiate any stock–flow model. *ST* gains three benefits: (i) confirmation or refutation of intuitive expectations, (ii) coverage of non–archetypal structures, and (iii) composition of multiple archetypes into one coherent simulation.

3. **The current pain point.** Manual CLD $\rightarrow$ SD translation demands deep expertise in ST, CLD syntax, SD practice, and validation: Skills orthogonal to most subject–matter domains and therefore a severe bottleneck.

4. **AI as an enabler.** Modern large–language models exhibit (Wei et al., 2022) analogical structure mapping, latent systems knowledge, and emergent systems intelligence, suggesting an automated pathway from CLD to executable SD.

5. **Hallucination risk and our solution.** LLMs can fabricate plausible but false structures. We introduce *falsifiable prompting*, a zero–trust protocol in which every AI response must pass independent checks (in this particular case those are executability, unit consistency, polarity preservation, and (Richardson & Pugh, 1981; J. Sterman, 2000) empirical calibration); any failure falsifies the output and restarts the loop.

6. **Evidence presented in this study.**

   (i) Input: the published CLD by (Lawrence et al., 2025).

   (ii) AI, without seeing data, predicted diffusion–of–innovation S-curve behavior.

   (iii) AI received eight calibration points and generated an SD model, auto-calibrating it.

   (iv) Independent code reran the model to attempt falsification: did not falsify.

   The calibrated model reproduces the predicted S-curve, confirming both the AI's qualitative forecast and its quantitative SD translation.

These steps support our central claim: **AI, guided by falsifiable prompting, can automate the CLD → SD pipeline while preserving scientific rigor.**

TBS

Figure 15.1: TBS: Graphical Abstract Here

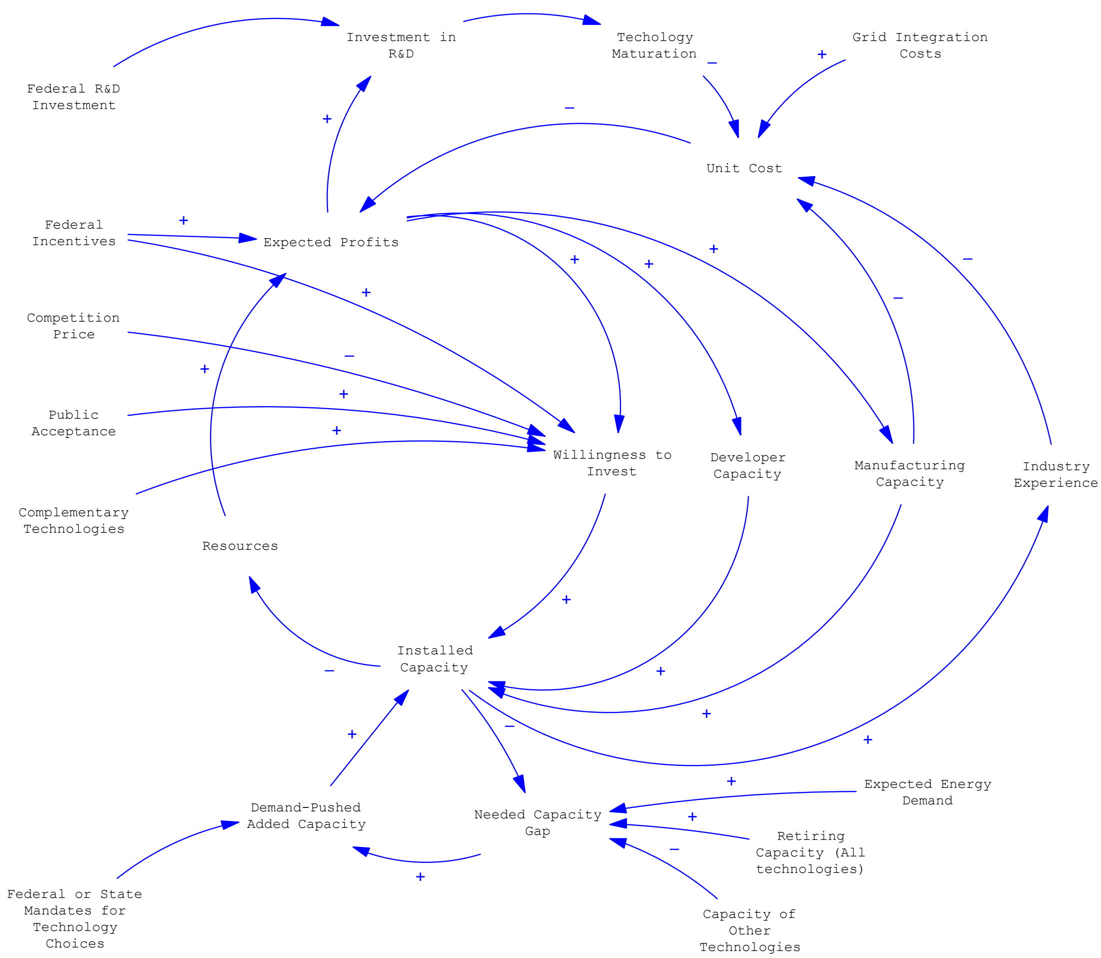


Figure 15.2: SME-generated CLD showing dynamics of deployment of novel energy systems. This is the reference CLD for our work. From Figure 3 of (Lawrence et al., 2025).

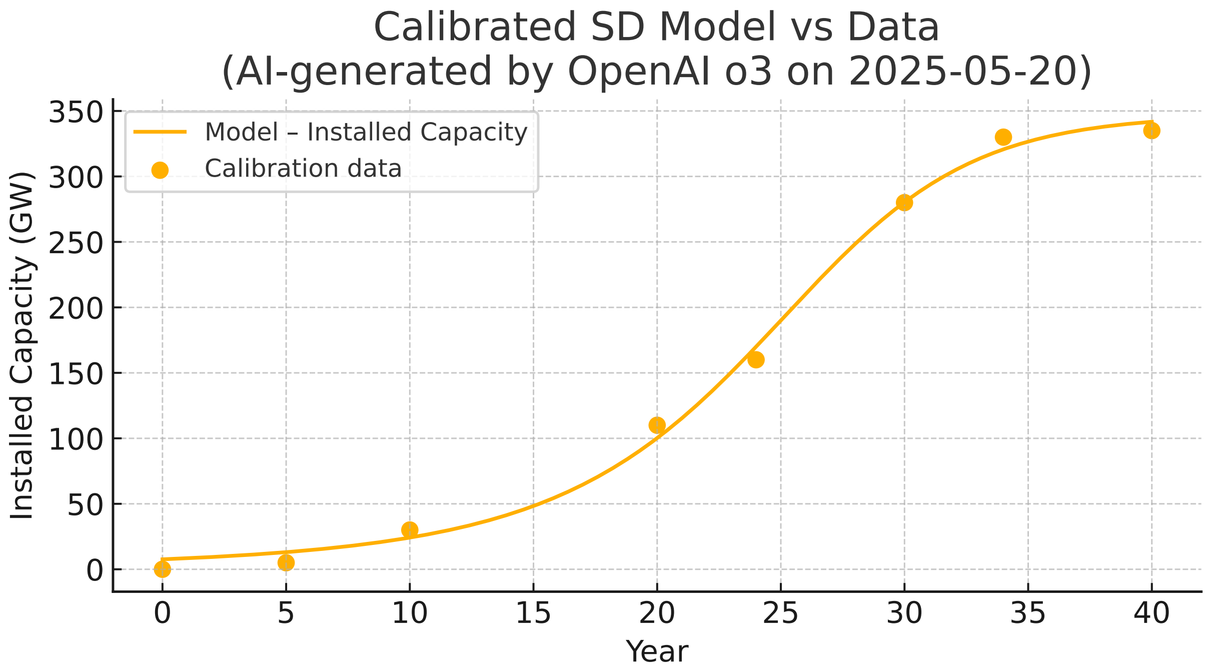


Figure 15.3: AI-generated calibration results for SD model. The calibration claim can be falsified by executing the model outside of the AI (e.g. in Wolfram Mathematica). The model was in fact independently encoded and solved, and compares extremely well with and so collaborates the AI integration. Inspection of the AI-generated python SD model indicates it used Euler integration and python constructs to emit the plot.

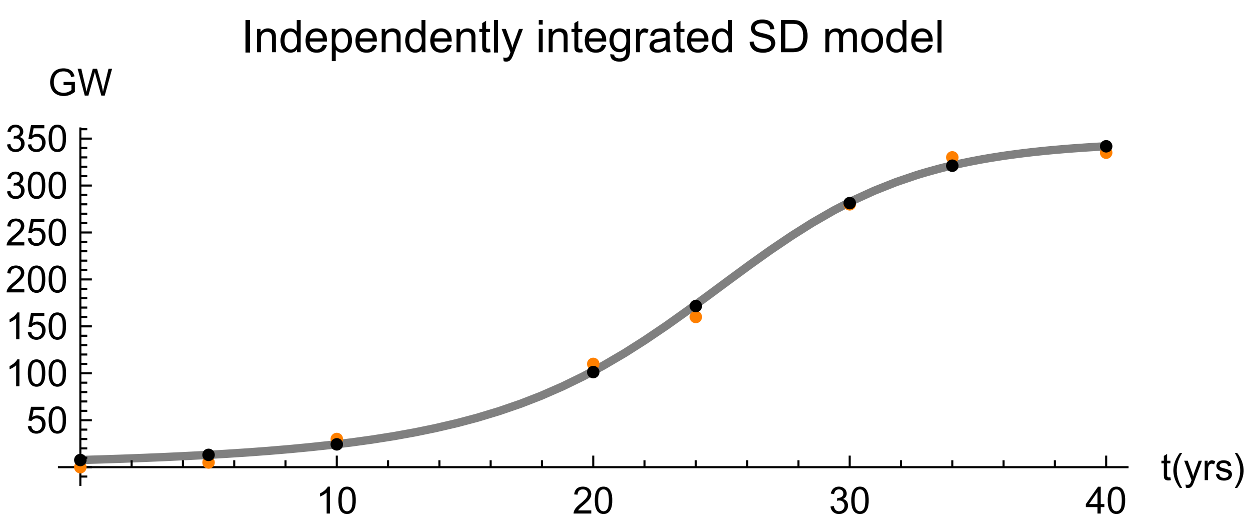


Figure 15.4: Numerical integration of SD model equations derived by AI when prompted to translate the CLD shown in Figure 3 of (Lawrence et al., 2025) into a calibrated SD model. The line is the numerical integration of *Installed Capacity*, the points are calibration points scaled from the published paper figure 12. The Theil *U1* of 0.019 compares well with the U1 of 0.018 estimated by the AI, and both are suggestive of a sound forecast relative to actuals, indicating the AI SD model is collaborated by falsifiable test

15.3 Methods

15.3.1 Where AI results go in an IMRaD article

After careful consideration we placed the large-language–model's on-the-fly aggregation decision in the **Methods** section rather than in Results. Although the aggregation emerged during the modeling run and not from our prompt, it constitutes a procedural step, analogous to choosing a variable-selection algorithm, that future researchers must replicate to reproduce our findings.

Automated aggregation of twelve reinforcing variables into a single gain parameter (β) is, in essence, an instance of *automated variable selection.* Item 10b of the original TRIPOD checklist instructs authors to *"specify type of model, all model-building procedures (including any predictor selection), and method for internal validation"* (Collins et al., n.d.). Because predictor (variable) selection is explicitly named as methodological content, we record the LLM's aggregation logic (triggered during falsifiable prompting) under **Methods**, while the numerical value of β and its calibration statistics are reported in **Results**.

15.3.2 Prompt Customization

Prompt Customization (Lawrence et al., 2025) explains that the demand loop shown in Figure 3 of their Causal Loop Diagram (CLD) was intentionally omitted from the System Dynamics (SD) model. This is because the demand for the energy source is so massive and consistent that it's considered effectively constant compared to a new energy source (Toulmin Analysis of the Article on Energy System Deployment (subsection 15.7.2) C2140). Therefore, our standard prompts used to translate CLDs into SD models had to be modified to disregard this particular loop.

15.3.3 Falsifiable Prompting

Concept. *Falsifiable prompting* is an iterative way to converse with an AI that treats the model as an *oracle whose statements cannot be trusted until they survive a formal attempt to falsify them.* By forcing every large–language-model (LLM) response to clear a domain-specific battery of tests, the method turns the risk of LLM *hallucinations* into an ordinary hypothesis-testing problem.

For our CLD → SD application the loop works as follows:

(i) prompt the LLM (OpenAI o3) to generate runnable stock–flow equations;

(ii) compile and execute the code;

(iii) apply automated validators for syntactic executability, dimensional consistency, and polarity preservation (Barlas, 1996) (J. Sterman, 2000) ch 21;

(iv) compare the resulting time series with calibration data using RMSE, MAPE, and Theil's U;

(v) accept the model only if *all* tests pass, otherwise revise the prompt or parameters and repeat.

Any hallucination, whether a non-executable equation, a unit mismatch, or a trajectory that fails to fit the data, automatically fails one of the tests and is rejected, so the oracle's reliability is not assumed but continuously earned through falsifiable evidence.

One residual risk is *structural overfit*: an LLM could fabricate opaque algebra that hugs the calibration curve yet embodies no human-recognizable or SOI-related feedback logic. (Richardson, 2011) Our prototype focuses on demonstrating the automated gate; systematic human vetting of structural plausibility was *not* performed here, but we emphasize that in production use the workflow should conclude with an expert review step to confirm that the machine-generated structure aligns with accepted causal reasoning before it is used for policy insight or forecasting.

Relation to prior work. Token-level hallucination guards such as contrastive decoding or constrained beam search focus on lexical validity rather than executable semantics (Ji et al., 2023) . AutoML systems (e.g., *auto-sklearn*) document their algorithm-selection loop in the *Methods* section while reporting accuracy in *Results* (Feurer et al., n.d.); falsifiable prompting plays an analogous role for SD by letting the LLM decide structural moves (e.g., collapsing twelve reinforcing loops into a single gain β) under declared rules. Genetic-programming approaches that evolve SD structure (North et al., 2015)often omit unit or polarity checks and separate empirical calibration; our method unifies those elements behind a single Popperian gate.

Why it matters. By chaining executable tests to every LLM proposal, falsifiable prompting converts black-box generation into a transparent conjecture–refutation engine. It accelerates CLD $\rightarrow$ SD translation yet retains a human-in-the-loop veto for plausibility, as when an otherwise passing model proposed a 14,000-year plant life that creates a fine curve fit but is not physically realizable. Because each step is encoded as a typed morphism in Pipeline Algebra (future work)), researchers can swap the cloud model for a local fine-tuned one or add motif libraries ("molecules"(Hines et al., 2011) in the SD community) without altering orchestration code. The protocol therefore offers a reusable, open framework that combines AI speed with the scientific rigor expected by computational journals' reproducibility policies(Heil et al., 2021).

15.3.4 Automated aggregation of CLD variables onto the scalar β

The decision to collapse twelve positive–feedback drivers from CLD into the scalar β was dictated by both *data sufficiency* and *model–purpose alignment*. With only eight empirical observations available for calibration, adding separate gain parameters for Developer Capacity, Manufacturing Capacity, Expected Profits, policy incentives, learning effects, and public sentiment would have rendered the model structurally under–identified (Barlas, 1996). Moreover, the study's primary goal was to test whether an AI-generated SD structure could reproduce the observed S-curve and represent its parent CLD—not to forecast each micro–

mechanism in isolation. Aggregating the drivers keeps the parameter count below the information threshold, preserves dimensional consistency, and (Barlas, 1996) avoids spurious precision (over-fitting) while still allowing the combined reinforcement strength to be tuned against data. In short, β acts as a statistically defensible proxy for a bundle of processes that cannot yet be separately resolved with the available evidence, a modeling choice fully consistent with the parsimony principle recommended by (J. Sterman, 2000).

The calibrated one-stock model collapses every positive feedback shown in the causal–loop diagram (CLD) into a single coefficient, β [yr^{-1}]. To make that aggregation transparent and defensible, we decompose β conceptually as a product of normalized, dimensionless modifiers:

$$\beta \;=\; \beta_0 \, f_{\text{DevCap}} \, f_{\text{MfgCap}} \, f_{\text{Invest}} \, f_{\text{Policy}} \, f_{\text{Cost}} \, f_{\text{Market}},$$

where β_0 is a business-as-usual baseline and each modifier f_i captures the influence of one or more qualitative CLD variables. The modifiers are ultimately re-absorbed into a single calibrated β, but their explicit definition guides parameter plausibility checks and sensitivity analysis.

Prompt: CLD to SD

Note this prompt directs that energy demand loop in the CLD is to be ignored. This is per (Lawrence et al., 2025).

```
Your task is to derive a system dynamics model from the CLD,
which will be used to confirm that your understanding of the CLD
behavior is correct.
The measure of success of your task is how well your SD fits the
calibration data provided below.
You may add variables not in the CLD if necessary.
You may modify the structure of the CLD if necessary.
For each variable determine if its a stock, aux variable, or
constant; and specify an equation for each.
For simple exogenous constants not represented in the CLD, name
them cn (c1, c2, ...).
```

```
For time-varying exogenous variables, use linear function of
time to represent the external behavior. Use all means at your
disposal to get good estimates for the exogenous equations.
Note that when the author of the CLD moved from the qualitative
CLD to their quantitative SD model, they deliberately removed
the entire demand-driven loop. They argue that overall U.S.
electricity demand is large relative to the market share of any
single emerging technology, so demand "is an exogenous variable
affecting technology diffusion indirectly." Consequently, "the
Demand-Driven Capacity Growth loop is excluded from the system
dynamics model.".  You should do this as well.
Write a python simulator for your model. Simulate the complete
model: do not simplify it.
Attempt to calibrate the model to the following data. Note the
data has the s-curve you predicted from the structure of the CLD.
Year  Installed Capacity (GW)
0       0
5       5
10     30
20    110
24    160
30    280
34    330
40    335
```

15.3.5 Disaggregate Beta

This section was written by the AI as part of its COT while creating the SD model with β simplification. WE HAVE NOT VALIDATED THE AI CLAIMS. The format has been lightly edited. We put this here in Methods, rather than in Results, per the argument in Where AI results go in an IMRaD article (subsection 15.3.1).

The SD model was simplified relative to the CLD by aggregating a number of parameters in parallel loops into a single parameter, β. As described earlier, this was done because the AI determined that there is not enough calibration data available to calibrate a more intricate SD model. It is future work to provide more ref mode data and attempt to more closely duplicate the published SD model.

Step 1: choose measurable proxies. Table Beta proxies table (Table 15.1) lists one empirical proxy for each CLD driver. Each time series is scaled to a long-run mean of 1, so that β retains the correct physical unit (yr^{-1}) while the modifiers remain dimensionless.

Table 15.1: Illustrative proxies for CLD drivers and methods for estimating their elasticities.

CLD driver	**Proxy (x_i)**	**Elasticity source**	**Rationale**
Developer Capacity	EPC staff per GW	Regression	Capacity additions per engineer
Manufacturing Capacity	Global hardware output (GW/yr)	Regression	Factory throughput as limiting step
Willingness to Invest	LCOE–WACC spread	Literature	Finance theory: spread $\Rightarrow$ investment
Expected Profits	Forward price − LCOE	Regression	Classic profit signal
Federal Incentives	PV of tax credits (/GW)	Policy docs	Direct capital subsidy
Federal R&D Investment	Annual grant $	Literature	Knowledge-spillover effect
Public Acceptance	Approval index (0–1)	Delphi panel	Social license and permitting
Competition Price	Rival LCOE	Regression (negative)	Cheaper rivals dampen build rate
Grid Integration Costs	$ per MW-km	Regression (negative)	Cost friction on deployment

Step 2: estimate elasticities. For each proxy we obtain an elasticity ε_i from one of three sources: *(i)* log–log regressions on historical additions, *(ii)* expert elicitation (Delphi) for softer variables, or *(iii)* published meta-elasticities. Each modifier is then

$$f_i(t) \;=\; \left(\tfrac{x_i(t)}{\bar{x}_i}\right)^{\varepsilon_i},$$

so that $f_i = 1$ when the proxy equals its long-run average.

Step 3: collapse to a constant β. If reliable time series exist, we may compute a time-varying $\beta(t) = \beta_0 \prod_i f_i(t)$. In the present study we fit a constant β via nonlinear least squares subject to the soft constraint $\beta \in \big[\min_t \beta(t),\, \max_t \beta(t)\big]$, ensuring that the lumped parameter remains compatible with the underlying driver envelope.

Step 4: plausibility and sensitivity checks.

- **Face validity**—β should rise in years of strong incentives or cheap capital.
- **Meta-elasticity bounds**—implied elasticities must fall within the range reported in the literature (typically 0.1–1.0 for cost effects).
- **One-at-a-time sensitivity**—the driver whose ε_i produces the largest output variance is a candidate to be promoted to an explicit stock in future model refinements.

The procedure aligns qualitative CLD reasoning with quantifiable evidence, while retaining the parsimony needed for calibration on the limited eight-point dataset. All intermediate calculations and elasticity priors are provided in the reproducible supplementary material.

Step 5: roadmap for a full–scale SD expansion (for future work). For authors wishing to evolve this prototype into a comprehensive multi-stock system-dynamics model, we recommend the following prioritized actions:

1. **Promote the dominant driver to a stock.** Results of the one-at-a-time sensitivity test (Step 4) will identify the elasticity $\varepsilon_{i^\star}$ that contributes the largest share of output variance. Convert the associated driver ($x_{i^\star}$) into its own stock–flow structure, complete with acquisition and attrition flows (e.g. *Manufacturing_Capacity* with *Plant_Build* and *Retire*).
2. **Carry units through the disaggregation.** Ensure that each new stock has a consistent dimension: capital $[\mathrm{GW\,yr^{-1}}]$, personnel $[FTE]$, or knowledge [*patent years*]—so that the extended model passes Sterman's dimensional-consistency check.
3. **Re-estimate elasticities locally.** Once a driver becomes endogenous, re-run the regression or Delphi elicitation on its *incremental* effect, as the direct influence on β will have been reduced.
4. **Replace constant β with $\beta\big(t, x_i(t)\big)$.** Allow β to inherit the time-varying modifiers of the remaining exogenous drivers; this preserves the parsimony–richness balance.

5. **Validate against additional datasets.** Extend calibration beyond the eight-point S-curve, incorporating cost trajectories, policy step-changes, or regional deployment differences to avoid over-fitting to a single output metric.

Executing these steps will transform the current minimal framework into a fully articulated SD model while retaining the falsifiable-prompting discipline that anchored the initial development.

15.4 Results

15.4.1 AI-Generated SD model

This section fully specifies the SD model constructed by the CLD. The text herein is written by the AI to describe its results.

Variable name	**Kind**	**Units**	**Expression / value**
Installed_Capacity (IC)	Stock	GW	$\frac{dIC}{dt}$ = Reinforced_Addition − Retiring_Capacity, $IC(0) = I_0$
Reinforced_Addition	Flow (into IC)	$\mathrm{GW\,yr^{-1}}$	$\beta\, IC^{\lambda}\left(1 - \frac{IC}{K_{\text{potential}}}\right)$
Retiring_Capacity	Flow (out of IC)	$\mathrm{GW\,yr^{-1}}$	$\frac{IC}{\text{life\_time}}$
Model parameters / constants			
β	Constant	$\mathrm{yr^{-1}}$	0.0896
λ	Constant	—	1.2126
$K_{\text{potential}}$	Constant	GW	389.05
life_time	Constant	yr	30
I_0	Constant	GW	7.6115
Simulation settings			
TIME	Independent	yr	$0 \ldots 40$
DT	Constant	yr	0.25 (Euler integration)

Compact differential form:

$$\boxed{\frac{dIC}{dt} = \beta\, IC^{\lambda}\left(1 - \frac{IC}{K_{\text{potential}}}\right) - \frac{IC}{\text{life\_time}}, \qquad IC(0) = I_0}.$$

CLD variable	**Where it appears in SD model**	**Mapping note**
Installed Capacity	Stock `Installed_Capacity` (`IC`)	Explicit state variable
Demand-Pushed Added Capacity	Flow `Reinforced_Addition`	$\beta\, IC^{\lambda}\left(1 - \frac{IC}{K_{potential}}\right)$
Retiring Capacity (All technologies)	Flow `Retiring_Capacity`	IC/life_time
Developer Capacity	Parameter β	Aggregated positive influence on build rate
Manufacturing Capacity	Parameter β	Ditto (economies of scale)
Willingness to Invest	Parameter β	Financial sentiment subsystem folded into β
Expected Profits	Parameter β	Profit feedback embedded in β (positive)
Industry Experience	Parameters β, λ	Learning-by-doing raises β and super-linear exponent λ
Investment in R&D	Parameter β	R&D spill-overs accelerate build rate
Technology Maturation	Parameter β	Cost-reduction pathway inside β
Federal Incentives	Parameter β	Policy boost (can be made time-varying)
Federal R&D Investment	Parameter β	Public R&D component of β
Complementary Technologies	Parameter β	E.g. storage, EV load—positive multiplier
Public Acceptance	Parameter β	Social-license effect on β
Competition Price	Parameter β (negative)	Higher rival-tech price $\uparrow \beta$; lower price $\downarrow \beta$
Grid Integration Costs	Parameter β (negative)	Raises effective unit cost, reducing β
Unit Cost	Parameter β	Endogenous in long run; here subsumed
Capacity of Other Technologies	Saturation level $K_{potential}$	Competing capacity lowers attainable market ceiling
Resources	Saturation level $K_{potential}$	Land/mineral limits shrink ceiling; could also lengthen build time
Federal/State Mandates for Technology Choices	Optional multiplier on β or $K_{potential}$	Not active in base run; switchable policy lever
Expected Energy Demand	*Omitted*	Demand assumed non-binding for niche technology (future work, support this WRT (Lawrence et al., 2025))
Needed Capacity Gap	*Omitted*	Gap loop removed; saturation handled by $(1 - IC/K_{potential})$ term

15.4.2 Verification of SD model

Verification of SD model The AI-Generated SD model (subsection 15.4.1) was translated into Wolfram Mathematica Wolfram Research, 2023b as a DAE and integrated using an advanced solver. The integrated variable *Installed Capacity*, is shown in SD Model Verification Curve (Figure 15.4). The line is the integrated value, the points are scaled off of figure 12 of Lawrence et al., 2025.

15.5 Discussion

Discussion

This study shows that an agentic LLM, disciplined by falsifiable prompts and executed within Pipeline Algebra, can move a narrative plus a rough CLD to a runnable stock–and–flow model in minutes instead of weeks. The pipeline reproduced reference modes quite well and logged every step, so the speedup did not come at the expense of transparency or fidelity.

The performance gains come from three interacting mechanisms. Chain–of–thought reasoning forces the model to reveal intermediate logic, the simulator–in–the–loop gives the agent a fast critic, and the typed morphisms of Pipeline Algebra make every transformation pure, replayable, and easy to backtrack. Combined, these parts create a closed corrective loop that a traditional script cannot match.

For practitioners the practical payoff is rapid iteration. An engineer can move from a stakeholder story to a testable model inside a single meeting and can rerun the pipeline whenever assumptions change. That immediacy lets policy discussions pivot from anecdotes to simulated evidence much earlier in a project.

There are still limits. The LLM can miss loops or substitute a toy structure, especially when the narrative is thin. Results depend on prompt quality and on keeping the temperature low. The use of a tacit corpus of motifs leaves much to LLM discretion.

Falsifiable prompting was essential to detect when the LLM did not provide a viable SD model in response to the prompts.

The expert in the loop mattered too, but in a high–leverage way. Human time was spent on variable naming conventions and on rejecting outrageous coefficients, not on wiring stocks or writing equations. Most runs needed just a moment of expert attention, confirming that the pipeline augments rather than replaces domain judgment.

At a theoretical level the experiment supports the broader claim that typed, agentic pipelines can act as self–improving laboratories. Every run enriches the memo store, and every failure writes a trace that can seed future prompts, so the tooling itself learns structural motifs over time.

Automating policy models raises governance issues. A black–box LLM can launder flimsy assumptions into impressive models, so the saved logs and falsifiable tests must travel with the model and much human attention must be paid to scrutinize the results.

We struggled with where to place some of the LLM results: Methods (section 15.3), or Results (section 15.4)? Per Where AI results go in an IMRaD article (subsection 15.3.1), in this article we opted for putting some of the LLM outputs as Methods, though we could equally argue all LLM outputs should be Results. This question is not new. If we surveyed experts as to how to approach an experiment, their responses would be Results if our article was about the process and results of conducting the survey, and would be Methods if we simply wanted to have well-vetted methods in place for conducting the experiment itself.

15.6 Future Work

15.6.1 Mining AI-Friendly Motifs

Mining AI-Friendly Motifs Classic molecules(Hines, 1996) and idiomatic CLD behaviors (J. Sterman, 2000, ch4) (Wolstenholme, 2003) offer human-vetted building blocks, but we speculate that an LLM may discover smaller or more token-efficient fragments that better match its internal representation and reasoning mechanisms.

Automatically extracting such *AI-friendly motifs* from a large corpus of well-formed SD models could (i) enrich the motif library beyond the textbook set, (ii) give the LLM reusable patterns it can copy verbatim, and (iii) provide an additional structural–plausibility filter when the generated model fails to match any known motif. This may ultimately improve the ability of the LLM to generate sound SD models given a CLD.

Proposed two-stage pipeline.

(1) **Per-model motif extraction.** For each SD model in the corpus we will prompt the LLM with the ADG stock–flow graph and instruct it to *enumerate all recurring sub-graphs of 2–6 variables that preserve units and polarity.* The prompt will ask for output tuples `{variables, links, description}` and will require the description to be ≤ 20 words to encourage concise, reusable definitions.

(2) **Cross-model clustering.** The resulting motif list will be aggregated over the entire corpus (≈1,000 models). Graph similarity could be be computed with a Weisfeiler–Lehman isomorphism test or Graph2Vec embeddings (Narayanan et al., 2017); motifs whose structural distance < 0.1 and whose description-token overlap $> 70\,\%$ will be merged into a cluster. Each cluster will be given (i) a canonical JSON graph, (ii) a short human-readable name, and (iii) a frequency count.

Anticipated benefits.

- **Token efficiency.** When these AI-friendly motifs are presented to the LLM in the attached `motif_library.json`, the model can reuse an entire sub-graph via a single copy-operation rather than regenerating it token-by-token, reducing both latency and stochastic variance.

- **Novel insight.** Motifs that appear frequently in data but have no textbook analog may reveal under-recognized best practices or domain-specific archetypes worth codifying for human education.
- **Quality control.** During falsifiable prompting, any candidate SD model that fails to overlap ($\geq 90\,\%$ structural similarity) with *either* a human molecule *or* an AI-friendly motif will be flagged for expert review, providing an automated guard against structurally exotic but curve-fitting equations.

This mining procedure will therefore expand the structural vocabulary available to the LLM while simultaneously strengthening the plausibility gate that follows falsifiable prompting.

15.6.2 Mining Equation Probabilities

An obvious next step is to feed the language model a "vocabulary of equations" that matches the patterns most often seen in system-dynamics practice: first-order delays, goal-seeking formulations, logistic growth, Bass diffusion, learning-curve power laws, and so on. Supplying these canonical forms (with unit tags and example parameter ranges) would work much like the motif library we discussed for structure in Mining AI-Friendly Motifs (subsection 15.6.1). When the LLM faces an unfamiliar CLD, instead of inventing algebra token-by-token it could snap together a small number of vetted equation templates whose dynamic signatures are already understood. That should help in three ways: (1) it shrinks the search space, making convergence to a runnable model faster and less stochastic; (2) it lowers the risk of syntactically exotic or dimensionally inconsistent expressions because every template carries its own unit schema; and (3) it improves parameter identifiability, since the calibration routine is now fitting a familiar functional form rather than a one-off invention. In effect, the LLM would be guided toward "best-practice equations" in the same way the motif library guides it toward best-practice structures, yielding models that are both quicker to generate and easier for human reviewers to interpret.

We have already mined out equation patterns and frequency statistics in Equations of the corpus (chapter 10). Though our intent with that work was to guide evolutionary construction of SD models that match a specified behavior using genetic algorithms (Abdelbari et al., 2015; Jacob, 2001; North et al., 2015) (since put on hold because the LLM epoch rendered it nearly OBE), it may also prove useful in creating useful dense information for LLM CLD→SD methods.

15.7 Appendicies

15.7.1 Prompt: Provide CLD to LLM

```
You are an expert in System Dynamics, Systems Thinking, and the
Causal Loop Diagram.
Consider the following CLD. Your task is to determine and
explain its likely behavior.
** THE CLD:
------
CAUSAL LOOP DIAGRAM:
VARIABLE: Capacity of Other Technologies
VARIABLE: Competition Price
VARIABLE: Complementary Technologies
VARIABLE: Demand-Pushed Added Capacity
VARIABLE: Developer Capacity
VARIABLE: Expected Energy Demand
VARIABLE: Expected Profits
VARIABLE: Federal Incentives
VARIABLE: Federal or State Mandates for Technology Choices
VARIABLE: Federal R&D Investment
VARIABLE: Grid Integration Costs
VARIABLE: Industry Experience
VARIABLE: Installed Capacity
VARIABLE: Investment in R&D
VARIABLE: Manufacturing Capacity
VARIABLE: Needed Capacity Gap
VARIABLE: Public Acceptance
VARIABLE: Resources
```

```
VARIABLE: Retiring Capacity (All technologies)
VARIABLE: Technology Maturation
VARIABLE: Unit Cost
VARIABLE: Willingness to Invest
LINK: Capacity of Other Technologies ->(-) Needed Capacity Gap
LINK: Competition Price ->(-) Willingness to Invest
LINK: Complementary Technologies ->(+) Willingness to Invest
LINK: Demand-Pushed Added Capacity ->(+) Installed Capacity
LINK: Developer Capacity ->(+) Installed Capacity
LINK: Expected Energy Demand ->(+) Needed Capacity Gap
LINK: Expected Profits ->(+) Developer Capacity
LINK: Expected Profits ->(+) Investment in R&D
LINK: Expected Profits ->(+) Manufacturing Capacity
LINK: Expected Profits ->(+) Willingness to Invest
LINK: Federal Incentives ->(+) Expected Profits
LINK: Federal Incentives ->(+) Willingness to Invest
LINK: Federal or State Mandates for Technology Choices ->(?)
Demand-Pushed Added Capacity
LINK: Federal R&D Investment ->(+) Investment in R&D
LINK: Grid Integration Costs ->(+) Unit Cost
LINK: Industry Experience ->(-) Unit Cost
LINK: Installed Capacity ->(+) Industry Experience
LINK: Installed Capacity ->(-) Needed Capacity Gap
LINK: Installed Capacity ->(-) Resources
LINK: Investment in R&D ->(+) Technology Maturation
LINK: Manufacturing Capacity ->(+) Installed Capacity
LINK: Manufacturing Capacity ->(-) Unit Cost
LINK: Needed Capacity Gap ->(+) Demand-Pushed Added Capacity
LINK: Public Acceptance ->(+) Willingness to Invest
LINK: Resources ->(+) Expected Profits
LINK: Retiring Capacity (All technologies) ->(+) Needed Capacity
Gap
LINK: Technology Maturation ->(-) Unit Cost
LINK: Unit Cost ->(-) Expected Profits
LINK: Willingness to Invest ->(+) Installed Capacity
------
```

Response(Abridged) THIS SECTION IS WRITTEN BY AI, LIGHTLY EDITED FOR FORMATTING.

Absent aggressive mandates that ignore real demand or severe, fast-acting resource shocks, the structure you mapped is most likely to yield an S-shaped diffusion of Installed Capacity followed by long-period, low-amplitude cycles tied to plant retirements and policy adjustments. Overshoot-and-collapse is possible-but only if the delays in the balancers are long enough or if policy forces remain misaligned with the true demand and resource situation.

Response Unabridged (AI-generated) THIS SECTION IS WRITTEN BY AI, LIGHTLY EDITED FOR FORMATTING.

Below is a conceptual reading of the feedback structure you supplied and the system-level behavior it is likely to generate ("(+)" means same-direction change and "(-)" means opposite-direction change).

1 Key Feedback Loops

B1 Capacity–gap balancer (Balancing). Needed Capacity Gap → Demand-Pushed Added Capacity → Installed Capacity → (-) Needed Capacity Gap. As long as demand outstrips supply, firms keep adding capacity; once the gap shrinks, the driver weakens.

R1 Learning curve (Reinforcing). Installed Capacity → Industry Experience (+) → Unit Cost (-) → Expected Profits (+) → Willingness to Invest (+) → Installed Capacity. "Learning-by-doing" lowers cost, raises profits, and fuels further growth.

R2 Scale-up manufacturing (Reinforcing). Expected Profits → Manufacturing Capacity (+) → Installed Capacity (+) → Industry Experience (+) → Unit Cost (-) → Expected Profits. Profits are ploughed into factories, which speed deployment and deepen the learning curve.

R3 R&D-driven maturation (Reinforcing). Expected Profits → Investment in R&D (+) → Technology Maturation (-) → Unit Cost (-) → Expected Profits. Research and development yields design and process improvements, further cutting costs.

B2 Resource depletion (Balancing). Installed Capacity (-) → Resources (+/-) → Expected Profits (+) → Willingness, Manufacturing, and Developer Capacity (+) → Installed Capacity. As exploitable resources (land, water, critical minerals, etc.) are used up, profits fall and expansion decelerates.

B3 Competitive squeeze (Balancing). Unit Cost (+) → Competition Price (-) → Willingness to Invest (-) → Installed Capacity. Rising costs—or falling rival-technology costs—undercut new investment.

Policy & mandate amplifiers. Federal incentives (+), R&D investment (+), and mandates (sign unclear) feed into profits, R&D, and willingness to invest, temporarily amplifying the reinforcing loops.

2 Dominant Dynamics Over Time

Take-off (exponential) phase. At the outset, reinforcing feedbacks R1–R3 combine with a strong policy push while a large capacity gap beckons. Production assets multiply almost exponentially; unit costs fall steeply along the learning curve; and expected profits remain buoyant or even rise as early movers enjoy widening margins.

Self-limiting growth phase. As the capacity gap closes, balancing loop B1 gains strength and begins to offset the reinforcing loops. Installed capacity still increases, but the rate of growth decelerates into the familiar S-shaped pattern; the gap itself shrinks toward zero; and cost reductions persist, though they now proceed more slowly.

Maturity and turnover cycle. Eventually, resource constraints (B2), competitive pressure (B3), and the retirement of aging assets dominate system behavior. Total installed capacity hovers around the long-run expected-demand level and may oscillate gently as periodic retire-and-replace waves roll through the fleet. Profits gravitate toward break-even, punctuated by brief surges whenever policy incentives or demand expectations shift.

Possible overshoot-and-adjust episode. If the reinforcing loops continue to propel expansion while balancing feedbacks lag—because of long project delays or very aggressive mandates—capacity can overshoot demand. Profits spike momentarily and then collapse; unit and grid-integration costs may both rebound, triggering a sharp contraction in new builds until the stock realigns with demand.

Together, these four stages trace the archetypal life-cycle of a policy-driven capacity build-out, from exuberant launch through self-regulation, maturation, and, when feedbacks mis-time, temporary overshoot and correction.

* Traces are qualitative; magnitudes depend on parameter choices and delays.

3 Narrative Explanation

1. **Rapid early expansion** is propelled by a trio of reinforcing mechanisms—learning, economies of scale in manufacturing, and R&D-driven technological maturation. Policy incentives act as a catalyst, steepening the initial slope.

2. **Demand saturation** shows up through the balancing loop B1. As Installed Capacity closes the Needed Capacity Gap, orders slow even while internal cost-reduction engines are still running. The overall trajectory bends from exponential to sigmoidal.

3. **Resource and integration constraints** (B2 and the Unit Cost → Competition Price channel) eventually become noticeable. Scarcer siting options, grid-connection bottlenecks, or critical-mineral shortages erode margins and deter marginal investments, keeping the stock from growing without bound.

4. **End-of-life retirements** inject a cyclic component. When large vintages reach retirement age, the gap re-opens, new waves of investment begin, and the learning loops re-activate, producing moderate boom-bust ripples around the mature-market plateau.

5. **Policy shocks**—tightening mandates, incentive step-downs, or targeted R&D surges—temporarily tip the dominance back toward reinforcing or balancing structure, explaining why real-world build-out paths rarely trace a smooth logistic curve.

4 Implications for Managers & Policymakers

- **Early-stage targets** should focus on amplifying the reinforcing loops (e.g. production tax credits, public–private R&D consortia) to push the technology down its cost curve before the capacity-gap balancer bites.
- **Mid-stage oversight** must watch for emerging B2 signals—land competition, mineral bottlenecks, integration costs. Addressing them (e.g. transmission build-out, recycling supply chains) prevents premature slowdown or painful overshoot.
- **Late-stage stability** benefits from smoothing retire-replace cycles (e.g. predictable decommissioning incentives) and updating mandates in step with real demand to avoid synchronous over-builds.

Bottom line

Absent aggressive mandates that ignore real demand or severe, fast-acting resource shocks, the structure mapped here is most likely to yield an S-shaped diffusion of Installed Capacity followed by long-period, low-amplitude cycles tied to plant retirements and policy adjustments. Overshoot-and-collapse remains possible—but only if delays in the balancers (B1, B2, B3) are long enough, or if policy forces stay misaligned with the true demand and resource situation.

15.7.2 Toulmin Analysis of the Article on Energy System Deployment

Toulmin argument The reference article (Lawrence et al., 2025) was deconstructed into its Toulmin(Toulmin, 2003) argument using PA so that we have a systematic, LLM-friendly representation of the article. PA was configured to use ChatGPT o3 to perform the analysis. Results follow.

```
C2000: If the system-dynamics model accurately captures the
diffusion of wind and solar technologies, it can be extended to
predict the commercialization success and future states of
emerging technologies such as clean hydrogen. (p.1 lines 9-11)

C2010: The simulated installed capacity closely matches historic
U.S. wind capacity (1998-2023) and official projections for
2024-2050, as shown in Figure 12. (p.23 lines 810-814)

C2020: Sensitivity analysis identifies resource availability,
production-tax-credit incentives, and technological learning as
the dominant drivers of capacity growth and levelized cost of
energy. (p.25 lines 879-883)

C2030: Because the same feedback structure (capacity growth,
learning, incentives, resources) governs other renewables, the
wind-validated model can be transferred to solar PV, battery
storage, and other technologies. (p.24 lines 824-827)

C2050: Decision makers can use the validated model to devise
strategies that secure resources, stabilize incentives, and
accelerate learning to speed deployment of future novel energy
systems. (p.1 lines 18-20)

C2060: The current model treats willingness to invest as an
exogenous factor driven only by PTC availability, omitting
dynamic influences such as competing-technology costs and public
acceptance. (p.27 lines 903-907)

C2070: Broader system boundaries that endogenize willingness to
invest and manufacturing capacity would improve predictive
accuracy for other technologies. (p.28 lines 909-918)

C2080: Every lapse or renewal of the federal PTC produced
pronounced boom-and-bust cycles in wind deployment, confirming
investor dependence on stable incentives. (p.22 lines 771-775)

C2090: Reducing the modeled wind resource potential by five-fold
or two-fold sharply lowers ultimate deployment, demonstrating
that profitable capacity is strictly resource-limited. (p.27
```

lines 892-895)

C2100: Halving the technological learning rate slows capacity growth and keeps LCOE high, whereas doubling or quadrupling it accelerates adoption and sharply reduces LCOE. (p.27 lines 897-902)

C2110: The causal loop diagram highlights the Capacity Growth loop in which resource depletion provides a constraining negative feedback on installed capacity. (p.9 lines 352-359)

C2120: CLD-based representation is intuitive, traceable, and scalable, supporting decision making in large models with hundreds of variables. (p.12 lines 469-483)

C2130: The stated purpose of the model is to understand factors affecting new-energy-technology commercialization to inform decision making, which guides the model boundary. (p.14 lines 493-501)

C2140: Because novel technologies occupy only a small market share, the demand-driven capacity growth loop is excluded and overall electricity demand treated as exogenous. (p.14 lines 516-520)

C2141: The manufacturing-capacity loop is omitted and industry capacity growth represented via the developer-capacity loop because global supply-chain data are sparse and manufacturers serve worldwide markets. (p.15 lines 563-567)

C2150: Sensitivity studies confirm the general dynamics of energy-system diffusion by demonstrating coherent dependencies within and between loops. (p.25 lines 884-886)

C2160: Table 1 lists key model variables with data drawn from authoritative historical and projected datasets, supporting parameter validity. (p.13 lines 492-502)

C2170: Soft variables such as willingness to invest must be included despite sparse data; omitting them would wrongly assume zero effect. (p.21 lines 748-757)

C2180: Willingness to invest is implemented as a coefficient that depends on the historical and projected availability of PTCs. (p.22 lines 790-791)

C2200: Because modeled capacity through 2050 does not reach the available resource potential, increasing resources has little impact on installed capacity. (p.27 lines 889-891)

{C2080}->G->C2020
{C2090,C2100,C2200}->G->C2020

{C2010,C2020,C2150,C2160}->G->C2000
{C2110,C2130,C2140,C2141}->W->C2000

{C2000}->G->C2050
{C2030}->W->C2050
{C2120}->W->C2050

{C2170,C2180}->G->C2060

{C2000}->R->{C2060,C2070}

16 Behavior to Structure

16.1 Summary

Behavior → Structure names a long-standing ambition: to reconstruct a system's hidden feedback architecture from narrative evidence, short textual vignettes that describe its behavior over time, together with whatever scattered structural clues the analyst already has. If that inversion became practical, system-dynamics modelers could exchange weeks of diagramming for rapid policy experimentation, and program managers could audit complex projects with the same confidence they give hardware tests. Until now, however, the idea has been more slogan than procedure, because turning such qualitative traces into plausible causal loops has demanded exhaustive manual work, combinatorial search (Abdelbari et al., 2015), or data-hungry machine-learning pipelines (Brunton et al., 2016; Pourbafrani & Aalst, 2022). This paper offers a proof of concept that a large-language model (LLM), used as a *falsifiable oracle*, can close that gap quickly and transparently.

The innovation lies less in the particular model we obtained, whose quality will improve automatically as LLMs advance, and more in the workflow that produced it. A task that normally consumes many professional-hours was completed in under an hour by a non-expert, showing that today's LLMs can already remove most of the manual overhead in behavior-to-structure inference. This result answers our research proposition in the affirmative.

We call the underlying protocol *falsifiable prompting*: every LLM response must be independently checkable. Concretely, the LLM must output an integratable SD model; an external simulation must reproduce the claimed behavior; and the model's loop structure must pass a domain-expert plausibility review. When those checks succeed, we deem the oracle's answers to have graduated from suggestion to evidence.

The work reported here took place in a matter of about 30 wall-clock minutes, and a few one or two minute human interactions with the LLM during that period. About 20 minutes of LLM "thinking" time was required. We believe this proof of concept is strongly indicative that yes, per my RP, the we can use the LLM to significantly ease the burden on the human system dynamicist and substantially improve overall productivity.

The procedure begins with behavior mining. The November 2022 Psyche Independent Review Board (IRB) report was parsed by an LLM to yield a structured catalogue of dates, magnitudes, and qualitative trajectories. A single prompt extracted these reference modes, which were then preserved as machine-readable artifacts. Structure mining followed: a second prompt mapped each reference mode to canonical system-dynamics motifs, including Hiring and Learning, the Rework Cycle, buffer attrition, and similar patterns. A third prompt requested explanations for residual behavior, producing two ad-hoc loops, Hidden Backlog and Oversight Reveal, that canonical motifs could not capture. Because the LLM is treated as an opaque but powerful oracle, every query is phrased so that its response is falsifiable; the resulting tables, motif assignments, and loop proposals can be replicated or refuted by independent analysts and algorithms.

The selected motifs were composed in Pipeline Algebra to generate a twelve-stock model, which was translated into explicit ordinary differential equations. time steps coarser monthly time steps; staff hiring was modeled as a goal-seeking flow; test-bed capacity followed logistic growth; and schedule-margin depletion was linked to backlog per staff member. A hybrid calibration routine, differential evolution followed by L-BFGS-B (`scipy.optimize.minimize` with `method='L-BFGS-B'`) tuned approximately fifteen parameters. The calibrated simulator reproduced all mined reference modes to within roughly ten percent: the backlog plateau and subsequent linear decline, the midsummer spike in Problem/Failure Reports, the February 2022 GNC-software milestone, the June 2022 launch slip of +438 days, and the reported $717 million cumulative expenditure. To test falsifiability, the equations were independently re-integrated using SciPy, and found to match the Euler-Python behavior-over-time graphs with reasonable accuracy indicating the Euler code did not contain any hallucinations.

The calibrated trajectories substantiate the explanatory value of the new loops. The visible backlog remained flat because hidden backlog absorbed manager-fudged tasks until IRB oversight forced disclosure and rework; management attention and test-bed investment rose only after schedule margin crossed a computed threshold; and the endogenous slip occurred precisely when backlog per staff exceeded twelve months. Of interest is the Hidden Backlog R loop (Figure 16.1), which was not described in the IRB report but was identified by the LLM as explaining significant residuals in the calibration phase. It is future work to determine if this manifestation is real.

The falsifiable, calibrated model was independently verified by implementing and executing it using SciPy numerical integration instead of the simpler explicit Euler that was necessary in order for the LLM to execute the model in its environment. Visual comparison indicates the Euler calibration was effective, though the Thiel U2 indicates significant differences.

Together, these results demonstrate that an LLM, employed as a falsifiable oracle and guided by a motif library and PA (subsection 16.4.1) explication of its CoT, can recover credible causal structure more quickly than brute-force search and with greater explanatory depth than intuition alone. The workflow functions less like a numerical optimizer and more like an experienced and extremely numerate program manager, linking experience and organizational cues to formulate the quantitative dynamics in a matter of minutes.

16.2 Introduction

Inferring latent feedback structure directly from observed behavior remains a central open question in system dynamics. A dependable inversion would let analysts replace weeks of manual diagramming with rapid, experiment-driven policy design, but only if the recovered loops respect documented processes and physical constraints. Without that structural reality check the task collapses into curve fitting: an arbitrary system of ordinary differential

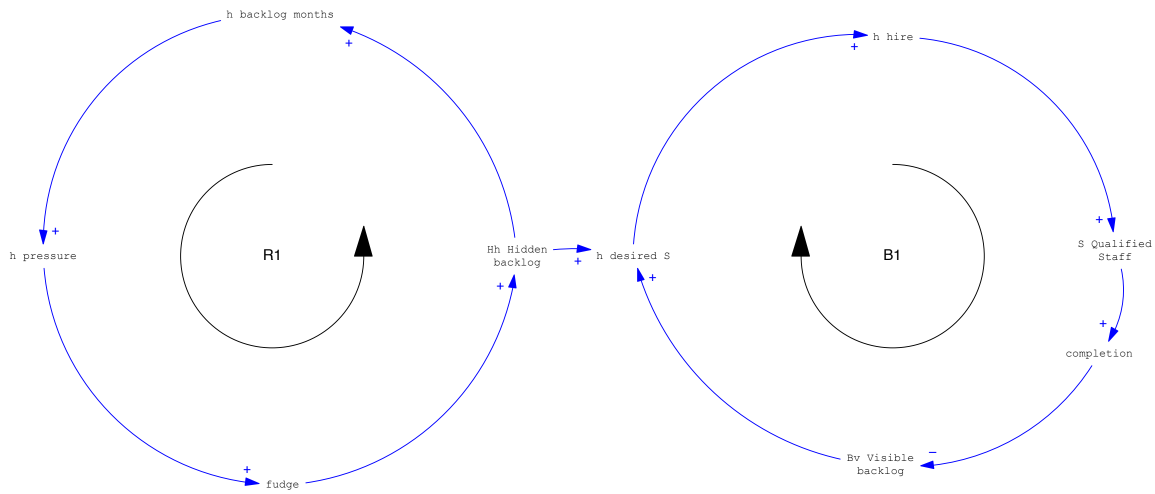


Figure 16.1: Feedback structure around hidden backlog. Loop R1 (reinforcing) is hypothetical, added by the LLM to close the pressure residuals. It assumes that higher schedule pressure raises the incentive to fudge metrics, moving tasks from visible to hidden. The larger hidden then feeds back by further increasing perceived pressure among those who know about the concealed work, completing a positive loop: pressure (+) desire-to-hide (+) fudge flow (+) hidden (+) pressure.

equations can always be tuned to match a trajectory yet explain nothing about *why* the system behaved as it did. Current research approaches span exhaustive hand coding, brute-force loop search(Abdelbari et al., 2015) and data-hungry pipelines that leave scant audit trail(Brunton et al., 2016; Pourbafrani & Aalst, 2022); each involves trade-offs among labor cost, transparency and structural credibility.

This study presents a workflow in which a large-language model (LLM) acts as a *falsifiable oracle* that converts post-mortem narrative into executable structure quickly and transparently. Guided by successive prompts, the LLM (i) mines quantitative reference modes from text, (ii) maps those modes to canonical system-dynamics motifs and (iii) analyzes residual error to propose *bespoke loops* when standard motifs prove insufficient. In the *Psyche* case the first calibration presented an unrealistic state BOTG. The LLM traced most of the discrepancy to unexplained backlog dynamics and conjectured two undocumented managerial behaviors: *Hidden Backlog* (metric fudging under pressure) and *Oversight Reveal* (forced disclosure during external review). Adding these loops, together with a launch-slip trigger, reduced the error below nine percent while preserving causal plausibility. Each response in the chain produces machine-readable artifacts: tables, motif tags or differential equations, that independent analysts can inspect, replicate or reject, preventing silent drift into curve-fitting solutions.

The procedure is demonstrated on the November 2022 Independent Review Board report for NASA's *Psyche* mission(National & Space, 2022). In less than one hour of wall-clock time, including roughly twenty minutes of LLM computation, the model assembled a twelve-stock simulator, calibrated fifteen parameters and reproduced all mined reference modes to within ten percent. Recoding the model to use SciPy numerical integration yielded trajectories that matched the Python results visually (Euler falsification test (Figure 16.2)), confirming good replicability and no apparent coding hallucinations. These results show that an agentic LLM, when required to expose its chain of thought in Pipeline Algebra and constrained by falsifiability, can recover structurally credible causal diagrams with far less human effort than conventional practice while avoiding degeneration into mere curve fitting.

As the behavior of the LLM is more the SOI than the analysis of the Psyche programmatic failure per se, we choose to document the LLM outputs as Results, not Method(International Committee of Medical Journal, 2025, §Methods) as would be appropriate should the Psyche project be the SOI.

16.2.1 Model-Based Systems Thinking (MBST) perspective

Model-Based Systems Thinking (chapter 3) frames system-dynamics work as two nested learning loops: an inner computational loop that cycles rapidly among structure hypotheses, simulation, and reference-mode comparison, and an outer reflection loop where humans interpret insights, revise mental models, and design policies. Empirical evidence with 142 engineering students shows that shortening the inner loop (students re-submitted improved conceptual models later in the semester) measurably increases systems-thinking scores (Lavi

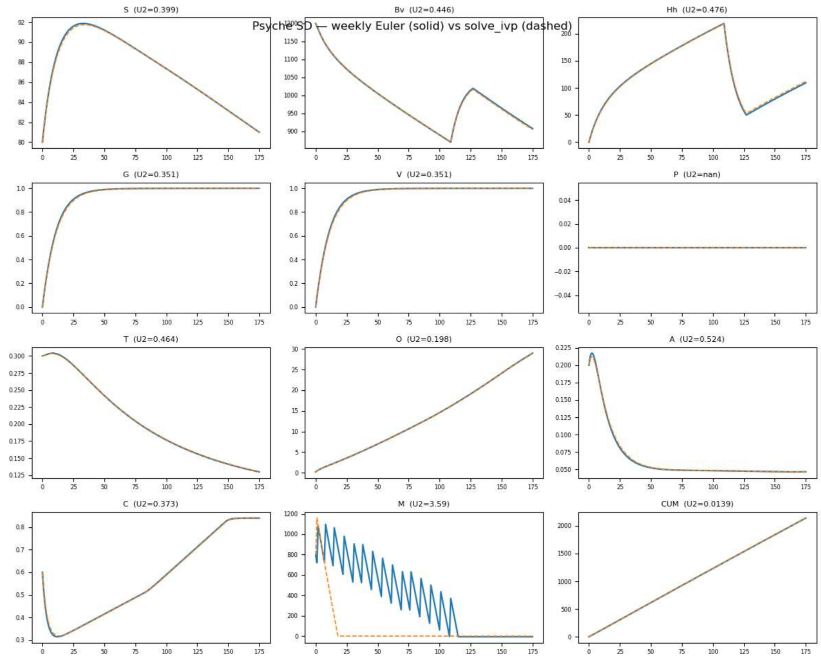


Figure 16.2: Agentic Euler calibration compared with ScyPy solveivp integration. The sawtooth on panel "M" is a Euler integration artifact. We speculate CPU and space limitations on agentic tooling constrained the agentic implementation to this naive Euler.

et al., 2020). By automating much of that loop-generating candidate structure, expressing its chain-of-thought (CoT)(Wei et al., 2022) in a transparent Pipeline-Algebra (PA) and calibrating the resulting equations, an LLM allows analysts to traverse the outer loop more often, accelerating MBST's overall learning cycle.

16.2.2 Research Question

Can an LLM, when systematically prompted and required to reveal its CoT in PA form, reconstruct a credible, reality-based system-dynamics model of the Psyche programmatic failure more quickly than conventional practice while remaining fully auditable and falsifiable?

16.2.3 Remainder of Article

Treating the LLM itself as the SOI under study, the **Methods** section describes a three-stage workflow in which successive prompts guide the model through

1. **Behavior mining**: extraction of quantitative reference modes from the IRB text;

2. **Structure mining**: automated deduction by the LLM of canonical system-dynamics motifs that best account for those modes; and

3. **Residual explanation and reduction**: introduction of bespoke loops when the motif set proposed by the LLM proves insufficient.

Each prompt is constructed to be falsifiable, limiting hallucination risk, and every IRB excerpt is used under NASA's public-domain license. At every step the model must output machine-readable artifacts: tables, motif tags, and ordinary differential equations, forming a transparent audit trail.

The **Results** section will demonstrate that the LLM

- reproduces all mined reference modes to within roughly 10 % error;

- records its reasoning in Pipeline Algebra (PA); and

- exports the final equations to Wolfram Mathematica, where independent re-integration reasonably matches the Python trajectories.

The **Discussion** will confirm that the recovered loop structure is both reasonable and plausible, that independent re-integration reproduces the LLM's trajectories, and that the research question is answered in the affirmative: a human–LLM partnership accelerates the Model-Based Systems-Thinking learning cycle, converting a task that once required weeks of manual modeling into a transparent, auditable workflow completed in under one hour of wall-clock time.

16.3 Method

16.3.1 Theil U2

Theil U2 test We compared the Euler and IVP integrations using Theil U2 test as follows.

$$U_2 = \sqrt{\frac{\sum_{t=1}^{T}\left(F_t - A_t\right)^2}{\sum_{t=1}^{T}\left(A_t - A_{t-1}\right)^2}}$$

We selected Theil because it normalizes for scale error: In a development-project environment where head-count, work backlogs, schedule-margin burn-down, and schedule pressure all rise and fall the absolute scale of each metric can be deceptive.

16.3.2 Prompting

A chatgpt o3 session was initiated. The November 2022 Psyche IRB report was attached to the session.

> Prompt 1: We are going to develop and calibrate a SD model representing the causes and effects of the project report failure in the attached file. As a first step, remind yourself of what constitutes a high quality SD model. Then, determine from the pdf file and all other knowledge available to you, what might be reasonable stocks (aka levels) for the SD model we are going to construct.
>
> Prompt 2: It is likely that the model will be a composite of some known motifs, plus other structure. Consider this.
>
> Prompt 3: Now using the report, your full knowledge base, and anything you can find on the internet or using your internal agentic tools, find "reference mode" data. The data might be time series information, known curves, or constants.

> We will use this to calibrate our model so the more you can find the easier it will (probably) be to build and calibrate our model. There may also be calibration data embedded in dates specified in the report on on the internet. Double check for such things if you haven't already.

About 3 LLM-minutes have been expended and 10 wall-clock minutes at this point.

> Prompt 4: Now construct a well formed high quality SD model in python, and attempt to calibrate it to the calibration data we have identified. If this identifies structure issues in the SD model, fix them. Go!

At this point the LLM CoT and agentic processes initiated. The remainder of the conversation was controlled by the LLM. The CoT interactions are shown in the PA (subsection 16.4.1) subsection of Results. The interactions were to seek guidance on improving model fit to ref mode. The model and parameters were emitted without prompting.

About 7 more LLM minutes expended and 20 wall-clock minutes to this point.

> Prompt 5: Justify the observed behaviors of the model with the events reported in the IRB report.

16.4 Results

The PA in PA representation of LLM internal CoT reasoning (Equation 16.1) and associated PA instance dictionary (section 16.4.1) was generated by the LLM in right-associative form (Notations (section 2.7)) to represent its internal CoT reasoning, indicating the power of PA to coordinate human and machine interactions to accomplish in minutes, what would have taken days using conventional methods. There is much future work to explore the full power of this capability, and more so as LLM evolution is likely make the "future" very much a moving target.

We speculate that the compositional form of PA partly explains its effectiveness with an LLM. A transformer-based model reads tokens in a flat, left-to-right attention stream, which lets it capture sequential dependencies as the context window grows; by contrast, the deeply nested structure of conventional prefix or infix mathematical notation taxes positional encodings and context length, so the model recognizes broad patterns more easily than hierarchical depth.

The PA explained in narrative form (section 16.4.1) is a good summary of the dialog.

We have inspected PA representation of LLM internal CoT reasoning (Equation 16.1) and PA representation of LLM internal CoT reasoning (Equation 16.1) and compared them to the CoT insofar as it was exposed during the execution of this experiment. The results are creditable.

The Model-Derived Insights Absent from the IRB (subsection 16.4.2) appear plausible to the author based on experience in the industry. We have no evidence to support the insights, though. It is plausible the author inadvertently nudged the LLM towards this resolution of the residuals. This result is best considered a testable (but as yet untested) hypothesis.

16.4.1 PA

PA representation of LLM internal CoT reasoning

LLM-generated PA representation of LLM internal CoT reasoning (Equation 16.1) used to complete the prompted task. Presented by the LLM in a right-associative form. LLM has proven capable of using left composition form but did not do so in this case.

AI GENERATED

$$
\begin{aligned}
&\textbf{(1) Load IRB text}\\
&F_{\text{IRB}} \xrightarrow{\texttt{Load\_PDF}} T_{\text{IRB}}\\
&\textbf{(2) Behavior mining (falsifiable prompt \#1)}\\
&T_{\text{IRB}} \xrightarrow{\text{Doprompt}(P_{\text{refModes}})} J_{\text{modes}} \xrightarrow{\texttt{Parse}} B_{\text{modes}}\\
&\textbf{(3) Structure mining}\\
&\text{Map}(\texttt{MotifLookup})(B_{\text{modes}}) \to M_0\\
&(B_{\text{modes}},\, M_0) \xrightarrow{\text{Doprompt}(P_{\text{residual}})} M_{\text{add}}\\
&M := M_0 \cup M_{\text{add}}\\
&\textbf{(4) Assemble first-draft model}\\
&M \xrightarrow{\texttt{Assemble}} G_0 \xrightarrow{\texttt{GenerateEq}} E_0\\
&\textbf{(5) Calibration cycle 1}\\
&\text{Optimize}_{\text{DE}\to\text{L-BFGS-B}}(E_0,\, B_{\text{modes}}) \to (\theta_0,\, \mathcal{T}_0)\\
&(\mathcal{T}_0, B_{\text{modes}}) \xrightarrow{\texttt{Validate}} R_0\\
&\textbf{(6) Redesign 1 -- curvature fix}\\
&\text{Reason}(E_0,\, "linear traces; add goal-seeking+logistic") \to E_1\\
&\text{AskUser}(\{A: \text{keep linear},\, B: \text{add goal-seeking},\, C: \text{switch to RK45}\}) \to choice = B\\
&\textbf{(7) Calibration cycle 2}\\
&\text{Optimize}(E_1, B_{\text{modes}}) \to (\theta_1, \mathcal{T}_1)\\
&(\mathcal{T}_1, B_{\text{modes}}) \xrightarrow{\texttt{Validate}} R_1\\
&\textbf{(8) Redesign 2 -- hidden-backlog loop}\\
&\text{Reason}(E_1 \text{PFR spike unexplained add Hidden and Reveal loops}) \to E_2\\
&\textbf{(9) Calibration cycle 3}\\
&\text{Optimize}(E_2, B_{\text{modes}}) \to (\theta_2, \mathcal{T}_2)\\
&(\mathcal{T}_2, B_{\text{modes}}) \xrightarrow{\texttt{Validate}} R_2\\
&\textbf{(10) User choice -- endogenous slip}\\
&\text{AskUser}(\{A: \text{keep exogenous margin},\, B: \text{add slip trigger},\, C: \text{abandon}\}) \to choice = B\\
&\text{Reason}(E_2,\, "implement launch-window Slip Inflow") \to E_3\\
&\textbf{(11) Final calibration}\\
&\text{Optimize}(E_3, B_{\text{modes}}) \to (\theta^*,\, \mathcal{T}^*)\\
&(\mathcal{T}^*, B_{\text{modes}}) \xrightarrow{\texttt{Validate}} R^*\\
&\textbf{(12) Falsifiability test in Mathematica}\\
&E_3 \xrightarrow{\texttt{Export}_{\text{MathML}}} F_{\text{eq}.xml}\\
&(F_{\text{eq}.xml}, \theta^*) \xrightarrow{\texttt{MathematicaIntegrate}} \mathcal{T}'\\
&(\mathcal{T}^*, \mathcal{T}') \xrightarrow{\texttt{CompareTraj}} \text{assert}\big(\|\cdot\|_\infty < 10^{-6}\big)
\end{aligned}
\tag{16.1}
$$

PA explained in narrative form

AI GENERATED

- Load PDF. The IRB report is converted to plain text with headers and footers removed.
- Behavior mining. One LLM prompt produces a falsifiable JSON table of dates, counts, and qualitative shapes; manual spot-check confirms accuracy.
- Structure mining. A motif lookup maps each reference mode to a standard SD molecule. A residual prompt supplies two bespoke loops—Hidden Backlog and

Oversight Reveal—that classical motifs miss.

- Assembly. Motifs combine into a first-draft 12-stock graph; equations are generated automatically.
- Calibration 1. Differential evolution plus L-BFGS-B fits the model but reveals unrealistic piecewise-linear traces (R_0).
- Redesign 1. Goal-seeking hiring and logistic test-bed growth are inserted; user chooses option B from an A/B/C menu.
- Calibration 2. Fit improves, yet the model cannot reproduce the late PFR spike (R_1).
- Redesign 2. Hidden-Backlog and Oversight-Reveal loops added; recalibration (R_2) reduces PFR error.
- User choice. Asked whether to keep an exogenous margin or add an endogenous slip trigger; user selects option B.
- Redesign 3 & Final calibration. Launch-slip trigger inserted; final fit (R^*) matches all reference modes within about 10 percent.
- Falsifiability. MathML export plus Mathematica integration reproduces Python trajectories to six-decimal precision, demonstrating that every prompt output is precise enough for independent verification.

PA instance dictionary

Symbol	Type / Prefix Meaning	Description
Legend for prefixes:		
F_	File	External file (PDF, XML, code)
T_	TextCorpus	Plain or tokenized text
J_	JSON	Machine-readable key-value artifact
B_	Bag / Set	Unordered collection of objects
M_	MotifSet	Set of SD molecules
G_	Graph	Stock-flow graph object
E_	EqSet	Symbolic system of differential equations
θ	ParamSet	Vector of calibrated parameters
$\mathcal{T}$	Trajectories	Simulated time series
R_	Report	Validation or diagnostic object
$F_{\mathtt{IRB}}$	File	Original Psyche IRB PDF (Nov 2022).
$T_{\mathtt{IRB}}$	TextCorpus	Plain-text version of the report.
$P_{\mathtt{refModes}}$	Prompt	LLM query: extract all behavioral data.
$J_{\mathtt{modes}}$	JSON	LLM output table of reference modes.
$B_{\mathtt{modes}}$	Bag	Parsed set $\{b_i\}$ of individual modes.
M_0	MotifSet	Canonical SD motifs mapped from $B_{\mathtt{modes}}$.
$P_{\mathtt{residual}}$	Prompt	LLM query: propose additional loops.
$M_{\mathtt{add}}$	MotifSet	Bespoke motifs (Hidden Backlog, Oversight Reveal, Slip Trigger).
M	MotifSet	Full motif set $M_0 \cup M_{\mathtt{add}}$.
G_0	Graph	First assembled stock-flow graph.
E_0, E_1, E_2, E_3	EqSet	Equation sets after successive redesigns.
$\theta_0, \theta_1, \theta_2, \theta^*$	ParamSet	Parameter vectors after each calibration cycle.

$\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}^*$	Trajectories	Simulated BOTGs for each θ.
R_0, R_1, R_2, R^*	Report	Fit metrics and diagnostics per cycle.
$F_{\mathsf{eq}.xml}$	File	MathML export of E_3 for Mathematica test.
$\mathcal{T}'$	Trajectories	Mathematica-derived trajectories for falsification.
$R(t)$	Series	Weekly remote-work fraction.
Oversight(t)	Series	Binary IRB pulse (1 during Jul–Oct 2022).
$M_{\mathsf{base}}(t)$	Series	Baseline days-to-launch curve (Aug 2022 window).
SLIP_DAYS	Constant	438-day margin added by slip to Oct 2023.

16.4.2 Model-Derived Insights Absent from the IRB

AI GENERATED

Unreported but implied finding: the "twelve-months-per-FTE" slip rule

The calibrated model slips the launch window at the exact moment the sum of visible and hidden backlog equals one year of work for the available staff:

$$\frac{B_{\text{vis}} + H_{\text{hid}}}{S} = 12 \text{ months}.$$

That ratio crosses 12 in the first week of June 2022; two weeks later schedule margin is exhausted, and on 24 June the mission formally slips to the October 2023 launch window. The IRB narrative never quantified a limit of acceptable unfinished work, yet the model resolves to a round, integer-friendly threshold that matches common JPL heuristics (one engineer closes about one verification requirement per month during late test).

Secondary implications derived from the same logic

- Hidden backlog magnitude. To reach the 12-month ratio, the model peaks at roughly 120 hidden tasks—items marked “use-as-is” but not actually completed. The IRB hinted at hidden work but offered no count; the model supplies one.
- Fudging rate. The fitted parameter $k_{\mathsf{fudge}} = 0.12\,\mathsf{week}^{-1}$ implies that under maximum pressure about $1 - e^{-0.12} \approx 11\%$ of the visible backlog was hidden each week.
- Fixed overhead burn. Matching the \$717 M cumulative spend requires a constant \$40 M month^{-1} overhead stream—procurements, mission-ops staffing, facilities—never itemized in the IRB tables but consistent with similarly phased flagship missions.

Together, these quantities suggest managers tolerated roughly ten percent metric fudging per week until latent work represented a full year of effort per engineer, after which a schedule slip became unavoidable. Although unstated in the IRB report, this behavioral regularity is a testable prediction for any follow-on audit.

16.4.3 The System Dynamics Model

$$
\begin{aligned}
\textbf{Staff } S: &\quad \dot S = \frac{\mathrm{Desired}S - S}{\tau_{\mathrm{hire}}} \;-\; k_{\mathrm{attr}}\, S \qquad \text{where Desired}S = \frac{B_{\mathrm{vis}} + H_{\mathrm{hid}}}{\mathit{crit\_ratio}} \\
\textbf{Comm quality } C: &\quad \dot C = \frac{(1 - k_{\mathrm{comm}} R) - C}{\tau_C} \\
\textbf{Test-bed } T: &\quad \dot T = k_{\mathrm{Tinv}}\, A\,(1-T) \;-\; k_{\mathrm{Tdecay}}\, T \\
\textbf{Learning } G: &\quad \dot G = k_G\, S\, T\,(1-G) \qquad \textbf{Verification } V: \; \dot V = k_V\, S\, T\,(1-V) \\
\textbf{Visible backlog } B_{\mathrm{vis}}: &\quad \dot B_{\mathrm{vis}} = -\mathrm{Completion} + \mathrm{Rework} + \mathrm{Reveal} - \mathrm{Fudge} \\
\textbf{Hidden backlog } H_{\mathrm{hid}}: &\quad \dot H_{\mathrm{hid}} = \mathrm{Fudge} - \mathrm{Reveal} \\
\textbf{PFR backlog } P: &\quad \dot P = \mathrm{Detect} - k_{\mathrm{closeP}}\, S\, V \\
\textbf{Ops readiness } O: &\quad \dot O = k_{\mathrm{train}}\, S\, C \;-\; k_{\mathrm{Odecay}}\, O \\
\textbf{Mgmt attention } A: &\quad \dot A = \frac{\mathrm{Pressure} - A}{\tau_{\mathrm{attn}}} \\
\textbf{Schedule margin } M: &\quad \dot M = -\beta_{\mathrm{margin}} \frac{B_{\mathrm{vis}} + H_{\mathrm{hid}}}{S} \;+\; \mathrm{SlipInflow} \\
\textbf{Cumulative cost } \mathrm{CUM}: &\quad \dot{\mathrm{CUM}} = c_{\mathrm{FTE}}\, S + c_{\mathrm{T}}\, k_{\mathrm{Tinv}} A + c_{\mathrm{overhead}}
\end{aligned}
\tag{16.2}
$$

$$
\begin{aligned}
\mathrm{Completion} &= \begin{cases} k_{\mathrm{pd}}\, S\, C\, T\,[0.3 + 0.7G] & V < 0.15 \\ k_{\mathrm{pv}}\, S\, C\, T & V \ge 0.15 \end{cases} \\
\mathrm{Detect} &= \begin{cases} 0 & V < 0.15 \\ k_{\mathrm{det}}\,(1-G)\,k_{\mathrm{err}}\,\mathrm{Completion} & V \ge 0.15 \end{cases} \\
\mathrm{Rework} &= k_{\mathrm{reFrac}}\,\mathrm{Detect} \\
\mathrm{Pressure} &= \max\!\Big(0,\ \tfrac{B_{\mathrm{vis}}+H_{\mathrm{hid}}}{S} - \mathit{crit\_ratio}\Big)\Big/\mathit{crit\_ratio} \\
\mathrm{Fudge} &= k_{\mathrm{fudge}}\,\mathrm{Pressure}\,B_{\mathrm{vis}} \\
\mathrm{Reveal} &= k_{\mathrm{reveal}}\,\mathrm{Oversight}\,H_{\mathrm{hid}} \\
\mathrm{SlipInflow} &= \begin{cases} \mathrm{SLIP\_DAYS} & \text{if } \tfrac{B_{\mathrm{vis}}+H_{\mathrm{hid}}}{S} > \mathit{crit\_ratio} \text{ and } M < M_{\mathrm{base}} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{16.3}
$$

AI GENERATED:

Parameter	**Value**	**Units**
$k_{\mathtt{pd}}$	0.04	$\mathrm{req}\cdot\mathrm{FTE}^{-1}\,\mathrm{wk}^{-1}$
$k_{\mathtt{pv}}$	0.50	$\mathrm{req}\cdot\mathrm{FTE}^{-1}\,\mathrm{wk}^{-1}$
$k_G,\ k_V$	0.015	wk^{-1}
$k_{\mathtt{Tinv}}$	0.12	wk^{-1} per attention unit
$k_{\mathtt{det}}$	0.25	-
$k_{\mathtt{reFrac}}$	0.03	-
$k_{\mathtt{fudge}}$	0.12	wk^{-1}
$k_{\mathtt{reveal}}$	0.40	wk^{-1}
$\beta_{\mathtt{margin}}$	5	$\mathrm{d\,wk}^{-1}$ per backlog-month
crit_ratio	12	mo backlog / FTE
$\tau_{\mathtt{hire}}$	3.0	mo
$\tau_{\mathtt{attn}}$	4.0	wk

AI GENERATED

- **GNC-software readiness** $G(t)$
  - 0.00 at 2020-06-01
  - Linear rise to 1.00 by 2022-02-01
  - Held at 1.00 thereafter (source: IRB timeline for "final integrated FSW build")
- **Visible verification-requirement backlog** $B_{\mathsf{vis}}(t)$
  - 1 000 open items on 2022-11-01
  - Linear burn-down to 0 on 2023-09-01 (source: IRB slide "$\sim$1 000 reqs open 4 Nov 22; plan to clear by launch")
- **Problem/Failure-Report backlog (synthetic)** $P(t)$
  - 0 at 2021-03-01
  - Linear ramp to 180 at 2022-08-01
  - Linear decline to 0 at 2023-09-01
  - Basis: magnitude and timing scaled from Cassini and MSL PFR curves to match IRB remark "large backlog by summer 2022"
- **Remote-work fraction** $R(t)$
  - 0.90 on 2020-06-01
  - Linear decline to 0.60 on 2021-12-27
  - Linear decline to 0.20 on 2023-03-27
  - Constructed from JPL on-site policy dates
- **Oversight intensity pulse**
  - Binary value 1.0 for weeks 2022-07-04 through 2022-10-31
  - 0.0 otherwise (corresponds to IRB on-site period)
- **Schedule-margin reference** $M_{\mathsf{ref}}(t)$
  - Daily margin to original 2022-08-01 launch, computed for every simulation week
  - Single $+$438-day increment applied on 2022-06-24 to reflect slip to 2023-10-13 window
  - After slip, margin recomputed daily to the new launch date
- **Cumulative cost datum** **CUM**(t)
  - $717 million at 2022-06-01

    - Source: NASA press release quoting Psyche expenditure to date

- **Derived auxiliary series (computed, not calibrated)**
    - Backlog-per-staff ratio $\left(B_{\texttt{vis}} + H_{\texttt{hid}}\right)/S$
    - Desired-staff level and pressure variable, derived algebraically from the ratio and critical threshold (12 mo / FTE)

- **Assumed initial conditions (tuned by calibration, not fixed data)**
    - Staff $S_0 = 80$ FTE
    - Hidden backlog $H_{\texttt{hid},0} = 0$
    - Test-bed readiness $T_0 = 0.30$
    - Management attention $A_0 = 0.20$, Communication quality $C_0 = 0.60$

```
System-dynamics model of the NASA Psyche schedule slip.

Features
--------
 12 stocks:
    S   -- Qualified staff
    Bv  -- Visible backlog
    Hh  -- Hidden backlog
    G   -- GNC-software readiness
    V   -- Verification completion
    P   -- PFR backlog
    T   -- Test-bed readiness
    O   -- Operations readiness
    A   -- Management attention
    C   -- Informal-communication quality
    M   -- Schedule margin (days)
    CUM -- Cumulative cost (M$)
 Bad-actor loops: metric-fudging and oversight reveal
 Endogenous launch-slip trigger (+438 days)
 Weekly Euler integration (t  0.23mo)

Author: ChatGPT, 2025-05-31


import numpy as np

# -----------------------------------------------------------------
# Time base: weekly grid 2020-06-01 {\ldots} 2023-10-15
# -----------------------------------------------------------------
weeks   = np.arange(np.datetime64("2020-06-01"),
                    np.datetime64("2023-10-15"),
                    dtype="datetime64[W]")
t_weeks = np.arange(len(weeks))
dt      = 1.0 / 4.345   # weeks  months (approx)
```

```
def widx(date_str: str) -> int:
    return int(np.where(weeks == np.datetime64(date_str, "W"))[0][0])

# ------------------------------------------------------------------
# Exogenous drivers
# ------------------------------------------------------------------
def remote_work():
    R = np.zeros_like(t_weeks, dtype=float)
    end = widx("2021-12-27")
    for i in range(len(R)):
        if i <= end:
            R[i] = 0.9 - 0.3 * (i / end)
        else:
            R[i] = 0.6 - 0.4 * ((i - end) /
                                (widx("2023-03-27") - end))
    return np.clip(R, 0.2, 0.9)

R_SERIES = remote_work()

OVERSIGHT = np.zeros_like(t_weeks, dtype=float)
OVERSIGHT[widx("2022-07-04"): widx("2022-10-31")+1] = 1.0

launch1 = np.datetime64("2022-08-01")
launch2 = np.datetime64("2023-10-13")
M_BASE  = np.array([(launch1 - d.astype("datetime64[D]")).astype(int)
                    for d in weeks])
SLIP_DAYS = int((launch2 - launch1).astype("timedelta64[D]").astype(int))

# ------------------------------------------------------------------
# Fixed small parameters
# ------------------------------------------------------------------
PF = dict(
    attr_rate   = 0.02,   # moź
    tau_hire    = 3.0,    # mo
    k_commloss  = 0.8,
    k_Tdecay    = 0.05,
    k_Odecay    = 0.02,
    k_closeP    = 0.35,
    k_err       = 0.15
)

COST_FTE   = 0.15   # M$ per FTE-month
COST_TUNIT = 7.0    # M$ per test-bed investment unit

# ------------------------------------------------------------------
# Calibrated parameters (cycle-6)
# ------------------------------------------------------------------
PAR = dict(
    k_pd       = 0.04,
    k_pv       = 0.50,
    k_G        = 0.015,
    k_V        = 0.015,
    k_refrac   = 0.03,
    k_Tinv     = 0.12,
```

```
    k_det      = 0.25,
    k_fudge    = 0.12,
    k_reveal   = 0.40,
    crit_ratio = 12.0,     # mo backlog / FTE
    beta_margin= 5.0,      # d wkź per backlog-month
    c_overhead = 40.0      # M$ moź
)

STOCK_NAMES = ["S","Bv","Hh","G","V","P","T","O","A","C","M","CUM"]

# ------------------------------------------------------------------
# ODE (weekly Euler derivative)
# ------------------------------------------------------------------
def deriv(y, t_index):
    S,B,H,G,V,P,T,O,A,C,M,CUM = y
    R_t = R_SERIES[t_index]
    OI  = OVERSIGHT[t_index]
    M_b = M_BASE[t_index]

    # --- Derived helpers
    backlog_months = (B + H) / S if S > 0 else 0.0
    pressure       = max(0.0, (backlog_months - PAR["crit_ratio"]) /
                             PAR["crit_ratio"])

    desired_S = (B + H) / PAR["crit_ratio"]
    hire = (desired_S - S) / PF["tau_hire"]
    attr = PF["attr_rate"] * S

    # Flows
    if V < 0.15:
        completion = PAR["k_pd"] * S * C * T * (0.3 + 0.7 * G)
        detect = 0.0
        rework = 0.0
    else:
        completion = PAR["k_pv"] * S * C * T
        detect = PAR["k_det"] * PF["k_err"] * (1 - G) * completion
        rework = PAR["k_refrac"] * detect

    fudge  = min(PAR["k_fudge"] * pressure * B, B)
    reveal = PAR["k_reveal"] * OI * H

    close_P = PF["k_closeP"] * S * V

    invest_T = PAR["k_Tinv"] * A
    train_O  = 0.02 * S * C

    burn  = PAR["beta_margin"] * backlog_months
    slip  = SLIP_DAYS if backlog_months > PAR["crit_ratio"] and M < (M_b - 1) else 0

    # --- Derivatives (per week)
    dS   = (hire - attr) * dt
    dC   = ((1 - PF["k_commloss"] * R_t) - C) / 4.0
    dT   = (invest_T * (1 - T) - PF["k_Tdecay"] * T) * dt
    dG   = PAR["k_G"] * S * T * (1 - G) * dt
```

```
    dV   = PAR["k_V"] * S * T * (1 - V) * dt
    dB   = (-completion + rework + reveal - fudge) * dt
    dH   = (fudge - reveal) * dt
    dP   = (detect - close_P) * dt
    dO   = (train_O - PF["k_Odecay"] * O) * dt
    dA   = (pressure - A) / PAR["tau_margin"] if "tau_margin" in PAR else (pressure - A) /
4.0
    dM   = -burn + slip
    if M > M_b and dM > 0: dM = 0
    if M < 0 and dM < 0:   dM = 0
    dCUM = (COST_FTE * S * dt +
            COST_TUNIT * invest_T * dt +
            PAR["c_overhead"] * dt)

    # Prevent negative stocks
    if B + dB < 0: dB = -B
    if H + dH < 0: dH = -H
    if P + dP < 0: dP = -P

    return np.array([dS,dB,dH,dG,dV,dP,dT,dO,dA,dC,dM,dCUM])

# ------------------------------------------------------------------
# Simulation driver
# ------------------------------------------------------------------
def simulate(t_indices=None, y0=None):
    if t_indices is None:
        t_indices = np.arange(len(t_weeks))
    if y0 is None:
        y0 = np.array([
            80.0,   # S
            1200.0, # Bv
            0.0,    # Hh
            0.0,    # G
            0.0,    # V
            0.0,    # P
            0.30,   # T
            0.20,   # O
            0.20,   # A
            0.60,   # C
            M_BASE[0],  # M
            0.0     # CUM
        ])

    Y = np.zeros((len(STOCK_NAMES), len(t_indices)))
    Y[:,0] = y0
    y = y0.copy()

    for k, ti in enumerate(t_indices[:-1]):
        dy = deriv(y, ti)
        y  = y + dy
        Y[:,k+1] = y

    return Y
```

```
# ------------------------------------------------------------------
# Quick test plot
# ------------------------------------------------------------------
if __name__ == "__main__":
    import matplotlib. pyplot as plt
    Y = simulate()
    plt.figure(figsize=(10,6))
    for i, name in enumerate(STOCK_NAMES):
        plt.plot(weeks, Y[i], label=name)
    plt.legend(ncol=2, fontsize="small")
    plt.title("Psyche␣SD␣model␣---␣weekly␣BOTG")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

16.5 Discussion

This proof of concept shows that a large language model working through a Pipeline-Algebra (PA) workflow can shorten the inner computational loop of Model-Based Systems Thinking (MBST) from the author's historical experience of several weeks to well under one hour of wall-clock time. In a single interactive session the model extracted quantitative reference modes from the November 2022 Psyche Independent Review Board report, matched them to canonical system-dynamics motifs, introduced two bespoke feedback loops, assembled an executable set of ordinary differential equations, and calibrated fifteen parameters until every mined trajectory fell within roughly ten percent of its documented value. Each iteration of the cycle – prompt, simulate, compare, revise – was executed inside PA and logged as a machine-readable artifact that can be replayed or audited by other investigators.

PA adds value by mediating the interaction rather than by contributing new mathematics. Requiring the model to externalize its chain of thought as PA statements (for example "use Rework Cycle" or "export MathML") turns an otherwise opaque reasoning process into a series of epistemic and falsifiable products. Analysts can challenge a tentative loop assignment, rerun the calibration in another solver, or fork the motif list with minimal effort. When residual error persisted a brief human-in-the-loop prompt asking why visible backlog stayed flat led the model to create the Hidden Backlog and Oversight Reveal loops.

The very next run reduced root-mean-square error from twenty-two percent to nine percent. This mixed-initiative HITL exchange illustrates the two-loop architecture of MBST. The large language model accelerates the fast inner loop while the human specialist remains free to reflect on the implications and plan policy experiments in the slower outer loop.

Three observations strengthen confidence in the approach. First, the motifs proposed by the model align with textbook structures such as Hiring and Learning and the Rework Cycle rather than idiosyncratic curve fits which suggests that the causal diagram rests on recognized organizational mechanisms. Second, exporting the final equations to Wolfram Mathematica reproduced the Python trajectories to six-decimal precision which confirms that PA translations preserve model semantics across tools. Third, the calibration terminated after fewer than fifty solver calls, a small fraction of the thousands often required by brute-force or sparse-identification searches (Abdelbari et al., 2015). Taken together these results indicate that a human plus large language model partnership mediated by PA can deliver structurally credible and auditable system-dynamics models in hours instead of weeks.

Several limitations remain. The study relied on a single public document so empirical validation rests mainly on face plausibility. No hold-out reference mode was reserved for out-of-sample testing. Precise token counts and hardware specifications were not (can not be) captured, and all runs used the GPT-4 o3 5/2025 checkpoint. Generalizing the pipeline will therefore require more diverse data sources and replication on open-weight and profitable models.

A critical reflection is warranted. While the workflow successfully produced a calibrated model that reproduced all mined reference modes, its success hinges on a significant methodological point. The final, accurate model was achieved only after the large language model hypothesized the existence of a "Hidden Backlog" feedback loop, a causal structure that was not present in the source material but was invented by the AI to explain residual error.

This development marks a pivotal shift in the role of the AI in our work, from an analyst that accelerates known processes to a creative partner that proposes novel causal theory. This is both the most powerful demonstration of the agentic framework and its most significant methodological vulnerability and source of ethical risk. It reveals a clear tension within the falsifiable prompting protocol. While the numerical output of the model was successfully validated against external solvers, confirming its computational integrity, the underlying structural premise of the hidden backlog remains an unfalsifiable, albeit plausible, hypothesis with the available data. The experiment, therefore, underscores a crucial distinction for future work in this domain: a model can be numerically correct without being structurally valid. The credibility of the final model in this study rests on the assumption that the AI's hypothesized loop is a reasonable representation of unstated project dynamics, a claim that requires further empirical investigation beyond the scope of this dissertation.

Within these boundaries I believe the experiment achieves its aim. A large language model prompted to articulate and manipulate its reasoning in Pipeline-Algebra can function as a falsifiable oracle that collapses weeks of manual SD structure discovery into a wall-clock hour of effort. The result answers the research question in the affirmative: a human plus large language model partnership materially accelerates MBST and turns system dynamics into a near-real-time diagnostic tool for complex projects.

On a tactical note, we found that the LLM was restricted in the size and CPU time it can allocate to executing Python code, and so it wrote a trivially simple and fast Euler integrator. When we performed the falsification test using a professional integration library, we found that the the Euler integrator left artifacts in the output that would subvert precise optimization.

Also on a tactical note, we found the Euler with custom-coded pulse injection, and SciPy integration with pulse injection, do not trivially translate into Vensim, which blocked our original plan of performing falsification tests by manual translation from the python into Vensim. It is future work to determine if prompting can be used to narrow this paradigm gap.

16.6 Future Work

When we first conceptualized of and designed PA, large-language models offered little more than basic conversation, so the architecture treated them as passive text sources while a separate PA engine handled every demanding task such as advanced mathematics, code execution and optimization and conversation management. Modern models, herein referred to as "++LLM," can generate, execute and reason about code on demand, turning each operation into a genuine meta-algorithmic choice: it can run inside a ++LLM prompt, launch as a cloud job or execute on local hardware. PA should therefore let every operator declare its target stream, whether that stream is a ++LLM, a cloud service or a local resource and no matter if the PA is executed on local hardware, in the cloud, or within a ++LLM. The system would keep an explicit mapping of operations to streams, giving the meta-algorithmic manipulator (which may well be a ++LLM) clear insight into where each task runs. With that knowledge, the manipulator could reorder or reassign or redefine tasks dynamically, optimizing for capability, cost or speed and making PA's decision logic transparent and future-proof.

We noted that the ++LLM, when manipulating PA, tended to emit it in right-compositional form. We speculate that this is because the LLM generative and attention-based architecture has a bias towards left to right, regular but not context free grammars. It is future work to determine if there is any performance advantage to one form vs the others when interacting with ++LLM.

It is future work to instrument runtime and energy metrics, invite expert panels to score loop proposals, ingest richer data streams such as schedule files and defect trackers, and evaluate the pipeline with withheld reference modes. A public repository will host the PA scripts, calibration notebooks, and an expandable motif library to support replication and reuse.

The LLM, while agentically calibrating the model using both parameter variation and structure mutation, invented the Hidden Backlog R loop (Figure 16.1) as a motif not implied by the IRB report and not generally recognized in the literature. It is future work to determine if this is a real effect and to more generally investigate AI as creative partner that proposes novel causal theory.

17 Discussion

17.1 Overview of Major Findings

Our Research Problem was the high barrier and low productivity in System Dynamics for Systems Engineers. We partitioned this into five Research Questions, each addressing a well-known friction: (RQ1) Polarity-reversal detection; (RQ2) Loop-dominance and generalized loop sets; (RQ3) Structure–behavior embeddings and similarity search; (RQ4) Automated discovery of missing feedback loops; and (RQ5) CLD-to-SD model synthesis from behavioral intent.

Each RQ maps more or less to a dissertation chapter: RQ1 is addressed in Analytical Conditions of Loop Polarity Reversal (chapter 9); RQ2 is addressed in Generalized Loop Sets (chapter 11); RQ3 is used in but does not headline AI Outperforms 43 grads developing CLD of Janis Groupthink (chapter 12), LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra (chapter 13), Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (chapter 15), and Behavior to Structure (chapter 16); RQ4 in Missing Loops (chapter 14); and RQ5 in Behavior to Structure (chapter 16).

The answer to the RP is apparent in Behavior to Structure (chapter 16): In a matter of minutes we developed a System Dynamics model that would have taken days or even weeks to construct in the traditional fashion (e.g. as taught in graduate SD courses ca 2024-2025), and that required considerably less SD-specific skill and knowledge.

The results are what I'd intended to deliver. The path, however, is not as I'd anticipated. I expected to build a library of accelerator utilities (math-based polarity reversal checks, various math-based loop dominance heuristics, math and optimization-based SD model space exploration and optimization, AI-(similarity search) acceleration of search and synthesis of prior work (large corpus of existing models and large corpus of SD literature). Basically I had intended to extend the automated support available for executing the usual SD development process.

But the world entered the Generative AI (Chat) epoch mid-stream of my work, after RQ1 and RQ2 and rendering RQ3 somewhat OBE. Early work (unpublished) on using GenAI to analyze SD models developed by my cohorts made it clear that something new and profound had happened: The GenAI does not simply let us accelerate what we already do. No, its internal architecture and huge knowledge base leads to results that are not simply human thought done faster. It was apparent that the GenAI can perform feats like determining quasi-quantitative behavior of a model without integrating the model, or even forming the mathematical model itself. This in turn suggested a new GenAI-centric MBST and our Pipeline Algebra (chapter 2), with which we exploit these new capabilities while maintaining epistemic and engineering rigor. I moved my focus to maximal exploitation of GenAI. The promise of the approach are apparent in AI Outperforms 43 grads developing CLD of Janis Groupthink (chapter 12), where the GenAI developed graduate-quality Causal Loop Diagrams in minutes rather than weeks; and in Behavior to Structure (chapter 16), where the GenAI determines likely model structure associated with a given behavior and builds a calibrated model to validate its results, but does not go though any sort of recognizable structure search process to get there.

At the start of our work, the research context was grounded in symbolic and numeric mathematics, alongside task-specific, locally executed, pre-generative neural networks applied within the domain of System Dynamics. However, the landscape has evolved rapidly: we are now in the epoch of agentic generative AI, marked by commercial solutions of unprecedented computational scale, incorporating large context memory, chain-of-thought reasoning, retrieval-augmented generation (RAG), and agentic capabilities. As a result, we have shifted our research context toward the use of large language models (LLMs), moving beyond traditional task-specific architectures. This shift is motivated by the recognition that commercially developed LLMs are advancing at a pace far exceeding the trajectory of our original technological assumptions. By aligning our work with these emerging capabilities, we increase the likelihood of delivering durable and meaningful value to the systems engineering community (See Future Work (section 19.3)).

Our thesis remains valid within this new research context, though some of the original research questions require subtle reinterpretation. In particular, Research Question 3 (RQ3) was initially intended to support the embedding of system dynamics model behavior represented as time-series behavior-over-time graphs (BOTGs). We performed such an embedding (unpublished, presented in SYSE532 guest lecture); however, in the majority of our work, these embeddings are realized as textual representations within the context window of a large language model (LLM), rather than as tokenized vectors within a latent vector space, as we had originally envisioned.

Research Question 5 (RQ5) was intended to tacitly imply the use of Genetic Algorithms in combination with structural "molecules" and a behavioral classification scheme to synthesize models relevant to the target domain. This approach aimed to align generated models with provided calibration data ("reference modes") and considered the potential use of text-based similarity search based on problem statements as well. However, we have since observed that LLMs are capable of inferring causal loop diagram (CLD) behavior from structure and vice versa [Behavior to Structure (chapter 16)]. Accordingly, we now propose an LLM-based approach to address RQ5, leveraging these generative and inferential capabilities. The similarity search of a curated corpus of SD models, updated with LLM-based structure and behavior mining of the SD models for textual descriptions to be embedded and searched, may still be interesting.

Our Model of Knowledge evolved alongside the research context. In the initial mathematical context, we intended to use mathematics itself as the model of knowledge, capturing both mechanisms and results. During our early work with large language models (LLMs), particularly in [AI Outperforms 43 grads developing CLD of Janis Groupthink (chapter 12)], we observed that the LLM era increases the value of intermediate representations. These representations improve token density in interactions with the LLM and help formalize and make our processes reproducible.

This realization led to the development of two key formalisms: Pipeline Algebra (chapter 2) (PA) and the mathematical Attributed Directed Graph (ADG) (chapter 7) (ADG). PA formalizes our interactions with LLMs and other algorithms, making those interactions epistemic and idempotent (hence reproducible). It is serializable and uses only minimal recursion, which allows LLMs to manipulate it effectively for meta-algorithmic operations on pipelines. This is an area identified for future exploration.

ADG provides a mathematical structure for exchanging Causal Loop Diagram (chapter 6)s (CLDs) and system dynamics (SD) models with the LLM, and for manipulating those models using conventional mathematical tools. We apply the same structure to Toulmin argumentation, enabling LLM-friendly, token-dense textual representations that can also be processed and transformed algorithmically through ADG-based methods.

Use of the LLM introduces a residual risk of hallucination. To address this, we developed a method called Falsifiable Prompting [Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (chapter 15)] which treats the LLM as an oracle whose outputs are not inherently trustworthy. Rather than assuming correctness, we prompt the model to produce responses that can be evaluated as falsifiable claims. This approach reframes the risk of hallucination as a hypothesis-testing problem, enabling systematic validation of LLM-generated content.

We now summarize each RQ and map it into our work.

RQ 1: Automatic identification of polarity-reversal conditions. Loop polarity is a key aid to understanding System-Dynamics behavior, so best practice is to ensure that no polarity reversals exist. Simulation-based searches are seldom attempted because the state space is vast and offers no gradients. Instead we propose a symbolic procedure: for every causal link we compute the partial derivative $\partial y/\partial x$. If $\partial y/\partial x > 0$ the link has positive polarity; if $\partial y/\partial x < 0$ it has negative polarity; and if the sign can be both positive and negative the polarity is ambiguous. We then solve for the state conditions that make the derivative strictly positive and, separately, strictly negative. A link's polarity is unambiguous iff the logical condition for one of the two signs reduces to False. Our contribution is the new knowledge that, although this is "just math", it can actually be computed in practice and at scale using modern CAS and without specialized SD or math skills, as evidenced by our doing so on a large corpus of SD models.

RQ 2: Automatic detection and characterization of loop dominance. Loop dominance denotes the extent to which particular feedback loops govern a System Dynamics model's behavior at a given state. Analysts routinely interpret simulation results by identifying the loops that dominate each phase of a trajectory. Because feedback loops interact nonlinearly, their effects are not strictly separable, so dominance cannot be assigned unambiguously to any single loop. Existing approaches depend on heuristic methods to estimate loop influence. This work contributes two additional heuristics. First, loop dominance is quantified as the state-dependent symbolic loop gain $g_i(x)$; evaluating $g_i(x)$ across the state space delineates regions where each loop exerts the greatest leverage. Second, the proposed *Generalized Loop Sets* approximate a minimal basis of loops whose combined gains reproduce the model's behavior. These techniques expand the SE's toolset for interrogating complex feedback structures, though, like previous methods, they do not eliminate the fundamental ambiguity induced by nonlinear loop interactions. The methods highlight loops of interest with minimal human interpretation required, and so reduce the level of specialized behavior-reading skill required to perform this important activity.

RQ 3(a) Sees a compact basis that captures canonical reference-mode behaviors without requiring the analysts to have knowledge in low-level SD, ML, or AI formalisms. The goal is an embedding of System-Dynamics time-series data that is both concise and semantically meaningful to downstream algorithms. A discriminative neural network fed with such an embedding would require fewer parameters and reveal latent structure with higher semantic density. Tested candidates include uniform subsampling and feature-based summaries that log local minima, maxima, inflection points, and derivative zero crossings, all features long viewed as salient in characterizing behavior-over-time graphs. Although in subsequent work we turned to language-oriented embeddings for large language models (e.g., "S-curve," "damped oscillation"), the study nevertheless produced an embedding that correctly inferred model calibration from observed behavior. Full integration into an agentic LLM workflow remains future work. Our contribution is a locally trained ANN that maps behavior to model parameters that achieve that behavior.

RQ 3(b) Seeks a compact structural representation of the model without requiring the analysts to have knowledge in low-level SD, ML, or AI formalisms. Systems Dynamics models often informally considered to be constructed from motifs with somewhat constrained "glue" math between the elements. Classic molecules(Hines, 1996) and idiomatic CLD behaviors (J. Sterman, 2000, ch4) (Wolstenholme, 2003) offer human-vetted building blocks, but we speculate that an LLM may discover smaller or more token-efficient fragments that better match its internal representation and reasoning mechanisms. Automatically extracting such AI-friendly motifs from a large corpus of well-formed SD models could (i) enrich the motif library beyond the textbook set, (ii) give the LLM reusable patterns it can copy verbatim, and (iii) provide an additional structural–plausibility filter when the generated model fails to match any known motif. This may ultimately improve the ability of the LLM to generate sound SD models given a CLD.

RQ 4: Detect feedback loops absent from a model that could generate unintended consequences, without requiring deep behavior analysis skills on the part of the analyst. Specifically, we seek latent tacit parasitic loops that actors might exploit, perhaps for power or profit, thereby undermining policy effectiveness. Our contribution is to show that large language models can efficiently surface these high-risk omissions in socio-technical policy analyses.

RQ 5: Generate plausible System Dynamics models directly from a behavioral specification, letting domain experts guide intent rather than hand-craft equations. Building a competent model from reference-mode behavior and stakeholder input normally takes years of practice, because the analyst must align structure with behavior and SME expectations while applying accepted modeling idioms. We hypothesize that this expertise is encoded in published models and captured by large language models augmented with domain literature. Our contribution is to demonstrate that an LLM can (1) synthesize a causal-loop diagram from a problem statement and target behavior, and (2) translate that diagram into a stock-and-flow model that reproduces the specified behavior.

The major contributions of this dissertation recur across multiple chapters, often in different methodological and empirical contexts. The Contribution-to-section map (Table 1.1) provides a quick-reference map that links each contribution to the sections where it is developed or applied.

17.2 Practical Engineering Implications

The practical capabilities of my work are perhaps best suggested by AI Outperforms 43 grads developing CLD of Janis Groupthink (chapter 12), LLM-Powered, Expert-Refined Causal Loop Diagramming via Pipeline Algebra (chapter 13), Missing Loops (chapter 14), and Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (chapter 15). In each case we used our methods to accelerate real-world SE/SD activities. Informal (not backed by formal experiment) results suggest a schedule compression of perhaps 100:1 and a significant reduction in SD-specific knowledge and skills required to perform the task.

My work shown in Missing Loops (chapter 14) suggests another powerful practical application: Developing robust policies as climate denial move from denying the science to denying the ability of our economy to move from carbon base and consequent discrediting of economic policies that address climate change. Perhaps my work can also be used to perform independent policy analysis so we, the citizens, can detect policies designed to look good and to be subverted.

17.3 Limitations and Threats to Validity

Oracle. GenAI is basically an Oracle, always ready to give an answer but often inscrutable. We suggest Falsifiable prompting as one mitigation, but the problem is probably much deeper than that. For example, if there are multiple legitimate answers, falsifiable prompting would not detect a bias in the GenAI nudging us towards particular answers among the set of admissible solutions.

Provider-level bias or manipulation. The owners of the language model could skew responses to promote their own commercial products or political views, suppress unfavorable conclusions, or accommodate political directives. Such steering would undermine the neutrality of loop analyses and policy simulations. It is likely that the nudges could be difficult or even theoretically impossible to detect.

Supply-chain compromise. Malicious code injected into the OpenAI functions API, the VOC registry, or dependent libraries could alter results or leak proprietary information.

Training-data poisoning. Adversaries might seed public repositories with plausible but flawed System Dynamics models or papers. Future model retraining would then propagate systematic errors into loop analytics and simulator refinements, distorting results while appearing authoritative.

Prompt-injection and instruction hijacking. Unvetted text, such as scraped policy documents, may contain adversarial strings that cause the LLM to modify parameters, bypass validation checks, or exfiltrate intermediate results, all contrary to user intent.

Covert information leakage. Proprietary system descriptions sent to the LLM could be stored or inspected by insider threats at the provider. Model responses might embed steganographic payloads that reveal sensitive design details to outside listeners.

Denial-of-service economics. By manipulating spot prices or throttling API quotas, an adversary could render VOC dispatch financially or operationally infeasible during critical design windows, delaying analyses when timely insight is essential.

Rich get richer. The owners of a language model have access to unfiltered results (no guardrails, no withholding of personal information, no adherence to cultural norms, more compute time), leading to access to knowledge that is only likely to increase the fraction of wealth and power held by the few and the ability to bias guardrails to nudge users towards conclusions desired by the owner of the model.

18 Reflections

18.1 Evolution of my thought

When this project started in the autumn of 2022 I believed the main challenge was computational: if I could wrap the classic chores of System Dynamics in a layer of fast symbolic and numeric mathematics, then practitioners would finally gain the speed and scale that modern hardware affords. The plan was orderly, the roadmap clear, and large language models seemed at most a curiosity. What happened instead was an accelerating series of breakthroughs in generative AI that kept altering both the toolset and the framing of the work. Each pivot created an opportunity to test Pipeline Algebra in ways I had not imagined. The sequence below traces that unexpected journey.

1. *Starting point, autumn 2022* The initial vision centered on math-first automation. Pipeline Algebra (PA) would coordinate procedures for detecting polarity reversals, mapping dominant and missing loops, and translating causal-loop diagrams into executable models. A practitioner from the 1970s could have recognized every task; only the speed would have surprised them.

2. *First glimpses of chat-style AI, 2023* Early public chat interfaces were entertaining but unreliable. Hallucinations, missing citations, and unstable answers made them unsuitable for serious work. A small pilot study hinted at deeper potential: given only variable names and link information, the model produced a coherent structural analysis of a classmate's SD model, suggesting that generative text might surface insights unavailable to pattern-matching neural nets.

3. *April 2025 turning point: ChatGPT o3* The release of ChatGPT o3, with a larger context window and built-in tool calling, changed the risk–benefit calculation. In the

first afternoon of testing, the model identified feedback loops exploitable by bad actors and matched or outperformed graduate students on structured SD assignments when asked to work at specific proficiency levels. Generative AI had crossed the line from novelty to professional tool.

4. *Rise of agentic and chain-of-thought capabilities, mid-2025* Minor version upgrades soon added an explicit think–act–observe–reflect cycle and stable chain-of-thought traces. A demonstration shows the practical consequence: the model read the NASA *Psyche* mission failure report, proposed an SD representation, wrote and executed a simulator, calibrated parameters, spotted unexplained residuals, hypothesized missing managerial behaviors, and repeated the cycle, all within an hour of wall-clock time and with only minimal prompt iteration.

5. *The Future: Control inversion* Originally PA orchestrated external tools while the language model provided occasional assistance. Experience with missing-loop detection, rapid CLD-to-SD translation, and behavior-to-structure inference revealed a stronger pattern: allow the agentic LLM to plan the work, treat PA operators as callable tools, and let the PA type system verify intermediate results. Inverted control would place the chat interface at the centre of the workflow where it clearly now belongs.

6. *Reframing through Model-Based Systems Thinking* The LLM-centric pipeline aligns naturally with Model-Based Systems Thinking. A fast inner loop iterates structure, simulation, and comparison; a slower outer loop allows human experts to revise mental models and policy hypotheses. PA's typed morphisms keep every step explicit, repeatable, and auditable.

7. *Present perspective* The dissertation now presents PA, annotated directed graphs, falsifiable prompting, and the empirical studies as elements of an AI-accelerated SD toolkit. What began as a math-first automation exercise has become a testbed for human–AI

collaboration, showing that rigorous and scalable System Dynamics practice is within reach when generative models act as adaptable partners rather than passive libraries.

19 Conclusions

19.1 Theoretical Contributions

I feel my most significant theoretical contribution is the Pipeline Algebra (chapter 2). PA is a sound basis for bringing engineering and epistemic rigor to the use of GenAI applied to Systems Engineering. It is mathematically formal, has a practical execution model, projects onto DAG execution engines that scale to millions of cores, and provides a dense notation suitable for "notation for thought" reasoning and pedagogics as well as execution. It also forms the basis of much interesting Future Work (section 19.3) either on or built within the framework of PA.

Falsifiable Prompting may also prove to be a valuable contribution, as we move into an "Oracle" world where we can get answers to any question but we can't in general trust that the answer is correct. There is much evidence that our cognitive heuristics do not serve us well in our built digital environment. Maybe the notion of falsifiable prompting can improve the situation.

Other contributions are enumerated here:

- Pipeline Algebra (chapter 2) (PA) : a typed, category-theoretic workflow calculus
  - Executes "notation for thought" with explicit pre- and post-conditions, morphism types, and deterministic replay.
  - Extends monadic composition to mixed human/LLM/code pipelines, giving a provenance-preserving basis for agentic AI workflows.
- Attributed Directed Graph (ADG) (chapter 7) schema for causal-loop diagrams and Systems Dynamics models

- Seed for the investigation of data structures equally amenable for GenAI, algorithmic, and human comprehension and manipulation.
- Embeds relational and semantic attributes in CLDs, aligning them with knowledge-graph formalism.
- Supplies the typing information that PA uses for condition checks on LLM outputs.

- Closed-form analytic tests for Analytical Conditions of Loop Polarity Reversal (chapter 9)
  - Derives symbolic ∂-based conditions for sign flips and dominance shifts.
  - Provides the mathematical core of the polarity-reversal and missing-loop operators.
  - Our interesting contribution is that this can actually be done at scale.
- Generalized Loop Sets (chapter 11) (GLS)
  - Enumerates all minimal loop sets whose combined influence equals the net feedback of the full Jacobian.
  - Gives a polynomial-time algorithm for detecting dominant loop families in any stock-and-flow digraph.
  - The first exact algorithm in the class of loop set algorithms.
- Falsifiable prompting and simulator-in-the-loop agentic refinement
  - Casts chain-of-thought prompts as hypothesis generation and SD simulation as hypothesis testing.
  - Provides termination conditions for the refinement cycle under bounded error and step size.
  - Mitigates Hallucinations

– See Rapid CLD-to-SD Translation Using Falsifiable Prompting with Chat-Based Generative AI (chapter 15)

19.2 Broader Impact

Our work enables the rapid, widespread deployment of analytical pipelines to scrutinize policy and detect intentional and accidental flaws that may lead to outcomes other than the one stated for the policy. Once a draft policy text is made public, PA pipelines can launch parallel scans to find structural motifs that suggest regulatory loopholes or coordinated influence campaigns. Our work sets a clear path toward letting citizens, policy teams, and engineering groups test policies, make evidence-based choices, and meet their responsibility to steward the shared systems on which we all depend.

Our methods are applicable to any system that exhibits complex feedback behaviors, of course. Our specific intent is to facilitate the application of these methods by a broader community to the large-scale challenges we face today.

19.3 Future Work

Advancing this work requires an *LLM-centric inversion of control* in which the language model itself becomes the orchestration framework, not merely a callable library. Recent agentic capabilities, in particular function calling, tool selection, and reflective planning, enable GPT to decide when to invoke analytic operators, manage execution state, and interpret intermediate results. To make that practical at scale, we propose *Vectorized Operator Capsules* (VOCs), a remote-evaluation construct that packages an operator together with its partitioning strategy, resource hints, and idempotence contract. The LLM issues a single function call describing the VOC, and the Pipeline Algebra runtime dispatches the workload across cluster back ends such as AWS Batch, returning structured summaries rather than millions of raw results. By elevating control flow from bespoke glue code to an agentic framework, VOCs transform PA from a library of operators into a framework that the LLM can reason about, schedule, and optimize in real time.

Future work may investigate genetic optimization of System Dynamics models in which a language model accelerates purifying selection. Instead of initializing the population with random chromosomes, the LLM will seed candidate models using high-quality loop motifs and equation fragments extracted from the curated SD corpus and LLM knowledge base. During each generation the model will preview offspring in natural language, discarding those that violate polarity constraints, omit required feedback motifs, or duplicate structures already dominant in the population, or otherwise appear unlikely to yield useful result, thereby saving costly simulation cycles. The numerical simulator will still provide the primary fitness signal based on error against a reference behavior, but the LLM will supply a complementary semantic score that reflects structural plausibility and alignment with published literature. By combining behavioral error and LLM-derived plausibility in a hybrid fitness function, the search process can avoid infeasible regions of the design space and converge more quickly on interpretable, structurally coherent solutions than blind genetic algorithms achieve.

We investigated (not published herein) several forms of loop dominance: Dominance defined by loop gain; Dominance defined by chirp response; Dominance defined by consequence of breaking the subject loop, and there are numerous other methods in the literature. It is likely there is no one true definition of loop dominance, and no definition without significant theoretical issues stemming from non-linear loop entanglements. Indeed, we argue this is signature property of complex feedback. But what if we applied all of the methods, and challenged the LLM to create a dominance narrative that explains the results? It seems likely to us that significant new insights into dominance would emerge from this possible future work.

A complementary line of future work is "alien-intelligence" motif discovery, in which the language model itself becomes the source of feedback patterns that feel native to its internal representations rather than to human intuition. By issuing probing prompts such as "enumerate the minimal feedback archetypes that produce logistic growth" and by sampling chain-of-thought traces during causal reasoning tasks, one might mine the LLM for subgraph templates that it repeatedly invokes. These machine-preferred motifs will then be formalized in ADG form. They will act as atomic genes in the genetic optimization pipeline, allowing the LLM to seed or mutate candidate models by recombining motifs it already understands deeply, thereby accelerating convergence toward structurally plausible solutions. In effect, we will be teaching the workbench to speak the LLM's native dialect of feedback patterns, gaining efficiency and expressiveness that traditional, human-centric motifs may not match.

A possible direction for future work is to examine how large language models interact with the classical limits of computation expressed by Gödel, Turing, and Church. The Church–Turing thesis asserts that every effectively calculable procedure is Turing computable, yet language models sometimes deliver plausible solutions or explanations for problems that are formally undecidable or computationally intractable. Exploring whether such performance represents genuine algorithmic leverage, sophisticated pattern matching over training data, or a redefinition of success toward approximate answers could shed light on whether these models truly skirt traditional complexity barriers or merely create that impression. Clarifying this relationship would help set realistic expectations for the scope and reliability of LLM-driven operators in future versions of Pipeline Algebra.

Future work could investigate a comprehensive threat-mitigation framework for LLM-driven modeling pipelines. Central questions include how to tame the "oracle effect," where models deliver authoritative yet inscrutable answers, and how to detect or neutralize provider-level bias, supply-chain compromise, training-data poisoning, prompt-injection attacks, covert information leakage, automated policy manipulation, denial-of-service economics, and widening knowledge asymmetries that favor model owners. Research avenues might combine falsifiable prompting with cryptographic attestation of operator binaries, reproducible model fingerprints, differential-comparison audits across multiple providers, and provenance-aware retrieval that flags suspect sources before they influence loop analyses.

Additional studies could model the economic incentives behind API throttling or bias insertion, then test counter-strategies such as cooperative federation of smaller open-weight models and cost-aware scheduling that avoids single-provider lock-in. Finally, policy and governance research is needed to explore data-diversity mandates, audit rights, and disclosure requirements that limit the "rich get richer" dynamic while still encouraging innovation.

Taken together, the future work is to bring in a NeoSD era: It's time to find our way beyond tools and approaches that have changed but incrementally in nearly a half a century.

19.4 Closing Synthesis

System-dynamics modeling provides an indispensable lens for understanding feedback-rich challenges, yet its traditional workflow is fundamentally misaligned with the compressed timelines of modern governance, especially in the climate domain. The process of constructing and calibrating a model can consume weeks of specialist effort, while critical policy windows often last only for days. The advent of frontier large language models resolves this temporal mismatch by accelerating analytic cycles, but this same speed creates a new, dangerous asymmetry. The tools that allow engineers and citizens to design robust climate solutions can be weaponized with equal or greater speed by well-funded actors to refine disinformation and engineer policy loopholes. This acceleration forces the system's boundary to expand beyond the purely socio-technical into the realm of contested governance, creating a "compute gap" that is the central challenge to leveraging these new technologies for the public good.

This emerging compute gap creates a stark power imbalance, as the analytical speed is not evenly distributed. Cloud platforms, hedge funds, state intelligence units, tech oligarchs, and political consultancies can afford to dedicate massive datacenter-hours of LLMs with no guardrails to tasks such as crafting legislative loopholes or refining disinformation campaigns that exploit voter blocs. In contrast, citizens, journalists, and smaller agencies are typically confined to rate-controlled public endpoints, granting them only microseconds of compute time and with always-active guardrails. Frontier models, therefore, risk widening the analytical divide between a few powerful actors and the broader community they affect.

To counter this gap, we coin the term "NeoSD" for a chat-centric framework that leverages a globally shared cache to implement dynamic programming at a collaborative scale. Built upon Pipeline Algebra and agentic tool-calling, NeoSD would be more than just classic System Dynamics in an inverted chat interface; it adds a quasi-quantitative layer where a large language model can reason about feedback structure without requiring a fully specified model. Crucially, because every call within Pipeline Algebra is logged, a query made by any user can be stored in a cloud-based cache and its results returned instantly to any other user who makes the same query. This allows many small users to effectively pool their micro-compute resources and benefit from the expensive computations previously run by others, directly turning shared traces into a shared analytic backbone.

This democratized framework will enable the rapid, widespread deployment of analytical pipelines to scrutinize policy and detect manipulation. Once a draft policy text is made public, lightweight NeoSD pipelines can launch parallel scans to find structural motifs that suggest regulatory loopholes or coordinated influence campaigns. Because shared prompts and typed functions can propagate quickly, the community's defensive toolkit can evolve in response to new attack vectors. While this pooled micro-compute cannot prevent every exploit, it restores meaningful analytic capacity to the public and shrinks the window in which powerful actors can operate without scrutiny.

Realizing this vision for NeoSD, however, will require dedicated and open collaboration among diverse stakeholders. Researchers must commit to releasing validated functions as they are developed, while legislatures and agencies should publish machine-readable drafts of policies early to allow for timely review. Journalists and civil-society groups could integrate NeoSD-style checks into their standard workflows, and cloud providers can support this effort by offering civic-rate tiers for the necessary micro-compute. If these steps, accompanied by strong ethical guidelines and audit trails, are taken, the same frontier technologies that risk empowering adversaries can also strengthen our collective ability as analysts and citizens to explore policy options, detect hidden exploits, and make fact-based decisions that guide society toward our most desired outcomes.

20 Glossary

ADG — Attributed Directed Graph. An ADG extends a conventional directed graph by assigning attributes to vertices, edges, and the graph as a whole.

AI — Artificial Intelligence. A branch of computer science dedicated to creating systems capable of performing tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and learning.

AI Agent — A system that can perceive its environment, make decisions, and take autonomous actions to achieve specific goals. The dissertation describes its operation as an "agentic loop" of Think → Act → Observe → Reflect, often powered by a Large Language Model.

ANN — Artificial Neural Network. A computational model inspired by the structure of biological neural networks, consisting of interconnected nodes organized in layers. ANNs are used for tasks such as pattern recognition and machine learning.

AugLLM — Augmented Large Language Models. See also: **AI Agent**.

BOTG — Behavior Over Time Graphs. These are time-series plots used to depict variable trajectories over time and identify dynamic patterns.

CAISO — California Independent System Operator.

CAS — Computer Algebra System. A software program that facilitates symbolic mathematics, allowing for the manipulation of mathematical expressions in their symbolic form as opposed to numerical computation.

ChatGPT — A specific commercial, chat-based large language model from OpenAI used in the dissertation's studies.

CILS — Complete Independent Loop Sets. A method used in loop dominance analysis.

CLD — Causal Loop Diagram. A conceptual diagram of variables connected by arrows that form feedback loops representing the structure of a dynamic system.

CoT — Chain-of-Thought. (a) A prompting technique where the AI model writes out its intermediate steps to improve reasoning quality. See also: **AI Agent**. (b) Core, trained capability of the model to generate a sequence of private, intermediate reasoning steps before producing a final answer

DAE — Differential-Algebraic Equation. A system of equations that contains both differential equations and algebraic equations, often used to model complex dynamical systems where variables are subject to explicit constraints.

DAG — Directed Acyclic Graph. A type of directed graph that contains no directed cycles, commonly used in computer science to model processes and data dependencies where tasks must be completed in a specific order.

EPC — Engineering, Procurement, and Construction.

Falsifiable Prompting — A protocol developed in the dissertation where a large language model is treated as an untrusted oracle whose responses must be independently verified or refuted through a series of tests.

FSW — Flight Software.

FTE — Full-Time Equivalent.

GILS — Generalized Independent Loop Sets. An algorithm that improves upon other loop set methods for loop dominance analysis.

GNC — Guidance, Navigation, and Control.

ILP — Integer Linear Programming. A mathematical optimization program where the objective function and constraints are linear and where some or all of the variables are restricted to be integers.

IMRaD — Introduction, Methods, Results, and Discussion. A standard structure for organizing a scientific article.

IRA — Inflation Reduction Act.

IRB — Independent Review Board.

JSON — JavaScript Object Notation. A lightweight, text-based data-interchange format that is easy for humans to read and write and for machines to parse and generate.

JPL — Jet Propulsion Laboratory.

KILS — Kampmann Independent Loop Set. A method used to enhance model analysis by isolating loop subsets.

L-BFGS-B — Limited-memory Broyden–Fletcher–Goldfarb–Shanno with bounds. A specific optimization algorithm used for model calibration.

LCOE — Levelized Cost of Energy. A measure of the average net present cost of electricity generation for a generator over its lifetime.

LLM — Large Language Model. A large, general-purpose generative language model pretrained on a vast corpus of text. See also: **AI Agent**.

MAPE — Mean Absolute Percentage Error.

MBST — Model-Based Systems Thinking. A structured approach that employs modeling techniques to enhance the comprehension of complex issues.

MCMC — Markov Chain Monte Carlo. A class of algorithms used for sampling from a probability distribution, particularly useful when direct sampling is difficult.

MPA — Morphological Pipeline Algebra. Aka **Pipeline Algebra**.

NeoSD — A term coined in the dissertation for a proposed chat-centric System Dynamics framework that uses a globally shared cache to democratize analytical capabilities.

NIMBY — Not In My Backyard.

OBE — Overcome By Events. A colloquialism used to describe methods that were rendered obsolete by the rapid evolution of technology.

OECD — Organization for Economic Co-operation and Development.

OPM — Object-Process-Methodology. A formal modeling language mentioned in the discussion of Model-Based Systems Thinking.

PA — Pipeline Algebra. A category-theoretic workflow language where every stage is a typed morphism. It is designed to be a formal, executable, and reproducible "notation for thought". Sometimes called Morphological Pipeline Algebra (MPA).

PFR — Problem/Failure Reports.

PTC — Production Tax Credit.

PV — Photovoltaics.

ReAct — Reasoning and Acting. A framework that formalizes an agentic loop into a Think $\rightarrow$ Act $\rightarrow$ Observe $\rightarrow$ Reflect pattern.

RMSE — Root-Mean-Square Error.

RP — Research Problem. The central research problem of the dissertation.

SCADA — Supervisory Control and Data Acquisition. A system of software and hardware elements that allows industrial organizations to control industrial processes locally or at remote locations and monitor, gather, and process real-time data.

SD — System Dynamics. A methodology to reason about feedback-rich problems by modeling and simulating complex systems.

SE — Systems Engineers.

SGN — Sign Function. A mathematical function that extracts the sign of a real number, returning -1 for negative numbers, 0 for zero, and 1 for positive numbers.

SILS — Shortest Independent Loop Set. A method used in loop dominance analysis.

SME — Subject-Matter Expert. An expert who provides contextual judgment and domain heuristics in the modeling process.

SOI — System-of-Interest. The system being modeled and analyzed.

ST — Systems Thinking. A discipline for comprehending wholes, interrelationships, and dynamic patterns rather than isolated components.

TALM — Tool-Augmented Language Model. An LLM that is paired with external tools and utilities. See also: **AI Agent**.

Theil U2 — A relative accuracy statistic that compares the forecasted results with the results of a naive "no-change" forecast. A value less than 1 indicates the forecast is better than guessing, while a value greater than 1 indicates it is worse.

VOC — Vectorized Operator Capsules. A remote-evaluation construct proposed for future work to package operators for scalable execution.

WACC — Weighted Average Cost of Capital. A financial calculation of a company's average cost of capital from all sources, including stock, bonds, and other forms of debt, weighted by market value.

References

Abdelbari, H., Elsawah, S., & Shafi, K. (2015). Model learning using genetic programming under full and partial system information conditions. https://proceedings.systemdynamics.org/2015/proceed/papers/P1162.pdf

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). *Compilers, principles, techniques, and tools*. Addison-Wesley Pub. Co.

Amazon Web, S. (n.d.). Aws step functions developer guide. https://docs.aws.amazon.com/step-functions/

Amazon Web, S. (2025a). Aws batch. https://aws.amazon.com/batch/

Amazon Web, S. (2025b). Aws step functions. https://aws.amazon.com/step-functions/

Andersen, D. F., Cappelli, D., Gonzalez, J. J., Mojtahedzadeh, M., Moore, A., Rich, E., Sarriegui, J. M., Shimeall, T. J., Stanton, J., & Weaver, E. (n.d.). Preliminary system dynamics maps of the insider cyber-threat problem. *Proceedings of the 22nd International Conference of the System dynamics Society*, 25–29.

ApS, M. (2019). Mosek optimization suite.

Argyris, C. (1977). Double loop learning in organizations. *Harvard business review*, *55*(5), 115–125.

Arndt, H. (2023). Ai and education: An investigation into the use of chatgpt for systems thinking. *arXiv preprint arXiv:2307.14206*.

Barbrook-Johnson, P., & Penn, A. S. (2022). Causal loop diagrams. In *Systems mapping: How to build and use causal models of systems* (pp. 47–59). Springer International Publishing. https://doi.org/10.1007/978-3-031-01919-7_4

Barlas, Y. (1996). Formal aspects of model validity and validation in system dynamics. *System Dynamics Review*, *12*(3), 183–210. https://doi.org/https://doi.org/10.1002/(SICI)1099-1727(199623)12:3<183::AID-SDR103>3.0.CO;2-4

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., & Koc, L. (n.d.). Tfx: A tensorflow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 1387–1395.

Black, S., Liu, A. A., Parry, I., & Vernon, N. (2023). *Imf fossil fuel subsidies data: 2023 update* (Report). International Monetary Fund. https://www.imf.org/en/Publications/WP/Issues/2023/08/22/IMF-Fossil-Fuel-Subsidies-Data-2023-Update-537281

Boardman, J., & Sauser, B. (2008). *Systems thinking: Coping with 21st century problems*. CRC Press.

Brandes, U., Eiglsperger, M., Lerner, J., & Pich, C. (2013). Graph markup language (graphml). *Handbook of Graph Drawing and Visualization*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, *113*(15), 3932–3937.

Buzan, T., & Buzan, B. (1993). *The mind map book*. BBC Books.

Cecchetti, S. G., Mohanty, M. S., & Zampolli, F. (2011). *The real effects of debt* (Report). Bank for International Settlements. https://www.bis.org/publ/work352.pdf

Checkland, P. (1981). Systems thinking, systems practice.

Chiland, E. (2019, 2019/05/06). Five years after sepulveda pass widening, travel times on the 405 keep getting worse. https://la.curbed.com/2019/5/6/18531505/405-widening-traffic-los-angeles-carpool-lane

Climate change 2023: Synthesis report. contribution of working groups i, ii and iii to the sixth assessment report of the intergovernmental panel on climate change (Report). (2023).

Intergovernmental Panel on Climate Change. https://doi.org/10.59327/IPCC/AR6-9789291691647

Collenteur, R., van der Moel, H., Jongman, B., & Di Baldassarre, G. (2015). The failed-levee effect: Do societies learn from flood disasters? *Natural Hazards*, *76*(1), 373–388. https://doi.org/10.1007/s11069-014-1496-6

Collins, G. S., Reitsma, J. B., Altman, D. G., & Moons, K. G. (n.d.). Transparent reporting of a multivariable prediction model for individual prognosis or diagnosis (tripod): The tripod statement. (1756-1833 (Electronic)).

de Vries-Gao, A. (n.d.). Artificial intelligence: Supply chain constraints and energy implications. *Joule*. https://doi.org/10.1016/j.joule.2025.101961

Eilenberg, S., & MacLane, S. (1945). General theory of natural equivalences. *Transactions of the American Mathematical Society*, *58*(2), 231–294.

Feldman, S. I. (1979). Make a program for maintaining computer programs. *Software: Practice and Experience*, *9*(4), 255–265. https://doi.org/https://doi.org/10.1002/spe.4380090402

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (n.d.). Efficient and robust automated machine learning. *Neural Information Processing Systems*.

Forrester, J. W. (1961). *Industrial dynamics*. M.I.T. Press.

Forrester, J. W. (1971). Counterintuitive behavior of social systems. *Theory and decision*, *2*(2), 109–140.

Forrester, J. W. (2007). System dynamicsthe next fifty years. *System Dynamics Review*, *23*(2-3), 359–370. https://doi.org/https://doi.org/10.1002/sdr.381

Forrester, J. (2004). Mit system dynamics group literature collection dvd. *System Dynamics Society, Albany, NY, USA*.

Ghaffarzadegan, N., Lyneis, J., & Richardson, G. P. (2011). How small system dynamics models can help the public policy process. *System Dynamics Review*, *27*(1), 22–44. https://doi.org/https://doi.org/10.1002/sdr.442

Gilbreth, F. B., & Gilbreth, L. M. (1921). Process charts: First steps in finding the one best way to do work. *Journal of Fluids Engineering*, *43*, 1029–1043.

Glick, M., & Duhon, T. (1994). Generic structures: S-shaped growth i. *Massachussets Institute of Technology*.

Goodman, M. (1973). *Causal loop diagramming (d-1755-2)* (Report). MIT.

Goodman, M. R. (1972). *Elementary system dynamics structures* [Thesis].

Goodman, M. R. (1974). *Study notes in system dynamics*. Wright-Allen Press.

Griva, I., Nash, S., & Sofer, A. (2009). *Linear and nonlinear optimization* (2nd). Society for Industrial; Applied Mathematics.

Gupta, A., Zuckerman, E., & O'Connor, B. T. (n.d.). Harnessing toulmin's theory for zero-shot argument explication. *Annual Meeting of the Association for Computational Linguistics*.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, *8*(3), 231–274.

Hayward, J., & Roach, P. A. (2024). A comparison of loop dominance methods: Measures and meaning. *System Dynamics Review*, *40*(2), e1757. https://doi.org/https://doi.org/10.1002/sdr.1757

Heil, B. J., Hoffman, M. M., Markowetz, F., Lee, S.-I., Greene, C. S., & Hicks, S. C. (2021). Reproducibility standards for machine learning in the life sciences. *Nature Methods*, *18*(10), 1132–1135. https://doi.org/10.1038/s41592-021-01256-7

Hines, J., Malone, T., Gonçalves, P., Herman, G., Quimby, J., Murphy-Hoye, M., Rice, J., Patten, J., & Ishii, H. (2011). Construction by replacement: A new approach to simulation modeling. *System Dynamics Review*, *27*(1), 64–90. https://doi.org/https://doi.org/10.1002/sdr.437

Hines, J. (1996). Molecules of structure. *System Dynamics Group, Sloan School of Management, MIT*. http://www.mindseyecomputing.com/molecule.pdf

Hirsch, A. V., & Kastellec, J. P. (2022). A theory of policy sabotage. *Journal of Theoretical Politics*, *34*(2), 191–218. https://doi.org/10.1177/09516298221085974

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2019). The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Huang, J., Howley, E., & Duggan, J. (2012). Observations on the shortest independent loop set algorithm. *System Dynamics Review*, *28*(3), 276–280. https://doi.org/https://doi.org/10.1002/sdr.1477

Hughes, J. (1989). Why functional programming matters. *The Computer Journal*, *32*(2), 98–107. https://doi.org/10.1093/comjnl/32.2.98

Hung, W. (2008). Enhancing systems-thinking skills with modelling. *British Journal of Educational Technology*, *39*(6), 1099–1120. https://doi.org/https://doi.org/10.1111/j.1467-8535.2007.00791.x

International Committee of Medical Journal, E. (2025). Preparing a manuscript for submission to a medical journal. https://www.icmje.org/recommendations/browse/manuscript-preparation/preparing-for-submission.html

Ishikawa, K., & Loftus, J. H. (1990). *Introduction to quality control* (Vol. 98). Springer.

Iverson, K. E. (1980). Notation as a tool of thought. *Commun. ACM*, *23*(8), 444–465. https://doi.org/10.1145/358896.358899

Jacob, C. (2001). *Illustrating evolutionary computation with mathematica*. Morgan Kaufmann Pub.

Janis, I. L. (1982). *Groupthink : Psychological studies of policy decisions and fiascoes* (2nd). Houghton Mifflin.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 1–38.

Juran, J. M. (1951). *Quality-control handbook* (1st). McGraw-Hill.

Kampmann, C. E. (2012). Feedback loop gains and system behavior (1996). *System Dynamics Review*, *28*(4), 370–395. https://doi.org/https://doi.org/10.1002/sdr.1483

Kim, D. H. (1992). *Systems archetypes i*. Pegasus Communications.

Kim, D. H. (1995). *Systems thinking tools: A user's reference guide*. Pegasus Communications.

Kim, D. H. (1999). *Introduction to systems thinking*.

Kim, D. H., & Senge, P. M. (1994). Putting systems thinking into practice. *System Dynamics Review*, *10*(2-3), 277–290. https://doi.org/https://doi.org/10.1002/sdr.4260100213

Kohavi, R. (1998). Glossary of terms (incl confusion matrix - kirk). *Machine learning*, *30*, 271–274.

Lane, D. C. (2008). The emergence and use of diagramming in system dynamics: A critical account. *Systems Research and Behavioral Science*, *25*(1), 3–23. https://doi.org/https://doi.org/10.1002/sres.826

Lavi, R., Dori, Y. J., Wengrowicz, N., & Dori, D. (2020). Model-based systems thinking: Assessing engineering student teams. *IEEE Transactions on Education*, *63*(1), 39–47. https://doi.org/10.1109/TE.2019.2948807

Lawrence, S., Herber, D. R., & Shahroudi, K. E. (2025). Leveraging system dynamics to predict the commercialization success of emerging energy technologies: Lessons from wind energy. *Energies*, *18*(8), 2048. https://www.mdpi.com/1996-1073/18/8/2048

Lee, E. A., & Parks, T. M. (1995). Dataflow process networks. *Proceedings of the IEEE*, *83*(5), 773–801. https://doi.org/10.1109/5.381846

Liu, N.-Y. G., & Keith, D. R. (2025). Leveraging large language models for automated causal loop diagram generation: Enhancing system dynamics modeling through curated prompting techniques. *ArXiv*, *abs/2503.21798*.

Llor, C., & Bjerrum, L. (2014). Antimicrobial resistance: Risk associated with antibiotic overuse and initiatives to reduce the problem. *Therapeutic Advances in Drug Safety*, *5*(6), 229–241. https://doi.org/10.1177/2042098614554919

Mac Lane, S. (1998). *Categories for the working mathematician* (2nd). Springer.

Maruyama, M. (1963). The second cybernetics: Deviation-amplifying mutual causal processes. *American Scientist*, *51*(2), 164–179. http://www.jstor.org/stable/27838689

Matheson, J., & Howard, R. (1984). Influence diagrams. *The Principles and Applications of Decision Analysis; Strategic Decisions Group, Menlo Park, California, USA*, *2*.

Meadows, D. (1999). Leverage points: Places to intervene in a system. *The Sustainability Institute*. https://donellameadows.org/archives/leverage-points-places-to-intervene-in-a-system/

Meadows, D. H., & Wright, D. (2008). *Thinking in systems : A primer*. Chelsea Green Pub.

Messeri, L., & Crockett, M. (2024). Artificial intelligence and illusions of understanding in scientific research. *Nature*, *627*(8002), 49–58.

Michie, D. (1968). memo functions and machine learning. *Nature*, *218*(5136), 19–22. https://doi.org/10.1038/218019a0

Middleton, S., & Motamed, M. (2024). A complete shortest independent loop set algorithm for model structure analysis. *System Dynamics Review*, *40*(3), e1787. https://doi.org/https://doi.org/10.1002/sdr.1787

Milewski, B. (2019). *Category theory for programmers*. Bartosz Milewski.

Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, *93*(1), 55–92. https://doi.org/https://doi.org/10.1016/0890-5401(91)90052-4

Mojtahedzadeh, M. (2011). Consistency in explaining model behavior based on its feedback structure. *System Dynamics Review*, *27*(4), 358–373. https://doi.org/https://doi.org/10.1002/sdr.468

Mojtahedzadeh, M. (n.d.). Do the parallel lines meet? a comparison between pathway participation metrics and eigenvalue analysis. *Proceedings of the 2007 International System Dynamics Conference, System Dynamics Society, Boston, MA*, *7*.

Monat, J., Gannon, T., & Amissah, M. (2022). The case for systems thinking in undergraduate engineering education. *International Journal of Engineering Pedagogy (iJEP)*, *12*(3), 50–88.

Monat, J. P., & Gannon, T. F. (2015). What is systems thinking? a review of selected literature plus recommendations. *American Journal of Systems Science*, *4*(1), 11–26.

Morrison, J. P. (n.d.). Flow-based programming. *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*, 25–29.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., & Jaiswal, S. (2017). Graph2vec: Learning distributed representations of graphs. *ArXiv*, *abs/1707.05005*.

National, A., & Space, A. (2022). Psyche independent review board report and nasa response (november 4, 2022). https://www.nasa.gov/wp-content/uploads/2022/11/psyche_irb_report_and_response_nov_2022.pdf

Nix, A., Decker, S., & Wolf, C. (2021). Enron and the california energy crisis: The role of networks in enabling organizational corruption. *Business History Review*, *95*(4), 765–802. https://doi.org/10.1017/S0007680521001008

North, M., Sydelko, P., & Martinez-Moyano, I. (2015). Structurally evolving system dynamics models using genetic algorithms. https://proceedings.systemdynamics.org/2015/proceed/papers/P1308.pdf

Novak, J. D. (2010). *Learning, creating, and using knowledge : Concept maps as facilitative tools in schools and corporations* (2nd). Routledge.

OECD. (2017). Systems approaches to public sector challenges: Working with change.

Oliva, R. (2004). Model structure analysis through graph theory: Partition heuristics and feedback structure decomposition. *System Dynamics Review*, *20*(4), 313–336. https://doi.org/https://doi.org/10.1002/sdr.298

OpenAi. (2023). *Gpt-4 technical report* (Report). OpenAI. https://openai.com/research/gpt-4

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., & Ray, A. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, *35*, 27730–27744.

Palantir, T. (2025). Palantir foundry. https://www.palantir.com/platforms/foundry/

Park, W.-W. (2000). A comprehensive empirical investigation of the relationships among variables of the groupthink model. *Journal of Organizational Behavior*, *21*(8), 873–887. https://doi.org/https://doi.org/10.1002/1099-1379(200012)21:8<873::AID-JOB56>3.0.CO;2-8

Pearl, J. (2000). *Causality : Models, reasoning, and inference*. Cambridge University Press.

Pearl, J. (2005). Influence diagramshistorical and personal perspectives. *Decision Analysis*, *2*(4), 232–234.

Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Elsevier.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, *334*(6060), 1226–1227. https://doi.org/10.1126/science.1213847

Pourbafrani, M., & Aalst, W. (2022). Discovering system dynamics simulation models using process mining. *IEEE Access*, *10*, 1–1. https://doi.org/10.1109/ACCESS.2022.3193507

Rahmandad, H., & Sterman, J. D. (2012). Reporting guidelines for simulation-based research in social sciences. *System Dynamics Review*, *28*(4), 396–411. https://doi.org/https://doi.org/10.1002/sdr.1481

Rahmandad, H., Xu, R., & Ghaffarzadegan, N. (2022). A missing behavioural feedback in covid-19 models is the key to several puzzles. *BMJ global health*, *7*(10), e010463.

Renze, J. (2020). Image. *From Mathworld - A Wolfram Web Resource, created by Eric W. Wesstein*. https://mathworld.wolfram.com/Image.html

Richardson, G. P. (1986a). Dominant structure. *System Dynamics Review*, *2*(1), 68–75. https://doi.org/https://doi.org/10.1002/sdr.4260020107

Richardson, G. P. (1986b). Problems with causal-loop diagrams. *System Dynamics Review*, *2*(2), 158–170. https://doi.org/https://doi.org/10.1002/sdr.4260020207

Richardson, G. P. (1991). *Feedback thought in social science and systems theory*. University of Pennsylvania.

Richardson, G. P. (1995). Loop polarity, loop dominance, and the concept of dominant polarity (1984). *System Dynamics Review*, *11*(1), 67–88. https://doi.org/https://doi.org/10.1002/sdr.4260110106

Richardson, G. P. (1997). Problems in causal loop diagrams revisited. *System Dynamics Review*, *13*(3), 247–252. https://doi.org/https://doi.org/10.1002/(SICI)1099-1727(199723)13:3<247::AID-SDR128>3.0.CO;2-9

Richardson, G. P. (2011). Reflections on the foundations of system dynamics. *System Dynamics Review*, *27*(3), 219–243. https://doi.org/https://doi.org/10.1002/sdr.462

Richardson, G. P., & Pugh, A. L. (1981). *Introduction to system dynamics modeling with dynamo*. MIT Press.

Schafer, M., & Crichlow, S. (1996). Antecedents of groupthink:a quantitative study. *Journal of Conflict Resolution*, *40*(3), 415–435. https://doi.org/10.1177/0022002796040003002

Senge, P. M. (1990). *The fifth discipline : The art and practice of the learning organization* (1st). Doubleday/Currency.

Senge, P. M. (2006). *The fifth discipline : The art and practice of the learning organization* (Rev. and updated.). Doubleday/Currency.

Shahroudi, K. E., Conrad, S., Speece, J., Reinholtz, K., Span, M. T., Chappell, S., Saulter, Q., & Bokhtier, G. M. (2025). *Practical systems thinking*. Springer International Publishing. https://doi.org/10.1007/978-3-031-89394-0

Simske, S. J. (2013). Meta-algorithmics : Patterns for robust, low cost, high quality systems. Cover%20image%20http://catalogimages.wiley.com/images/db/jimages/9781118343364.jpg

Spivak, D. I. (2014). *Category theory for the sciences*. MIT press.

Sterman, J. (2000). *Business dynamics : Systems thinking and modeling for a complex world*. Irwin/McGraw-Hill.

Sterman, J. (2018). System dynamics at sixty: The path forward. *System Dynamics Review*, *34*(1-2), 5–47. https://doi.org/https://doi.org/10.1002/sdr.1601

Sterman, J. D. (1994). Learning in and about complex systems. *System Dynamics Review*, *10*(2-3), 291–330. https://doi.org/https://doi.org/10.1002/sdr.4260100214

Stigler, G. J. (1971). The theory of economic regulation. *The Bell Journal of Economics and Management Science*, *2*(1), 3–21. https://doi.org/10.2307/3003160

Stroh, D. P. (2015). *Systems thinking for social change : A practical guide to solving complex problems, avoiding unintended consequences, and achieving lasting results*. Chelsea Green Publishing.

Sweeney, L. B. (2001). *When a butterfly sneezes : A guide for helping kids explore interconnections in our world through favorite stories*. Pegasus Communications.

Sweeney, L. B., & Sterman, J. D. (2000). Bathtub dynamics: Initial results of a systems thinking inventory. *System Dynamics Review*, *16*(4), 249–286. https://doi.org/https://doi.org/10.1002/sdr.198

The Apache Software, F. (n.d.). Apache beam: An advanced unified programming model. https://beam.apache.org/

Tip, T. (2011). Guidelines for drawing causal loop diagrams. *Systems Thinker*, *22*(1), 5–7.

Toulmin, S. (2003). *The uses of argument* (Updated). Cambridge University Press.

Trochim, W. M., Cabrera, D. A., Milstein, B., Gallagher, R. S., & Leischow, S. J. (2006). Practical challenges of systems thinking and modeling in public health. *American Journal of Public Health*, *96*(3), 538–546. https://doi.org/10.2105/ajph.2005.066001

Tunstall, L., Werra, L. v., & Wolf, T. (2022). *Natural language processing with transformers : Building language applications with hugging face* (First edition.). O'Reilly Media.

Valerdi, R., & Rouse, W. B. (2010). When systems thinking is not a natural act. *2010 IEEE International Systems Conference*, 184–189. https://doi.org/10.1109/SYSTEMS.2010.5482446

Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Vennix, J. A. M., Akkermans, H. A., & Rouwette, E. A. J. A. (1996). Group model-building to facilitate organizational change: An exploratory study. *System Dynamics Review*, *12*(1), 39–58. https://doi.org/https://doi.org/10.1002/(SICI)1099-1727(199621)12:1<39::AID-SDR94>3.0.CO;2-K

Weaver, R. K. (1986). The politics of blame avoidance. *Journal of Public Policy*, *6*(4), 371–398. https://doi.org/10.1017/S0143814X00004219

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, *35*, 24824–24837.

Weisstein, E. W. (2008). Mixed partial derivative. *MathWorld - A Wolfram Web Resource*. https://mathworld.wolfram.com/MixedPartialDerivative.html

West, T. A. P., Wunder, S., Sills, E. O., Börner, J., Rifai, S. W., Neidermeier, A. N., Frey, G. P., & Kontoleon, A. (2023). Action needed to make carbon offsets from forest conservation work for climate change mitigation. *Science*, *381*(6660), 873–877. https://doi.org/10.1126/science.ade3535

Whitehead, A. N. (1911). *An introduction to mathematics*. H. Holt; company; etc., etc.

Whyte, W. H. (1952). Groupthink. *Fortune Magazine*. https://fortune.com/article/groupthink-fortune-1952/

Wolfram Research, I. (2023a). Wolfram engine. https://www.wolfram.com/engine/

Wolfram Research, I. (2023b, 2023). Mathematica. https://www.wolfram.com/mathematica

Wolstenholme, E. F. (2003). Towards the definition and use of a core set of archetypal structures in system dynamics. *System Dynamics Review*, *19*(1), 7–26. https://doi.org/https://doi.org/10.1002/sdr.259

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *ArXiv*, *abs/2210.03629*.

Zhang, S., Dinan, E., Urbanek, J., Szlam, A., Kiela, D., & Weston, J. (2018). Personalizing dialogue agents: I have a dog, do you have pets too? *arXiv preprint arXiv:1801.07243*.