LunaSim Copilot: A Case Study on LLM-generated System Dynamics Models

William J. Park*, Karthik S. Vedula*†, Ishan Khetarpal, Mark R. Estep

Poolesville High School, Poolesville, MD, USA

*Equal Contribution. †Corresponding Author: karthik@vedula.me

Abstract

[ADD ABSTRACT HERE]

Keywords: keyword1, keyword2, keyword3, keyword4, keyword5, keyword6.

1. Introduction

System dynamics (SD) modeling involves visually representing the components of a complex system—often mirroring real-world processes—and simulating their interactions to predict how the system evolves over time. A key approach is stock-and-flow diagrams, where stocks represent accumulative elements, while flows control their changes. Variables/converters help group calculations performed at each timestep for clarity, and influences/connectors use arrows to indicate relationships between elements. SD modeling software facilitates this entire design process.

Recently, artificial intelligence, specifically large language models (LLMs) have been incorporated as assistive technologies within the software development programs. Examples include GitHub Copilot on Visual Studio Code and Amazon Codewhisperer ('CodeWhisperer', n.d.; 'GitHub Copilot', 2025). LLMs specialized in generating computer programs have also been developed, such as Code Llama and Codestral ('Codestral', 2024; Rozière et al., 2024). These tools have significantly increased the productivity of the user of these software development programs. The integration of such kinds of LLMs into the SD modeling development environment, however, is underexplored.

Natural language processing has been applied for information extraction in order to generate SD diagrams (causal loop diagrams, stock and flow models, etc). This includes COATIS, which used causal verb patterns to identify causal relationships (Garcia, 1997); Chan & Lam (2005) also explored causation relation extraction from natural language text. Hosseinichimeh et al. (2024) used LLMs to construct causal loop diagrams from given textual data. Some studies evaluated the ability of LLMs to act as assistants aiding a user creating a SD model. Akhavan & Jalali (2024) evaluated the use of ChatGPT in the creation of SD models starting from the problem definition to the final model and analysis. Liu & Keith (2024) also evaluated LLMs on the ability of generating SD models. However, these studies do not feature these LLM-assistants integrated into SD modeling software; rather, they interface with LLMs externally.

Additionally, with the recent developments of reasoning models such as OpenAI's o3-mini and DeepSeek's R1, LLMs have the potential to perform even better on SD modeling tasks (DeepSeek-AI et al., 2025; 'OpenAI o3-mini', n.d.). Reasoning ability enables LLMs to tackle problems in multiple steps, which can possibly be beneficial for the complex nature of the task of creating SD models. Therefore, this paper includes these models as well to assess their performance.

We make the following key contributions:

1. We introduce an AI-powered assistant, *LunaSim Copilot*, **directly integrated** into our system dynamics modeling software called LunaSim (Vedula et al., 2024), enabling seamless AI-assisted model generation and editing.

2. We test this AI assistant on five system dynamics examples, assessing its ability to interpret and generate stock-and-flow models.

3. We evaluate four state-of-the-art LLMs as models behind the AI assistant, including two **reasoning models**, comparing their accuracy and reasoning ability in assisting system dynamics modeling.

2. Methods

2.1 Software Architecture

LunaSim Copilot is integrated into our SD modeling software called LunaSim. LunaSim is a web-based SD modeling software for creating, simulating, and visualizing stock and flow diagrams. Since LunaSim is web-based, LunaSim Copilot interacts with LLMs through web APIs. Given a user instruction (e.g. "create a stock and flow model for modeling amoeba growth"), the instruction and the LLM system prompt (which informs the LLM of its objective and gives context on how to generate stock and flow models in regards to rules and formatting) are sent to the LLM. The LLM then outputs the new SD model in the LunaSim file format (including specifying equations of different stocks, flows, etc) and the new model is loaded into the LunaSim application. Figure 1 displays an overview of this architecture.

The LLM has access to the entire chat history, allowing the user to reference previous instructions and model outputs in new instructions. Therefore, LunaSim Copilot can aid in the generation of SD models from scratch or edit existing SD models as per user

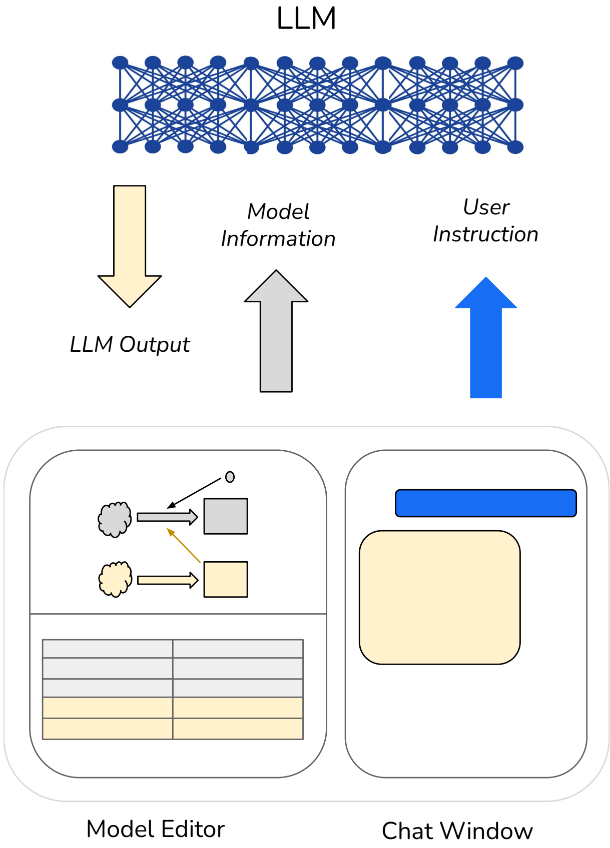

Figure 1: Architecture of LunaSim Copilot. The user provides an instruction to the LLM on what changes or model generation must be made. The instruction along with the current model details is sent to the LLM. The LLM returns a new model conforming to this new instruction, and the new model is loaded back into the application.

instruction. Note that since LunaSim Copilot is built on top of LunaSim, the user has full access to LunaSim's features (SD model editing, equations, visualization).

2.2 Experimental Setup

2.2.1 LLMs Tested

We evaluated a total of four LLMs: OpenAI o3-mini, OpenAI GPT-4o, Deepseek-R1, and Anthropic Claude 3.7 Sonnet (with no reasoning enabled), of which OpenaAI o3-mini and Deepseek-R1 are reasoning models ('Claude 3.7 Sonnet and Claude Code', n.d.; DeepSeek-AI et al., 2025; 'Hello GPT-4o', n.d.; 'OpenAI o3-mini', n.d.). All models were used with the default hyperparameters.

2.2.2 SD Model Generation

Each of the four LLMs were evaluated on five tasks. These tasks required the LLM to generate the SD model schema (according to the LunaSim model format) given a request from the user. These SD models were the following:

- **Algae growth**: simple logistic regression model
- **Hooke's law**: oscillating spring with weight on the end pulled by gravity
- **Projectile motion**: 2D model of a projectile factoring in air resistance
- **Trebuchet**: simulates a see-saw-like catapult using rotational motion, as outlined in Vedula et al. (2024)
- **Binary stars**: simulates the trajectories of two planetary objects that exert a gravitational force on each other, as outlined in Vedula et al. (2024)

The LLM was required to generate the SD model from scratch, with only given the system prompt and the user instruction, i.e., it did not have an existing SD model to build from. LLM outputs were evaluated using rubrics created for each kind of SD model task. Each rubric consisted of a general section and the SD model-specific section. The general rubric assessed the validity of the LLM output: whether it correctly outputs into LunaSim's (JSON-based) expected format. This not only includes whether the SD model loads into LunaSim, but also whether the SD model follows the rules of system dynamics (e.g. influences cannot point into stocks). The SD model-specific section evaluated the accuracy of LLM output with respect to an exemplar SD model for that particular scenario. The presence of specific elements (e.g. a stock representing x-position for the projectile motion scenario) are evaluated. Accuracy of corresponding equations for each of these elements is also assessed. The totals for both parts of these rubrics were calculated and compared among different LLMs. Rubrics are included in Appendix C.

3. Results

Table 1 displays the performance of the LLMs on the five tasks. o3-mini had the highest average score of 94.6% along with the lowest standard deviation of 8.4%. While GPT-4o performed decent on the simple Algae growth and Hooke's law examples, it severely underperformed on the other three (more complex) scenarios. Claude 3.7 and o3-mini performed significantly better on all of the five scenarios, while Deepseek-R1 struggled with the Trebuchet SD model task. Three of the four models (all except GPT-4o) performed the lowest on the trebuchet task. Claude 3.7 and o3-mini were the two models that achieved perfect scores, with Claude 3.7 acing the Algae growth and Binary stars tasks and o3-mini acing Algae growth and Hooke's law tasks. Specific model scores, visualizations of SD models, and summaries on missed points are in Appendix A.

Table 1: LLM performance on each task based on the rubrics. Values are percentage correct, with rubrics assessing whether the LLM output adheres to SD modeling rules and whether the LLM output is in the valid format.

	Accuracy (% of total points from rubric)			
SD Model	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1
Algae growth	84.1	100.0	100.0	97.7
Hooke's law	88.2	90.2	100.0	94.1
Projectile motion	46.4	76.8	97.1	100
Trebuchet	58.0	86.0	80.0	67.0
Binary stars	57.3	100.0	95.8	97.9
Average:	66.8	90.6	**94.6**	91.3
Std. Dev:	18.3	9.9	**8.4**	13.8

4. Discussion

Our study highlights the promising capability of LLMs in assisting in SD modeling. Claude 3.7, o3-mini, and Deepseek-R1 particularly displayed significant capability of generating SD models given high-level user instructions. These LLMs illustrated the ability to discern the kind of element (stock, flow, variable) a given component of a simulation should be, since the prompts given to them did not mention the specifics of the types of elements each component should be. LLMs also displayed the ability to predict intermediate components of the SD model scenarios: components that were neither the input nor output elements outlined by the prompts.

Despite reasoning models being intended for excelling at multi-step problems such as generating SD models, Claude 3.7 (which was ran without reasoning mode) had comparable performance to the two reasoning models: o3-mini and Deepseek-R1. Additionally, the trebuchet SD model proved more difficult for the three models than the binary star system. This might be due to the fact that the trebuchet model contained less stocks-and-flow relationships, rather having more complex equations underlying those limited stock-and-flow relationships. This is in contrast with the binary star model, which had many more stock-and-flow relations but with simpler equations. The improved performance on the binary star system from these LLMs suggests that SD models that are more broken down to simpler components (as is the objective of SD) are easier for LLMs to create.

This study, by evaluating these LLMs through the LunaSim Copilot framework, assessed not only the ability of LLMs to "think" in terms of SD, but also the practical ability of them to output in a usable (i.e. directly loadable, visually clear) SD format. Since our rubrics contained evaluations of whether the model loads correctly and quality of element positioning, this study illustrates the ability for LLMs to assist SD model generation seamlessly through direct integration into a SD modeling software.

4.1 Limitations

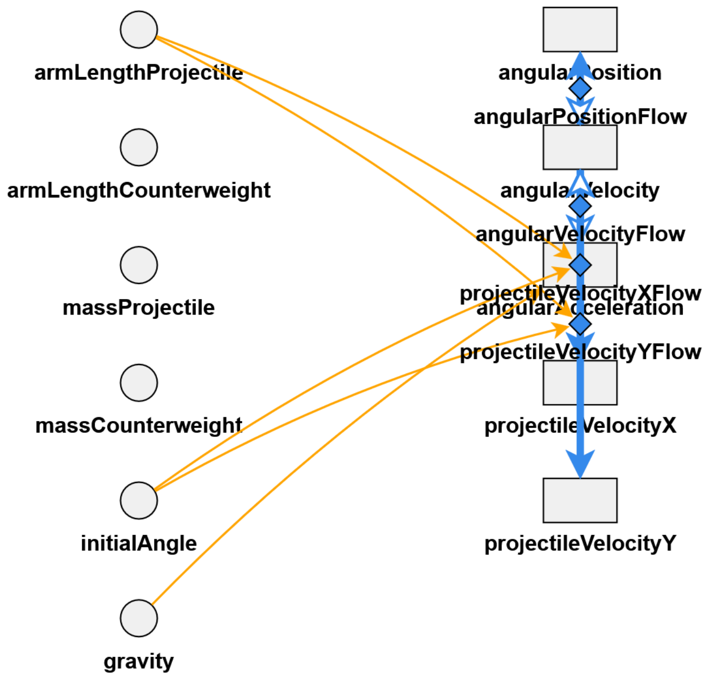

Figure 2: Example of incoherent element placement.

Our study faces certain limitations. The current version of LunaSim Copilot does not support ghosting, limiting the degree to which outputs can be graphically organized. Additionally, LLMs inherently do not have the ability to visualize the placement of stocks and flows (which was evaluated through assessing position quality of LLM-generated models in this study), hindering the visual organization of SD models. This can lead to some cases where LLM-generated SD models are jumbled. However, many of these cases are easily resolved through user intervention by dragging the elements around in LunaSim.

The evaluation rubric used in this study may not always capture the full spectrum of model quality, potentially affecting assessment reliability. Specifically, the point weights in the rubric are not definitive, as SD model quality (apart from whether the model yields the same values) is subjective. In addition, the study focuses on four LLMs, which can be expanded to include other models as well. Finally, this study does not utilize multiple human evaluators of SD models.

5. Conclusion

Code & Data Availability

The data availability statement should provide information on where and under what conditions the data directly supporting the publication can be accessed. Sample data availability statements are available at the following site: https://academic.oup.com/pages/open-research/research-data#Data%20Availability%20Statements.

References

Akhavan, A., & Jalali, M. S. (2024). Generative AI and simulation modeling: How should you (not) use large language models like ChatGPT [Publisher: John Wiley & Sons, Ltd]. *System Dynamics Review*, 40(3), e1773. https://doi.org/10.1002/sdr.1773

Chan, K., & Lam, W. (2005). Extracting causation knowledge from natural language texts. *Int. J. Intell. Syst.*, 20(3), 327–358.

Claude 3.7 Sonnet and Claude Code. (n.d.). Retrieved March 16, 2025, from https://www.anthropic.com/news/claude-3-7-sonnet

Codestral. (2024). Retrieved March 16, 2025, from https://mistral.ai/news/codestral

CodeWhisperer. (n.d.). Retrieved March 16, 2025, from https://docs.aws.amazon.com/codewhisperer/latest/userguide/what-is-cwspr.html

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., … Zhang, Z. (2025, January). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning [arXiv:2501.12948 [cs]]. https://doi.org/10.48550/arXiv.2501.12948

Garcia, D. (1997). COATIS, an NLP system to locate expressions of actions connected by causality links. In E. Plaza & R. Benjamins (Eds.), *Knowledge Acquisition, Modeling and Management* (pp. 347–352). Springer. https://doi.org/10.1007/BFb0026799

GitHub Copilot. (2025). Retrieved March 16, 2025, from https://github.com/features/copilot

Hello GPT-4o. (n.d.). Retrieved March 16, 2025, from https://openai.com/index/hello-gpt-4o/

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

Liu, N.-Y. G., & Keith, D. (2024, June). Leveraging Large Language Models for Automated Causal Loop Diagram Generation: Enhancing System Dynamics Modeling through Curated Prompting Techniques. Retrieved March 14, 2025, from https://papers.ssrn.com/abstract=4906094

OpenAI o3-mini. (n.d.). Retrieved March 16, 2025, from https://openai.com/index/openai-o3-mini/

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., . . . Synnaeve, G. (2024, January). Code Llama: Open Foundation Models for Code [arXiv:2308.12950 [cs]]. https://doi.org/10.48550/arXiv.2308.12950

Vedula, K. S., Simms, S., Patil, A., Khetarpal, I., & Estep, M. R. (2024). LunaSim: A Lightweight, Web-Based, Open-Source System Dynamics Modeling Software. *2024 International System Dynamics Conference*. https://proceedings.systemdynamics.org/2024/papers/P1049.pdf

Appendix

A. SD Model & Score Details

A.1 Algae Growth

Prompt: Create a model to simulate the growth of an algae colony using a logistic growth curve. Add a carrying capacity, initial population, and a coefficient of growth.

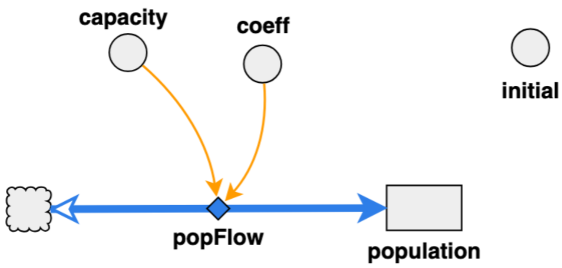

Figure A1: Algae SD Model

Table A1: Subscores for Algae Growth Model

Criteria	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1	Max Points
General Rubric					
Output Integrity	5	5	5	5	5
Names	5	5	5	5	5
Flows	1	1	1	1	1
Variables	3	3	3	2	3
Positioning	2	4	4	4	4
SD model-specific rubric					
Initial Conditions	6	6	6	6	6
Relationships	15	20	20	20	20
Summary					
Total	37	44	44	43	44

Comments:

- **GPT-4o:** A bit messier than o3-mini but got the main components correct
- **Deepseek-R1:** Initial population hardcoded

A.2 Hooke's Law

Prompt: Create a model for the oscillating motion of a block on a spring according to Hooke's law. Initial variables should be starting position, mass, and the spring constant. The block originally starts at rest.

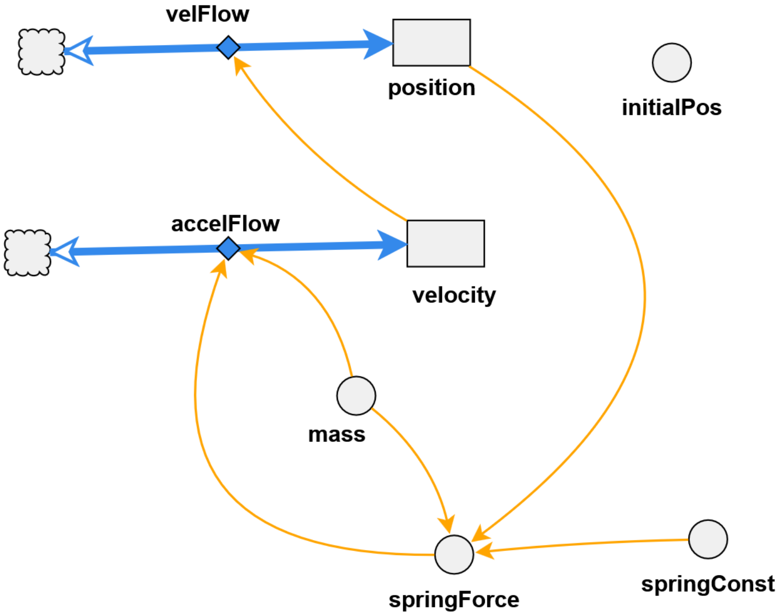

Figure A2: Hooke's Law SD Model

Table A2: Subscores for Hooke's Law Model

Criteria	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1	Max Points
General Rubric					
Output Integrity	0	5	5	5	5
Names	5	5	5	5	5
Flows	2	2	2	2	2
Variables	2	3	3	2	3
Positioning	4	4	4	2	4
SD model-specific rubric					
Initial Conditions	6	6	6	6	6
Relationships	26	21	26	26	26
Summary					
Total	45	46	51	48	51

Comments:

- **GPT-4o:** Included comments in JSON which made the file invalid. Hardcoded initial position.
- **Claude 3.7:** Almost perfect besides minor issue in position equation.
- **Deepseek-R1:** Bad positioning & hardcoded start position. Correct numerical output however.

A.3 Projectile Motion

Prompt: Create a model for 2D projectile motion. Initial variables should be starting position, mass, and angle. Incorporate a drag coefficient that affects acceleration proportional to velocity.

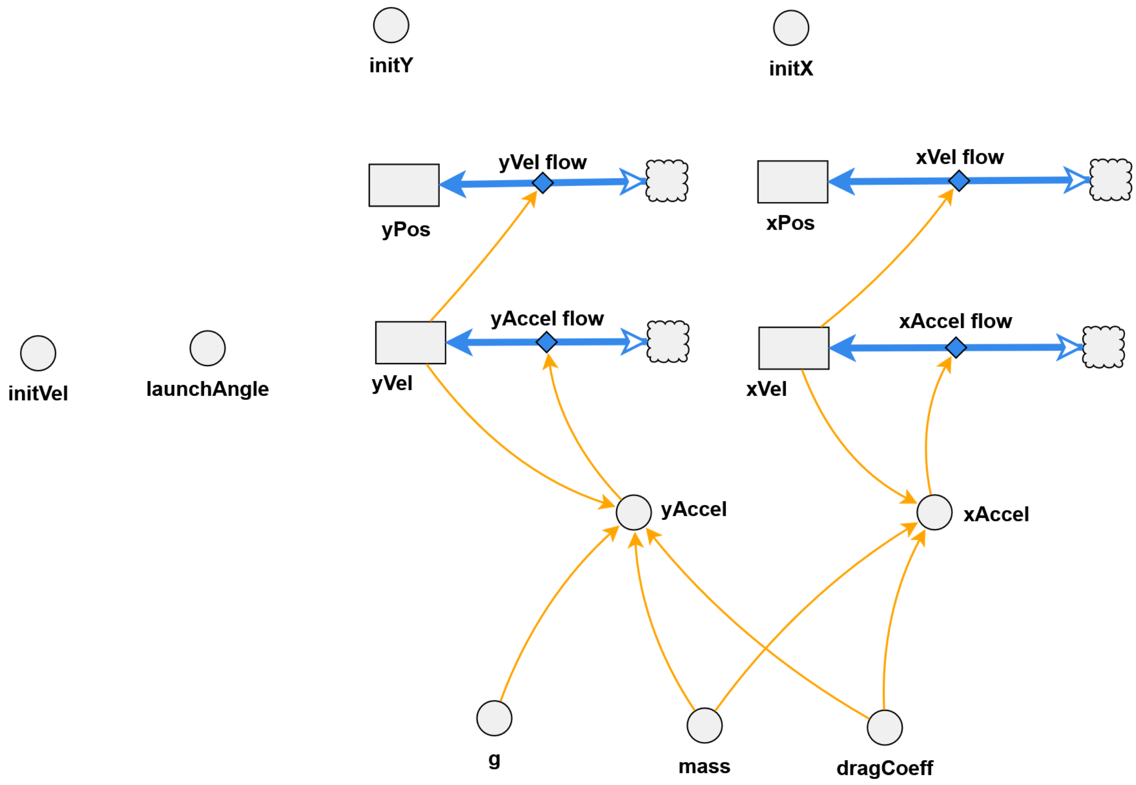

Figure A3: Projectile SD Model

Table A3: Subscores for Projectile Motion Model

Criteria	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1	Max Points
General Rubric					
Output Integrity	5	5	5	5	5
Names	5	5	5	5	5
Flows	1	4	4	4	4
Variables	5	7	7	7	7
Positioning	2	4	2	4	4
SD model-specific rubric					
Initial Conditions	8	14	14	14	14
Relationships	6	14	30	30	30
Summary					
Total	32	53	67	69	69

Comments:

- **GPT-4o:** Failed to split initial conditions into x-y components. Flows were drawn from stocks (incorrect) instead of creating a cloud source element. Correct equation but incorrect flow origin.

- **Claude 3.7:** Notably, used angles in degrees and converted to RAD for flow/stock equations. Drag coefficient equation was incorrect which led to incorrect numerical results. However, excellent model structure.

- **o3-mini:** All equations and relationships perfect, spacing of elements could be better however

A.4 Trebuchet

Prompt: Create a model that simulates the movement of a trebuchet. The arm of the trebuchet can be simulated by a line segment that rotates around a fixed point. Initial variables include the length and mass of the portion of the trebuchet arm with the projectile and the length and mass of the portion of the trebuchet arm with the counterweight. The mass of the projectile and the counterweight are also initial variables. Finally, include the starting angle of the trebuchet as a variable. Other constants such as gravity should also be stored as variables. The output stocks/variables for the simulation should be: beam angular speed & angular acceleration, the launch velocity (speed & angle components) of the projectile at any given moment. Make an appropriate element for each of these. Any other helper nodes or elements can be created if necessary.

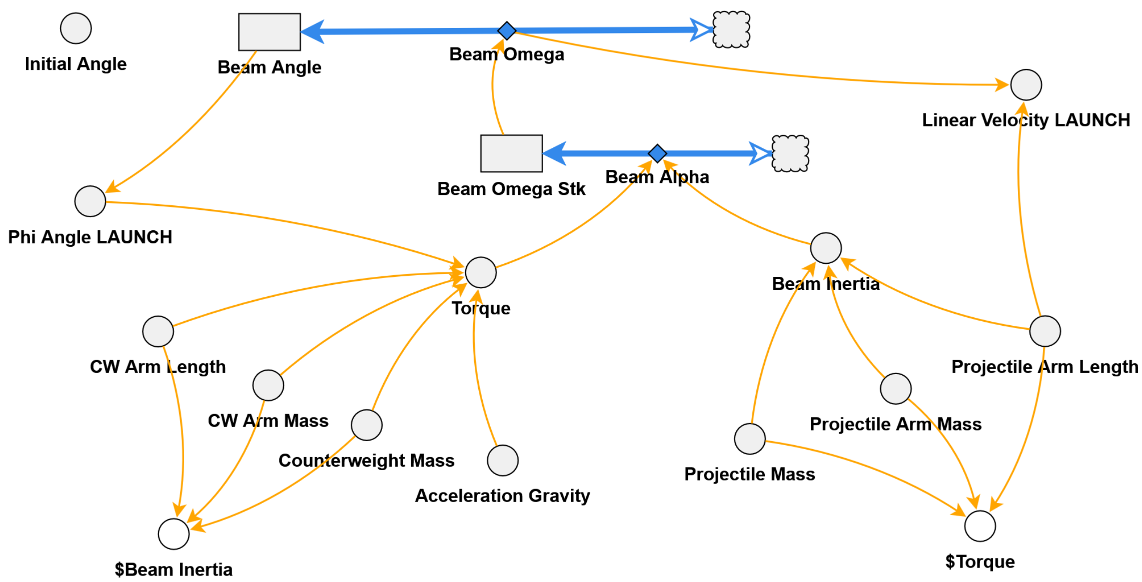

Figure A4: Trebuchet SD Model

Table A4: Subscores for Trebuchet Model

Criteria	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1	Max Points
General Rubric					
Output Integrity	5	5	5	5	5
Names	5	5	5	5	5
Flows	0	2	2	2	2
Variables	8	10	10	7	10
Positioning	2	4	4	2	4
SD model-specific rubric					
Initial Conditions	16	16	16	12	16
Relationships	16	44	38	34	58
Summary					
Total	58	86	80	67	100

Comments:

- **GPT-4o:** Failed to recognize the difference between when to use a stock or variable for output. No angle stock. Incorrect flow origins, no clouds. Treated trebuchet arm as a point mass rather than a rotating bar for inertia. Did not incorporate angle or trebuchet arm into torque. Failed to establish a relationship between angular acceleration, angular speed, and angular position. Failed to differentiate between project launch angle/speed and beam angular speed.
- **Claude 3.7:** Treated trebuchet arm as a point mass at the same location as the projectile/counterweight rather than a rotating bar for inertia. Did not incorporate trebuchet arm into torque. Very close to the answer key.
- **o3-mini:** Very impressive performance with logical element placement. The model failed to incorporate the weight of the trebuchet arm into either torque or inertia, causing inaccuracies in the final output model. The model also used Math.sin instead of the correct Math.cos in torque calculations. However, there is a significant similarity between the model produced by the AI and the answer key model.
- **Deepseek-R1:** Failed to incorporate the mass of the trebuchet arm into any component of the model. Treated the trebuchet as a simple "two-point" system and ignored the trebuchet arm itself. Failed to create the requested launch angle output variable.

A.5 Binary Stars

Prompt: Create a model that simulates a binary star system in space. Initial variables should specify the masses of each star. The starting x and y-positions and starting x and y-velocities of each star can be hard-coded into the initial values of the relevant stocks. The two stars should move and orbit around each other, and the only force acting on either star should be the force of each star's gravity on the other. Any other constants should be stored in variables. Create intermediate variables storing the force of gravity on each star (x-y components), the acceleration of each star (x-y components), and the distance between the two stars (overall & x-y components) at any given time.

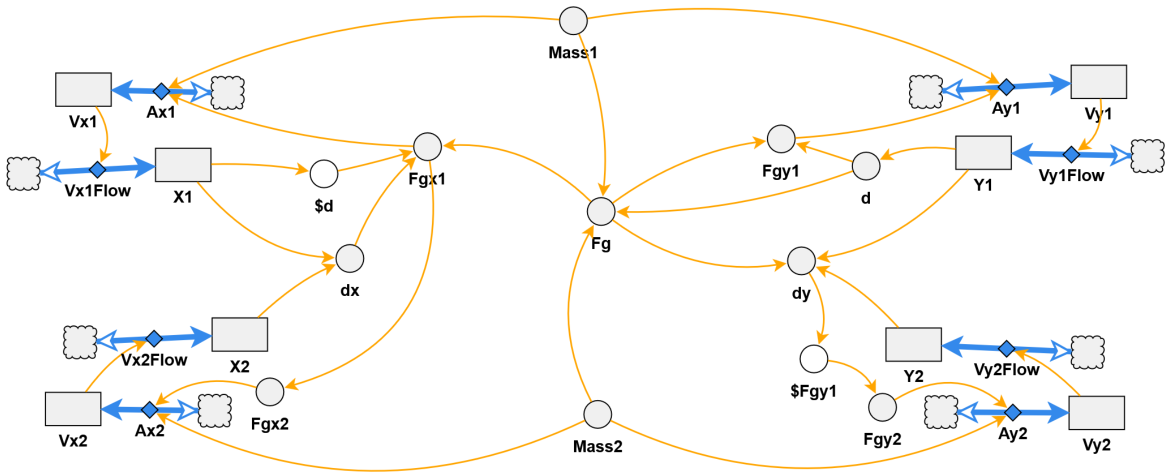

Figure A5: Binary Stars SD Model

Table A5: Subscores for Binary Stars Model

Criteria	GPT-4o	Claude 3.7	o3-mini	Deepseek-R1	Max Points
General Rubric					
Output Integrity	5	5	5	5	5
Names	5	5	5	5	5
Flows	2	8	8	8	8
Variables	12	12	10	10	12
Positioning	2	4	2	4	4
SD model-specific rubric					
Initial Conditions	12	12	12	12	12
Relationships	17	50	50	50	50
Summary					
Total	55	96	92	94	96

Comments:

- **GPT-4o:** Equations seem correct but the model fails to understand how flow relationships connect related elements or how the variables should interact with each other. Understands the principles behind a binary star system but fails to correctly integrate them into a new scenario.

- **o3-mini:** Output exactly matches the answer key, very impressive. Struggles with positioning of elements graphically. Failed to create the requested output variables for F_{g2}.

- **Deepseek-R1:** Matches output model exactly, good spatial reasoning when placing elements. Failed to create the requested output variables for F_{g2}.

B. System Prompt

The system prompt used for the LLM is available at https://github.com/oboy-1/LunaSimCopilot/blob/main/prompts/prompt.txt

C. Rubrics

Table C1: General Scoring Rubric

Criterion	**Penalty**	**Max Points**
Output Integrity	0% or 100%	5
Names	-0.5 if name contains “ position” or “ flow” at the end (when not used as such in the equation) -1 if element has a different name from the one in the equation	5 (minimum 0)
Flows	-0.5 per “flipped” flow -0.75 if flow draws from a stock instead of a cloud -1 per missing flow -1 per incorrect flow	# of flows in answer key
Variables	-1 if variable is missing & hardcoded in equations -0.5 if variable is defined but not used -0.5 if variable is used but not defined	# of variables in answer key
Positioning	100% - good placing 50% - good placing, but with overlap 0% - incoherent	4

Table C2: Sample SD Model Specific Rubric (for Projectile Motion)

Criteria	**Scoring**	**Max Points**
Output Integrity		5
Names		5 (min 0)
Flows	Cloud → xVel Cloud → xPos Cloud → yVel Cloud → yPos	4
Variables	Initial conditions and other constants are properly expressed as variables: initX initY initVel initAngle gravity mass dragCoeff	7
Positioning	Elements are appropriately placed	4
Specific Model Rubric		
Initial Conditions	Reasonable values are set for each of the following, including any equations if further calculations are needed to transform the model parameters: 2pts - xPos 2pts - yPos 2pts - Initial Speed 2pts - Initial Angle 2pts - Drag Coeff 2pts - Mass 2pts - Gravity	14
Relationships	Variables may be renamed if model does not run (penalize in general rubric). If model still does not run, -5pts per misc. necessary element change for model to run. 2pts - Initial Pos 2pts - Initial Vel. 2pts - Correct Gravity 4pts - Acceleration to Velocity 4pts - Velocity to Position 4pts - Drag Coefficient affects Velocity 12pts - Numerical Correctness	30