Evaluating the Capabilities of AI-Enabled Tools to Build and Edit Models with Modules

William J. Park, William Schoenberg

March 2026

Abstract

Introduction

Despite rapid progress in AI assisted SD modeling, an important gap remains: there is currently no systematic way to evaluate the extent to which AI-enabled modeling tools can correctly construct and modify SFD models in a modular manner.

Approach

To evaluate the ability of AI tools to correctly understand, create, and iterate on modular SFD models, three new evaluation categories were created and added to BEAMS Initiative's evaluation framework. These evaluations test the ability of AI tools to reason, translate, and modify modular SFD models.

Results

AI tools are able to competently use modules in SFD model construction and editing. Their performance wanes as the tasks grow in complexity. The tools excel on translation and modification tasks, and perform less well on reasoning tasks.

Discussion

The main challenge identified was with cross-level ghost creation, future AI tools should find a better way to emphasize cross-module relationships in their design. Reasoning tasks performance can likely be enhanced through iteration and multiple applications of existing tools in series.

1 Introduction

The rapid emergence of generative artificial intelligence (AI) has begun to reshape the practice, pedagogy, and accessibility of System Dynamics (SD). As Black and Greer (2024) note, expert modelers long served as an essential, but limiting, gateway to building credible SD models. Today, however, advances in large language models (LLMs) and AI-assisted modeling tools are expanding the range of people who can

engage in SD practice, lowering barriers to entry and making the long-held aspiration of "everybody thinking in systems" (Meadows, 2008) more feasible than ever before.

A growing body of research has begun to examine the role of AI in SD modeling. Early explorations demonstrated that generative tools such as ChatGPT could assist untrained users in producing qualitative causal maps and causal loop diagrams (CLDs) (Hosseinichimeh et al., 2024). Subsequent studies have investigated AI support for problem articulation, dynamic hypothesis generation, partial model formalization, and even model analysis (du Plooy & Oosthuizen, 2023; Giabbanelli et al., 2023; Armenia et al., 2024; Ghaffarzadegan et al., 2024; Jalali & Akhavan, 2024; Liu & Keith, 2024; Veldhuis et al., 2024; Giabbanelli et al., 2025; Hu, 2025). As these capabilities mature, AI-enabled modeling tools are becoming increasingly capable of producing substantive modeling artifacts.

Recent work has begun to systematize these developments. The sd-ai platform (Schoenberg et al., 2025) provides the first comprehensive framework cataloging the landscape of AI applications in SD. As a part of that platform Schoenberg (2026), developed the `quantitative` engine demonstrating how LLMs can construct, and iteratively refine quantitative stock-flow-diagram (SFD) models using natural language, revealing both the promise of AI-driven model generation and the challenges of ensuring correctness and interpretability. Together, these contributions highlight the central tension driving current research: while AI may dramatically expand participation in SD modeling, it may also enable users to generate models without necessarily engaging in systems thinking. This raises an urgent need for methods to assess the quality of AI-enabled SD modeling tools in order to maintain the rigor and credibility of the field.

1.1 Research Gap and Question

Despite rapid progress, an important gap remains: there is currently no systematic way to evaluate the extent to which AI-enabled modeling tools can correctly construct and modify SFD models in a modular manner. Existing work (Lynch & Schoenberg, 2026; Schoenberg et al., 2025) primarily evaluates generated models holistically e.g., whether a complete model "works", but no studies examine whether LLMs can reason about model structure in terms of reusable components, modules. Yet modular reasoning is foundational in SD practice: it underlies model conceptualization, team collaboration, and the construction of large models with well-defined boundaries.

This paper focuses on the following research question: *How effectively can AI-enabled SD modeling tools reason modularly when constructing, editing, and extending SFD models?* To address this question, we develop a suite of evaluations designed to test an AI modeling tool's ability to understand, manipulate, and integrate modules within SFD models. These evaluations complements existing research by shifting from "Can AI generate a model?" to "Can AI reason about the structure of a model in a way that aligns with professional SD practice?"

1.2 Benchmarking AI for Modeling and Simulation

Benchmarking is essential for assessing AI reliability and guiding safe deployment of AI systems. In machine learning more broadly, benchmarks such as SWE-Bench (Jimenez et al., 2023) have demonstrated how structured, task-specific evaluation suites can drive rapid improvements in model capability and

robustness. Within modeling and simulation, Lynch & Schoenberg (2026) argue for domain-specific benchmarks that reflect the unique reasoning tasks required in technical modeling disciplines and highlight risks associated with unvalidated AI assistance.

In this spirit, groups such as the BEAMS Initiative (Benchmarking for Evaluation of AI in Modeling and Simulation) (BEAMS, 2025), have begun developing systematic, simulation-oriented evaluations that reflect the complexities of SD modeling. BEAMS offers curated evaluations, ground-truth models, and structured scoring systems to assess how AI tools perform on domain-relevant reasoning and modeling tasks.

The evaluations introduced in this paper therefore add a new component to the BEAMS Initiative's benchmarking toolkit. They target an important feature of SD modeling, modular model structure, and provide a principled way to measure AI performance on SD modeling tasks that matter for both research and practice. By placing modular reasoning at the center of evaluation design, we aim to support the development of AI tools that not only generate models but do so in a way that makes them more useful and understandable.

1.3 Contribution

This paper contributes to the emerging SD & AI literature by:

- Identifying modular reasoning as a critical and underexplored capability for AI-enabled SD modeling tools.
- Developing a structured set of evaluations specifically designed to test AI performance on modular model construction, editing, and integration tasks.
- Positioning these evaluations within the broader ecosystem of AI benchmarking, including BEAMS, a domain-specific framework from the modeling and simulation fields.
- Providing empirical insight into the quality and reliability of AI-generated models, thereby supporting the safe and rigorous adoption of automation in SD practice.

As AI becomes increasingly integrated into the modeling workflow, the SD community faces a dual responsibility: to harness the potential of automation to broaden participation, and to ensure that the resulting models remain grounded in sound systems thinking. The evaluations proposed here represent a step toward meeting that responsibility by rigorously assessing how well AI tools can model modularly.

2 Methods

2.1 AI supported module creation

A significant recent development in this area is the addition of quantitative, stock and flow, modular SD modeling capabilities to the `sd-ai` platform's `quantiative` engine in February 2026. This update enables LLMs to create and edit modules in SFD models. This was done by introducing three key rules for translating generated output to valid systems dynamics models:

- Following XMILE conventions (Eberlein & Chichakly, 2013), variable names can be prefixed with a module name followed by a period; this corresponds to a variable being created inside the specified module name. For instance, a variable named "Hares.Birth Rate" is interpreted as a variable named "Birth Rate" placed inside a module named "Hares". If the specified module does not already exist inside the model, it is created.

- The `sd-json` format presented in (Schoenberg, 2026) was extended so that the variable object has an additional field `crossLevelGhostOf`. When filled in, this field specifies that the variable should "cross-level ghost" a variable from another module, allowing for cross-module causal relationships. For example, a variable named "Lynx.Hares" with specified `crossLevelGhostOf` "Hares.Hares" is interpreted as a cross-level ghost of the "Hares.Hares" variable located in the "Lynx" module.

- The `sd-json` format was also updated with a top level "modules" list, where each entry lists out a module, including it's parent module (if present). This allows for the creation of hierarchical models.

As part of the upgrade, the system prompt for the `quantiative` engine was significantly altered including the addition of an example modularized predator-prey model. The purpose of this addition was to show the LLM how to handle naming conventions and cross-level ghosting in a model. Furthermore, the system prompt of the engine was updated to include a section on proper role of cross-level ghosting in modular models and when their usage would be appropriate.

2.2 Evaluating AI supported module creation

To evaluate the ability of the `quantiative` engine to correctly understand, create, and iterate on modular SFD models, three new evaluation categories were created and added to BEAMS Initiative's evaluation framework (BEAMS, 2025):

1. **Modular Reasoning**, which evaluates the ability to create a modular SFD model given an expert description of a situation.

2. **Modular Translation**, which evaluates the ability to create a modular SFD model from a "nonsense" scenario involving fictional components and interactions.

3. **Modular Modification**, which evaluates the ability to interpret a given modular SFD model and modify its key causal processes and modular relationships.

The first two evaluation categories build directly upon preexisting non-modular quantitative equivalents produced by the BEAMS Initiative; namely, `quantitativeCausalReasoning` and `quantitativeTranslation`, respectively (Schoenberg et al., 2025, Lynch & Schoenberg, 2026). The third evaluation category, Modular Modification, builds upon the Modular Reasoning evaluation following in the general design of the `quantitativeIteration` evaluation (Schoenberg, 2026b), by providing a base model that the engine must interpret and refer to when responding to the user prompt.

2.2.1 Quantitative Modular Reasoning

Modular Reasoning, the first evaluation category, assesses an engine's ability to generate modular SFD models based on a user description of a real-world scenario. This evaluation directly tests if an engine can successfully understand real-world interactions in a modular context and properly form causal relationships between different modules in an SFD model.

The existing `quantitativeCausalReasoning` evaluation tests the ability of an AI-enabled modeling tool to extract key causal relationships from an expert description of a system and generate a correct SFDmodel (Schoenberg & Lynch, 2026). To integrate modular checks into the testing criteria, the `quantitativeModularReasoning` evaluation takes in a list of expected modules as part of the ground truth, and checks if this module list matches exactly with the LLM-generated output model. Under this evaluation, both unexpected and missing modules are reasons for failure. An additional check is integrated for variables to ensure that the `crossLevelGhostOf` field that links modules together, is correctly present in the generated output if expected.

The `quantitativeModularReasoning` evaluation is constructed in the manner described below. From `quantitativeCausalReasoning` evaluation, let $N(\cdot)$ be a canonicalization function that lowercases and trims component names. Note that N preserves the module name prefix when passed the name of a component within a module. Additionally, let $P(\cdot)$ be a function that normalizes and extracts the module name from a component label, if possible. For instance, $P(\text{“Hares.Birth Rate”}) = \text{“Hares”}$, and $P(\text{“count”}) = \text{“”}$ (as this component is not part of a module). Formally, the quantitative engine generates a SFD model with component names V_{llm}. For each variable $(x, g) \in V_{llm}$, if the output of $P(x)$ is not a blank string, $P(x)$ is added to a list of modules M present in the generated model. This list is checked against the list of modules $\mathcal{M}$ in the ground truth. Any module $\mathcal{M}_i$ that is not present in M is listed as a missing module and grounds for failure. Similarly, any module M_i that is not present in $\mathcal{M}$ is listed as an unexpected module and also grounds for failure. A generated model must contain exactly the expected modules in the ground truth to pass the evaluation. Additionally, every expected variable $(x, g) \in \text{Vars}(\mathcal{R}_i)$ requires the presence of a variable (y, h) in the generated model such that $N(x)$ matches $N(y)$ and, if g is specified, $N(g)$ matches $N(h)$. If no variables in V_{llm} satisfy both conditions, the variable (x, g) is declared to be missing (a failure). The procedure for verifying the presence of expected edges $(x, y, \sigma) \in \text{Edges}(\mathcal{R}_i)$ in E_{llm} remains unchanged from the `quantitativeCausalReasoning` evaluation; since $N(\cdot)$ preserves module names, it remains sufficient as a method of verification for expected edge creation in the generated output.

Algorithm 1 Modular Reasoning Evaluation (Required Modules)

Require: Background Text B; expectations $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_m\}$ where each group $\mathcal{R}_i$ contains required variables $\text{Vars}(\mathcal{R}_i)$ and required signed edges $\text{Edges}(\mathcal{R}_i)$; expected module list $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_m\}$ of required module names $\mathcal{M}_i$; quantitative experimental engine ε

Ensure: Failure List F.

Generate modular SFD

... *(unchanged)*

Canonicalize names

... *(unchanged)*

Initialize failures with structural checks

... *(unchanged)*

Check required module list

$M \leftarrow [\,]$
for all variable $(x, g) \in V_{llm}$ **do**
  if $P(x)$ is not blank and $P(x)$ not in M **then**
    Append $P(x)$ to M
for all name m in M **do**
  if m not in $\mathcal{M}$ **then**
    Append failure "unexpected module m" to F
for all name m in $\mathcal{M}$ **do**
  if m not in M **then**
    Append failure "missing module m" to F

Check each required variable group

for all requirement groups $R_i \in \mathcal{R}$ **do**
  $missing \leftarrow [\,]$
  (1) Required variables: must all be present
  for all variable $(x, g) \in \text{Vars}(\mathcal{R}_i)$ **do**
    $found \leftarrow$ **false**
    for all generated variables $(y, h) \in V_{llm}$ **do**
      if x matches y under $N(\cdot)$ **then**
        if g is unspecified or g matches h under $N(\cdot)$ **then**
          $found \leftarrow$ **true**
    if $found =$ **false then**
      Add (x with g) to $missing$
  (2) Required relationships: must all be present (polarity enforced if specified
  ... (unchanged)
  (3) Group Decision
  ... (unchanged)
return F

The `quantitativeModularReasoning` evaluation focuses on the ability of an engine to create a modular SFD model from a description of expert knowledge on a system, with no pre-existing model context. Critically, this evaluation emphasizes the need for engines to understand the role of cross-level ghost components in modular SFD models and reason through when to create them. Engines that fail to recognize the importance of cross-level ghost components and instead attempt to directly establish relationships between two components in different modules will fail associated tests in this evaluation, and perform poorly overall.

The test suite for the modular reasoning evaluation, listed in Table 1, consists of three groups of two scenarios. The first two tests, T1, and T2, cover scenarios where modules are explicitly prohibited from being created, to ensure that such modular engines can still create non-modular SFD models with key causal relationships when requested by the user. The next two tests, T3 and T4, cover simple two-module scenarios to assess an engine's basic understanding of modular structure and cross-level ghosting

abilities. Engines that pass T3 and T4 demonstrate the minimum level of modular reasoning to create valid two-module SFD models from expert descriptions. The final two tests, T5 and T6, cover complex three-module scenarios with involved cross-modular relationships that may not be immediately obvious from an expert description of a system.

Table 1: Quantitative Modular Reasoning Test Suite

Test ID	Test Description
R1	Create a system with no modules (Business Model)
R2	Create a system with no modules (Predator-Prey)
R3	Create a system with two modules (Predator-Prey)
R4	Create a system with two modules (Population Resource Usage)
R5	Create a system with three modules (Business Model)
R6	Create a system with three modules (Predator-Prey)

2.2.2 Quantitative Modular Translation

The Modular Reasoning evaluation focuses on the ability of an engine to generate a SFD model with appropriate modular relationships based on an expert description of a system. However, expert system descriptions of real-world scenarios are often ambiguous, with multiple correct ways to construct a model that simulates the given scenario. As a result, tests in the Modular Reasoning test suite cannot easily check for full correctness of a generated output model without requiring extensive manual review.

To address this issue, the Modular Translation evaluation assesses the raw ability of an engine to translate an exact description of a given scenario's causal relationships into a modular SFD model. Scenario descriptions are designed to be as simple as possible, with no possibility for ambiguity in modelling the scenario. This allows for much more rigorous automatic validation of generated models, to assess whether an engine has exactly understood and implemented the key causal relationships in a system. Furthermore, dataset contamination is always a concern when creating evaluation benchmarks, because an LLM that has seen a SFD model corresponding to a specific scenario in its training data is much more likely to perform well on an evaluation that references the same scenario, due to memorization and information recall rather than dynamic reasoning. To combat this, engines are instructed to create SFD models based on nonsensical, dynamically generated scenarios involving fictional objects such as "whatamajigs" and "frimbulators." These scenarios both eliminate any possible ambiguity in misconstruing causal relationships and are additionally highly unlikely to be present in any training dataset, making them a more reliable method for evaluating true reasoning capabilities.

The pre-existing `quantitativeTranslation` evaluation implements a non-modular version of "nonsensical scenario" test case generation and evaluation. To expand this evaluation to assess modular reasoning, the `quantitativeModularTranslation` evaluation expects the creation of an appropriately-named module for each fictional "component" mentioned in the problem scenario. Stocks named "count" are expected to be created within each module to represent the quantity of each object, and the creation of appropriate cross-level ghosts is expected when establishing a relationship between two separate objects (i.e. the quantity of object A affects the rate at which object B is consumed).

When creating the expectations for a test object T, the `quantitativeModularTranslation` evaluation takes in a list of expected stocks $\mathcal{S}$, where each stock $\mathcal{S}_i$ has an expected `name`, `initialValue`, inflows $\text{Inflows}(\mathcal{S}_i)$, and outflows $\text{Outflows}(\mathcal{S}_i)$. The evaluation then modularizes the expectations by converting

$\mathcal{S}_i.name$ to a component with module prefix $\mathcal{S}_i.name$ and stock name *count*. Any inflow and outflow relationships are also modularized, with any component names corrected to their modular equivalents. Furthermore, if the quantity of an object other than $\mathcal{S}_i.name$ affects $\mathcal{S}_i.count$, then a cross-level ghost of that object in the $\mathcal{S}_i.name$ module is added to a list of *expectedGhosts*.

The returned generated test T includes the list of modularized stocks as expected stocks, and the list of ghost variables as expected variables. Other attributes of T, such as the story-like problem description, remain unchanged from the base implementation in `quantitativeTranslation`.

Algorithm 2 Modular Translation Test Generation

Require: Expected stocks $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_n\}$ where each stocks $\mathcal{S}_i$ has attributes name and initialValue, required inflows Inflows($\mathcal{S}_i$), and required outflows Outflows($\mathcal{S}_i$); quantitative experimental engine ε
Ensure: Test case T.

Generate story-like problem description

... *(unchanged)*

Modify ground truth to have modular expectations

$expectedGhosts \leftarrow [\,]$
$modularStocks \leftarrow [\,]$
for all stock $\mathcal{S}_i \in \mathcal{S}$ **do**
  $\mathcal{S}_i.name \leftarrow$ Pluralize($\mathcal{S}_i.name$)
  $modularInflows \leftarrow [\,]$
  $modularOutflows \leftarrow [\,]$
  (1) Modularize inflows
  for all inflow $(fixed, rate, of) \in$ Inflows($\mathcal{S}_i$) **do**
    if $fixed$ is undefined **then**
      Append $(fixed, \text{undefined}, \text{undefined})$ to $modularInflows$
    $of \leftarrow$ Pluralize(of)
    if of matches $\mathcal{S}_i.name$ **then**
      Append $(\text{undefined}, rate, of)$ to $modularInflows$
    Append $(\mathcal{S}_i.name + \text{“.”} + of + \text{“ count”}, of + \text{“.count”})$ to $expectedGhosts$
    Append $(\text{undefined}, rate, \mathcal{S}_i.name + \text{“.”} + of + \text{“ count”})$ to $modularInflows$
  (2) Modularize outflows
  for all outflow $(fixed, rate, of) \in$ Outflows($\mathcal{S}_i$) **do**
    if $fixed$ is undefined **then**
      Append $(fixed, \text{undefined}, \text{undefined})$ to $modularOutflows$
    $of \leftarrow$ Pluralize(of)
    if of matches $\mathcal{S}_i.name$ **then**
      Append $(\text{undefined}, rate, of)$ to $modularOutflows$
    Append $(\mathcal{S}_i.name + \text{“.”} + of + \text{“ count”}, of + \text{“.count”})$ to $expectedGhosts$
    Append $(\text{undefined}, rate, \mathcal{S}_i.name + \text{“.”} + of + \text{“ count”})$ to $modularOutflows$
  (3) Modularize Stock
  $\mathcal{S}_i.name \leftarrow \mathcal{S}_i.name + \text{“.count”}$
  Inflows($\mathcal{S}_i$) $\leftarrow modularInflows$
  Outflows($\mathcal{S}_i$) $\leftarrow modularOutflows$
  Append $\mathcal{S}_i$ to $modularStocks$

Generate test

$T \leftarrow \{\,\}$
$T.expectations.stocks \leftarrow modularStocks$
$T.expectations.variables \leftarrow expectedGhosts$
(... unchanged)
return T

The evaluation of generated models for correctness in `quantitativeModularTranslation` remains largely unchanged from the base version, as module name prefixes are preserved in both the ground truth and the generated model when checking for equality. However, an additional check is added to ensure

that all expected cross-level ghost variables are created in the generated model.

Similar to the logic in `quantitativeModularReasoning`, for all variables $(x, g) \in E.variables$, representing a variable with name x that ghosts g, there must be a variable $(y, h) \in V_{llm}$ such that h is specified, x matches y, and g matches h. Otherwise, the variable (x, g) is declared missing. If any variables are missing, then a failure is appended to F with a list of missing cross-level ghost variables not present in the generated model.

Algorithm 3 Modular Translation Evaluation

Require: Background text B; expectations E; quantitative experimental engine ε
Ensure: Failure list F.

Generate modular SFD
... *(unchanged)*
Verify AI stocks with Ground Truth stocks
... *(unchanged)*
Verify AI variables with Ground Truth variables
Extract $V_{llm} \leftarrow R.model.variables$
$missing \leftarrow [\,]$
for all variable $(x, g) \in E.variables$ **do**
    $found \leftarrow$ **false**
    for all variable $(y, h) \in V_{llm}$ **do**
        if h is not undefined **then**
            if x matches y and g matches h **then**
                $found \leftarrow$ **true**
    if $found$ is **false then**
        Append (x, g) to $missing$
if $missing$ is not empty **then**
    Append failure "Missing cross-level ghost variables" with $missing$ to F
Check for expected inflows and outflows
... *(unchanged)*
return F

Table 2, below, lists the test suite for the Modular Translation evaluation. Tests T1 through T9 cover various scenarios highlighting different cross-modular causal relationships, in roughly increasing complexity. Tests T4 through T6 in particular cover the three main causal relationship paradigms for three-module systems, and tests T7 through T9 cover scenarios with extensive cross-module relationships to assess whether engines create proper cross-level ghost variables wherever necessary.

Table 2: Quantitative Modular Translation Test Suite

Test ID	Test Description
T1	Extract a single module system with one flow
T2	Extract a single module system with two flows
T3	Extract a two module system
T4	Extract a three module linear chain
T5	Extract a three module feedback system
T6	Extract a three module convergent system
T7	Extract a four module system with mixed flows
T8	Extract a four module branching system
T9	Extract a five module system

2.2.3 Quantitative Modular Modification

The Modular Reasoning and Modular Translation evaluations both focus on the abilities of quantitative engines to create brand-new modular SFD models based on narrative descriptions. However, an important

usage of the `quantitative` engine in SD modeling workflows is to make adjustments to an already existing model that was created by either the user or a previously generated model output. To address these cases, the `quantitativeModularModification` evaluation presents the LLM engine with various base modular SFD models and requests specific changes to the provided model. In particular, the Modular Modification evaluation focuses on the ability of an engine to iterate and modify a model created by the user. When given a model that already contains several modular relationships and cross-level ghost variables, can an LLM interpret them correctly and use that information to create new relationships or modify existing ones? The Modular Modification evaluation answers that question by forcing evaluated engines to follow rules and relationships established in pre-existing models rather than having the freedom to generate new ones. This evaluation is the most applicable to real-world use cases where experienced SD modellers may need to quickly add a new component or modify an existing relationship in a simulation that they have already been working on, rather than creating a brand new simulation model.

The `quantitativeModularModification` evaluation builds on many of the evaluation and test generation methods defined in `quantitativeModularReasoning`. To allow for the modification of existing models, a pre-generated SD model is passed as part of the user prompt in each generated test. Otherwise, evaluation for correctness against the ground truth is identical to the method used in the Modular Reasoning evaluation.

Table 3 outlines the test suite for the Modular Modification evaluation. Tests M1 through M3 assess the ability to "modularize" and "demodularize" a given SFD model — namely, turn an SFD model that contains no modules into one that does, and vica versa. Tests M4 and M5 assess the ability to directly iterate on an already modular SFD model by adding and removing specific modular components while preserving the validity of the remainder of the model. In particular, M4 requires engines to interpret key causal relationships in a model's existing modules in order to correctly add a new component to the model's scenario.

Table 3: Quantitative Modular Modification Test Suite

Test ID	Test Description
M1	Create a modular model from a non-modular system
M2	Create a non-modular model from a two-module system
M3	Create a non-modular model from a three-module system
M4	Add a module to a modular system
M5	Delete a module from a modular system

3 Results

The `quantitative` engine was tested on the three modular evaluation categories with four LLMs: Gemini 2.5 Flash, Gemini 3 Flash Preview (Low Thinking), Claude Opus 4.5, and Claude Opus 4.6. Results on each evaluation category, as well as individual successes and failures on each test and generation times for correct responses are listed in the tables below.

Table 4: Results Summary

LLM	Score			
	Modular Reasoning	Modular Translation	Modular Modification	Overall
Gemini-2.5-Flash	0.67	0.89	0.60	0.72
Gemini-3-Flash	0.50	1.00	0.60	0.70
Claude-Opus-4-5	0.67	1.00	1.00	0.89
Claude-Opus-4-6	0.67	1.00	0.80	0.82
Average	0.63	0.97	0.75	0.78

Each LLM was assigned a decimal score from 0 to 1 on each evaluation, representing the proportion of passed tests within the evaluation (where 1 is a perfect score). The overall score for each LLM was calculated as an unweighted arithmetic mean of the scores on the three evaluation categories.

Table 5: Response Times in Seconds

Test	Response Time (s)				
	Gemini-2.5	Gemini-3	Opus-4-5	Opus-4-6	Average
R1	16	16	30	33	24
R2	15	9	24	25	18
R3	31	11	26	38	27
R4	30	X (13)	31	33	31
R5	X (23)	X (17)	X (35)	X (38)	X
R6	X (32)	X (20)	X (39)	X (41)	X
T1	6	5	15	14	10
T2	9	7	22	18	14
T3	22	14	29	29	24
T4	19	18	24	33	24
T5	24	18	32	33	27
T6	X (24)	21	33	39	31
T7	31	23	38	45	34
T8	25	26	34	48	33
T9	38	33	43	51	41
M1	29	19	41	38	32
M2	25	16	28	37	27
M3	X (72)	31	48	53	44
M4	X (63)	X (28)	53	60	57
M5	23	X (23)	33	X (55)	28

An "X" for a test indicates a failure, with the response time included in parentheses. The average response time for each test was calculated as an unweighted arithmetic mean of response times among successful results; an "X" mark indicates that no successful responses were produced for that test.

All four LLMs achieved overall scores above 0.70, with no individual evaluation scores below 0.50. The `quantitative` engine performed exceptionally well on the Modular Translation evaluation, with only one failure (gemini-2.5-flash on T6) across the entire evaluation dataset. The weakest performance was on the Modular Reasoning evaluation, with no engines being able to successfully generate a valid response for tests R5 and R6.

Model performance on the Modular Modification evaluation was varied, with most LLMs performing well on the "modularize" and "demodularize" tests (M1-M3) but struggling on tests involving direct modifications to existing modular models (M4-M5). On M4, LLMs that failed did so due to failing to properly create cross-level ghosts of ecosystem area and population counts when computing lynx density, while LLMs that succeeded correctly extrapolated the Lynx-Hare predation relationship to the relationship between the Cougars and Lynx modules. On M5, LLMs that failed simply did not preserve

all key causal relationships when modifying the model to remove the Commerce module.

Almost every LLM achieved a perfect score on the Modular Translation evaluation, correctly creating ghost component references of relevant objects in each module when reference. However, model generation time notably increased with the complexity of the scenario description; generating a five-module system (T9) took 41 seconds on average to complete, more than quadruple the average for generating a single-module system (T1, at an average of 10 seconds).

On the Modular Reasoning evaluation, all four LLMs were able to create models when expressly told to avoid modular relationships (R1-R2), and three of the four were able to perfectly create two-module SFD models (with only Gemini 3 Flash Preview failing on R4). However, none of the LLMs were able to correctly create a three-module SFD model based on expert system descriptions (R5-R6). Failure reasons on R5 and R6 varied, but by far the most common reason was the failure to create cross-module relationships using the proper ghost components; for instance, in R6, none of the four LLMs managed to correctly create a ghost of the "lions.count" stock in the "foxes" module to influence fox deaths.

4 Discussion

The results of this study reveal both the growing capability and the persistent limitations of current AI-enabled tools for modular System Dynamics modeling. While the evaluated LLMs demonstrated strong performance on tasks where causal structure was explicit—particularly in the Modular Translation evaluation, their performance declined rapidly as tasks required deeper conceptual reasoning about how modules interact. The most consistent source of failure across evaluations was the incorrect handling of cross-level ghost components, especially in complex or multi-module scenarios. The `quantitative` engine often failed to recognize when a cross-level ghost was required to represent a causal dependency between modules. This difficulty was most apparent in the Modular Reasoning evaluation, where none of the models successfully produced valid three-module SFD models based on system descriptions, and in the Modular Modification evaluation, where the `quantitative` engine struggled to maintain valid cross-module connections when modifying existing structures. These shortcomings suggest that while LLMs can follow explicit instructions, they still lack the ability to reliably produce modular models of significant size or complexity.

These findings point toward a broader design challenge for future AI-based modeling tools: the `quantitative` engine does not sufficiently emphasize cross-module relationships in its internal representations or output constraints. Because modular modeling requires understanding not only within-module dynamics but also the structural logic linking separate components, future tools need better support for identifying when a module boundary implies the need for a ghosted variable. Without such support, the `quantitative` engine tends to default toward non-modular reasoning patterns, malformed module structures. Improving the clarity and prominence of cross-module relationships in the system prompt, or changing the design of the structured output used by the `quantitative` engine may help to address these gaps. For example, shifting away from the current "module.variable" flattened naming convention toward an explicit attribute-based representation of module membership could reduce ambiguity and improve the `quantitative` engine's ability to track modular structure. Such an approach may also increase robustness in modification tasks by reducing the risk of unintentionally breaking relationships or losing

cross-level ghosts when modules are restructured, behaviors observed in the M4 and M5 failures.

5 Conclusions

Broadly, this work highlights the necessity of targeted benchmark evaluations when introducing advanced modeling capabilities into AI enabled modeling tools. The three test suites developed here expose nuanced weaknesses that would not surface in the existing assessments of the BEAMS Initiative. In particular, modular reasoning, a core skill in professional SD practice, poses challenges that differ qualitatively from those seen in non-modular modeling tasks. The evaluation framework introduced in this paper provides a structured method for identifying these challenges, enabling systematic improvement of future modeling engines. As AI enabled modeling tools become increasingly integrated into modeling workflows, such domain-specific benchmarks will be vital for ensuring that automation supports, rather than undermines, sound systems thinking and model quality.

References

Armenia, S., Franco, E., Iandolo, F., Maielli, G., & Vito, P. (2024). Zooming in and out the landscape: Artificial intelligence and system dynamics in business and management. *Technological Forecasting and Social Change*, 200, 123131. https://doi.org/10.1016/j.techfore.2023.123131

Benchmarking and evaluating AI for modeling and simulation. institute for artificial intelligence and data science. (2025, December). https://www.buffalo.edu/ai-data-science/research/beams.html

Du Plooy, C., & Oosthuizen, R. (2023). AI USEFULNESS IN SYSTEMS MODELLING AND SIMULATION: GPT-4 APPLICATION. *South African Journal of Industrial Engineering*, 34(3). https://doi.org/10.7166/34-3-2944

Eberlein, R. L., & Chichakly, K. J. (2013). XMILE: A new standard for system dynamics. *System Dynamics Review*, 29(3), 188–195. https://doi.org/10.1002/sdr.1504

Ghaffarzadegan, N., Majumdar, A., Williams, R., & Hosseinichimeh, N. (2024). Generative agent-based modeling: An introduction and tutorial. *System Dynamics Review*, 40(1), e1761. https://doi.org/10.1002/sdr.1761

Giabbanelli, P. J. (2023, June 21). GPT-based models meet simulation: How to efficiently use large-scale pre-trained language models across simulation tasks. https://doi.org/10.48550/arXiv.2306.13679

Giabbanelli, P. J., Gandee, T. J., Agrawal, A., & Hosseinichimeh, N. (2025). Benchmarking and assessing transformations between text and causal maps via large language models. *Applied Ontology*, 20(1), 125–134. https://doi.org/10.1177/15705838241304102

Hosseinichimeh, N., Majumdar, A., Williams, R., & Ghaffarzadegan, N. (2024). From text to map: A system dynamics bot for constructing causal loop diagrams. *System Dynamics Review*, 40(3), e1782. https://doi.org/10.1002/sdr.1782

Hu, B. (2025). ¡span style="font-variant:small-caps;"¿ChatPySD¡/span¿ : Embedding and simulating system dynamics models in ¡span style="font-variant:small-caps;"¿ChatGPT¡/span¿ -4. *System Dynamics Review*, 41(1), e1797. https://doi.org/10.1002/sdr.1797

Jalali, M. S., & Akhavan, A. (2024). Integrating ¡span style=”font-variant:small-caps;”¿AI¡/span¿ language models in qualitative research: Replicating interview data analysis with ¡span style=”font-variant:small-caps;”¿ChatGPT¡/span¿. *System Dynamics Review*, 40(3), e1772. https://doi.org/10.1002/sdr.1772

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024, November 11). SWE-bench: Can language models resolve real-world GitHub issues? https://doi.org/10.48550/arXiv.2310.06770

Liu, N.-Y. G., & Keith, D. R. (2025, March 23). Leveraging large language models for automated causal loop diagram generation: Enhancing system dynamics modeling through curated prompting techniques. https://doi.org/10.48550/arXiv.2503.21798

Lynch, C., & Schoenberg, W. (2026). From principles to practice: Risks, case studies, and domain-specific benchmarking for large language models in modeling and simulation. In *Simulation foundations, methods and applications*. Springer Nature.

Meadows, D. H., & Wright, D. (2008). *Thinking in systems: A primer*. Chelsea Green Pub.

Schoenberg, W. (n.d.). *Quantitative iteration [source code]* [Quantitative iteration]. https://github.com/UB-IAD/sd-ai/blob/main/evals/categories/quantitativeIteration.js

Schoenberg, W. (2026). Building and learning with models using ¡span style=”font-variant:small-caps;”¿AI¡/span¿. *System Dynamics Review*, 42(1), e70019. https://doi.org/10.1002/sdr.70019

Schoenberg, W., Girard, D., Chung, S., O'Neill, E., Velasquez, J., & Metcalf, S. (2025, March 19). How well can AI build SD models? https://doi.org/10.48550/arXiv.2503.15580

Veldhuis, G. A., Blok, D., De Boer, M. H., Kalkman, G. J., Bakker, R. M., & Van Waas, R. P. (2024). From text to model: Leveraging natural language processing for system dynamics model development. *System Dynamics Review*, 40(3), e1780. https://doi.org/10.1002/sdr.1780