Streamlining EABSS with Generative AI: Part 3

Sener Topaloglu, 28/07/25

1. Introduction

I have spent some time improving upon "Streamlining EABSS with Generative AI: Part 2".

I used the same virtual machine configuration as Streamlining EABSS with Generative AI: Part 2 - a Google Cloud n1-high-mem-4 machine[1] with 26GB RAM and 4vCPU cores. I attached an Nvidia T4 GPU[2] (which has 16GB GPU memory) to the virtual machine. With this configuration, I can locally run LLMs via Ollama[3] that quickly process input tokens and generate responses in a responsive manner. **In my case; 'locally' refers to on a cloud VM. However, if your hardware permits, every instance of 'locally' in this report can be re-interpreted as running on your own physical machine (again, using Ollama, or another LLM server).** If you are running these tasks regularly, I would consider procuring a physical GPU to be a worthwhile investment, and nevertheless, having a GPU attached to a cloud VM is also an option. Since the University has a partnership with Microsoft, a similar setup could be achieved by instantiating a virtual

[1] https://cloud.google.com/compute/docs/general-purpose-machines#n1_machine_types
[2] https://www.nvidia.com/en-gb/data-center/tesla-t4/
[3] https://ollama.com/

machine in Azure; the NCas_T4_v3-series[4] of virtual machines appears to use the same GPU, however, you may have to double-check with whom is responsible for cloud computing at the University for specifics. It is important to note that you do not need a lot of RAM (8GB should suffice) if you have a good GPU.

In this document, I use the term "small/medium-sized models" for models that have around 10B-15B parameters.

2. Important Ollama Context Length Information & Improving the Python EABSS Automation Script

Whether doing manual LLM testing via Ollama CLI or using the EABSS automation script (discussed in paragraphs below) or the OpenWeb UI (discussed in 9), the context length must be set via a variable (`OLLAMA_CONTEXT_LENGTH`) every time `ollama serve` command is used. The variable is stored temporarily, so setting it just once is not enough.

If you wish to manually use LLMs and test them, follow the steps per 11.1.

Example usage of all the automation scripts above can be found in 11.2.

Every set of instructions in the appendix contain the important steps to set the context length via the said variable.

I have made improvements to the Python script that automates the process of running EABSS scripts on LLMs that are served via Ollama. I added support for "verbose" output (to print the number of input & output tokens used in the previous interaction and a running total of input & output tokens used so far). Secondly, the context length configuration argument has been removed from the CLI, as I mentioned in the first paragraphs in this chapter, Ollama provides a variable that can be set each time the Ollama server is started. Finally, I removed the call to the Ollama API `pull(...)` method (which would fetch the model before initiating the chat), because this method cannot pull user-defined custom models. Removing this method call is not an issue because the first call to the Ollama API `chat(...)` method later in the Python script will start the model, if it is not already running. The latest version can be found in `eabss_automation_bot.py`.

[4] https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ncast4v3-series

I also wrote two new Python scripts to automate EABSS run-throughs for OpenAI ChatGPT models via the OpenAI API[5], and Google Gemini models via the Gemini API[6]. These scripts can be found at `chatgpt_eabss_automation_bot.py` and `gemini_eabss_automation_bot.py` respectively. The verbose output for these scripts may not output the exact number of tokens used as it was difficult to understand the REST responses from the respective APIs.

When any of the automation scripts are run, the conversation history is saved in a JSON file. The JSON filename contains the name of the model used, but not the hyperparameter values because the file names could get too long (this can be alleviated by using a custom Modelfile which allows you to set a custom model name - please see the Appendix - 11.3).

3. Improving the Test Scenario Prompts

In Part 2, I submitted 4 test scenario prompts, named `bass.txt`, `conway.txt`, `flu.txt` and `predator_prey.txt`. These prompts are text files that contain input values for compulsory user input per the EABSS script (such as {key-topic}). I refactored each of these prompts to use consistent British English spelling and added an input value for a key domain-related role, which will be injected in the string below:

Take on the "role" of a "{KEY DOMAIN RELATED ROLE HERE}" with experience in "Agent-Based Social Simulation". Memorise this role as {key-role1}.

I have included the latest versions of these test scenario prompts in the `test_prompts` folder.

4. Refining the Streamlining EABSS with Generative AI Script

4.1 Small/Medium Models Script

I decided to refine the Streamlining EABSS with Generative AI: Part 2 script (itself based on the script submitted as part of my summer internship). This new version of the script aims to:

- Fix as many typos as possible and enforce consistent formatting, e.g. always use numbered lists 1), 2), 3), etc. and capitalise each list item.

[5] https://openai.com/api/
[6] https://ai.google.dev/

- Reduce confirmations, such as “Do you understand?”, which accumulate within the context window.
- Generate less stereotypical debates.
- Generate ABSS experimental factors that are more relevant to the memorised {key-hypotheses} and memorised {key-objectives} by explicitly prompting the LLM to do so.
- Generate class diagrams that account for all memorised {key-UMLUseCases}, {key-experimentalFactors} and {key-outputs} by explicitly prompting the LLM to do so.
- Generate class diagrams with more comprehensive relationships by providing the correct relationship syntax in both the first prompt (which produces a draft class diagram) and second prompt (which improves upon the draft class diagram).
- Generate more accurate sequence diagrams by providing the correct Mermaid.js[7] syntax to implement loops, alternative paths, optional paths and parallel interactions.
- Incrementally build a GAML script in such a way that the implementation prompts can be extended and even more detailed in the future. The existing GAML generation prompts are too large to make any more additions, consequently they were split into smaller, more incremental prompts to make the implementation process more manageable. There are 6 new prompts:
    - First prompt: Instructs the model to become a GAML developer and use the contextual knowledge it has gathered. The LLM is then asked to generate a global block containing all global attributes, reflexes and code to instantiate the species.
    - Second prompt: Instructs the LLM to create the species and declare the species’ attributes.
    - Third prompt: Instructs the model to generate the pseudocode and steps to implement the logic for the methods inside the classes (which will become actions/reflexes inside species). The model is also instructed to think about the inter-species interactions necessary for each method.
    - Fourth prompt: Instructs the LLM to implement the pseudocode from the previous prompt, inside actions and reflexes in the correct species.
    - Fifth prompt: Instructs the model to generate an experiment block that accounts for manipulating {key-experimentalFactors}, displaying {key-outputs}, verifying {key-hypotheses} and thus meeting the {key-objectives}.

[7] https://mermaid.js.org/

- Sixth prompt: Asks the LLM to carefully scan through the GAML code generated at the previous stage, find anything missing and any mistakes and then implement the corrections and missing parts. This prompt also asks the model to reflect and improve upon the GAML script.

As described above, the broken-down implementation prompts allow for the model to "focus" on specific parts of the code. There is a GAML "scaffold" in each prompt that increases in complexity throughout the implementation process. Consequently, the implementation prompts should:

- Make for better {key-output} generation, e.g. text files, various charts, grids
- Encourage the model to generate all necessary species
- Reduce instances of stubbed actions/reflexes
- Better facilitate interactions between different species.

The prompts were refined via repeated testing and trial & error. Some of the techniques used in the implementation prompts are credited to Peer-Olaf Siebers' most recent LLM blog post which provides tips to improve the quality of code generation[8].

Generating pseudocode (in the third implementation prompt) acts as a stepping stone to generate actions and reflexes from method names as it adds detail to what would otherwise be just a list of methods in the class diagram. The resulting specification aims to provide the LLM with much stronger signal about the contents of each action & reflex. Another advantage of this approach is that it can expose semantic misunderstandings before any GAML code is written, thus helping to catch issues early; even individuals who are unfamiliar with GAML can understand the LLM output and provide corrections since it is closer to human language.

In the final implementation prompt, the LLM is instructed to check the most recent draft ("saved" as {key-gamlScriptDraft4}) for any missing code and any mistakes (such as stubbed actions/reflexes and missing interactions between species). The LLM is then asked to resolve these issues and reflect and improve on the GAML script. The final output (which should be as complete and valid as possible) is then memorised. The motivation for this step is that it provides the LLM with a chance to surface errors, such as methods that have been called but unimplemented, and gives an extra opportunity to implement any code that may prove useful.

In some places, I have used Markdown bold syntax `**...**` to emphasise pieces of text (and sometimes to increase the importance of instructions that were already in

[8] https://people.cs.nott.ac.uk/pszps/llm.html (exact link unattainable at the time of writing)

upper case) - I found this an effective way to get the LLM to pay attention to important information.

I have made other minor changes that have not been included in this document so not to digress. Please use `git diff` to see these clearly.

This script can be referred to as "Streamlining EABSS with Generative AI: Part 3 - Small/Medium Model Script" and can be viewed in MS Word and JSON formats in `streamlining_eabss_3_small_medium_model_script.docx` and `streamlining_eabss_3_small_medium_model_script.json` respectively.

4.2 Advanced Models Script

Due to the progress of ChatGPT models in natural language conversations as well as Markdown, Mermaid.js & GAML code generation, I have written another version that is dedicated to such models. The "Streamlining EABSS with Generative AI: Part 3 - Advanced Models Script" makes the following further changes to the Small/Medium Model Script:

- Removing confirmations such as "Do you understand?" and "Confirm you have memorised" in their entirety.
- Removing verbiage around explicitly differentiating each of the {key-stakeholders}.
- Removing statements such as "Only show the final, resulting Markdown file code from this prompt".
- Removing reminders to "Use a scientific tone" (and replacing it with a single instruction near the beginning of the script).
- Removing reminders to "replace the keys inside {} with their values" (and replacing it with a single instruction near the beginning of the script).
- Removing instructions to "avoid using
" and "IGNORE ALL space limitations" when creating Markdown tables.
- Removing instructions to: "remove all brackets from actor names", since the latest ChatGPT models are quite good at Mermaid.js generation
- Removing Mermaid.js syntax reminders for class diagram relationships
- Removing reminders to consider "including a start and stop transition for state diagram generation".
- Removing a reminder to use `sequenceDiagram` syntax during sequence diagram generation.
- Removing syntax reminders for generating loops, alternative paths, optional paths and parallel interactions in sequence diagrams.
- Removing reminders on basic GAML syntax conventions, such as initialising variables with `<-` and using the `model` keyword.

- Removing reminders pertaining to how to write actions & reflexes
- Removing the reminders to fully implement the methods that are referenced in the GAML script.
- Removing examples of experimental factors, hypotheses and output. Also removing instructions to correctly implement logic in the appropriate code blocks (the latest LLMs do not have to be reminded of this).
- Removing reminders to not generate boilerplate code in the penultimate prompt. (since there is already such a reminder in the final prompt).

This script can be viewed in MS Word and JSON format in `streamlining_eabss_3_advanced_model_script.docx` and `streamlining_eabss_3_advanced_model_script.json` respectively.

An important note: while the script asks the LLM to imitate certain hyperparameter settings, such as temperature and top_p, it is a best practice to set them (if possible). The Python EABSS automation scripts already support setting some hyperparameters (including temperature and top_p) via CLI arguments.

4.3 Next Steps

I have aggregated a list of improvements that can be made to improve the scripts further:

- [Formatting error in Small/Medium Model Script only] The second Markdown prompt (immediately following the prompt that instructs the LLM to define an aim and memorise as {key-aim} contains a bulleted list of 3 items that are not capitalised. This is a minor formatting error that should not affect the LLM response, but it is good to correct it for consistency.
- [Formatting error in Small/Medium Model Script only] The 9th and 13th prompts (both in “Problem Statement”) omit the statement “Make sure to replace the keys inside "{}" with their values.”. While this hasn't been a problem for Gemma 3 12B models, there have been occasions where Mistral NeMo has printed {key-role1}, for example, rather than its actual value. You may wish to consider adding the statement to both prompts.
- [Formatting error in both scripts] The prompt to generate the model scope Markdown table contains a numbered list. At the end of the first item, there is no full stop. Again, this is a minor formatting error that should not affect LLM responses, but it is good to correct it for consistency.
- Better prompting could be employed in “Analysis: Study Outline” to guarantee the experimental factors and outputs are chosen to verify every hypothesis and meet all the objectives – perhaps the following statement would suffice: “CAREFULLY THINK ABOUT WHICH EXPERIMENTAL FACTORS/OUTPUTS ARE

NECESSARY TO FACILITATE VERIFICATION OF THE MEMORISED {key-hypotheses} AND MEET THE MEMORISED {key-objectives}". Alternatively, the GAML scaffolds in the implementation prompts could be modified; please see the last item in this list.

- A step towards slimming down the Streamlining EABSS with Generative AI: Part 2 - Advanced Model Script could be to reduce the two-step Mermaid.js diagram generation to one prompt by removing step 1 of 2, which is generating a draft diagram.
- It may be useful to explicitly instruct the LLM to use the memorised user stories when generating the state diagrams.
- Explicitly instructing the LLM to store any helpful global variables and functions in the artificial lab class (in the class diagram generation prompt) could be useful for downstream GAML generation.
- The first GAML implementation prompt asks the LLM to become a GAML software developer; you may wish to assign a {key-role5} for this, although I am not sure if this would be allowed per EABSS rules.
- If you wish to get more accurate domain representation in each GAML species, e.g. a predator species that hunts, moves and reproduces *even when such methods have not been considered by the LLM*, one approach could be to include a statement such as "ensure semantic fidelity with memorised {key-modelScope}" in the prompts.
- Prompting the LLM to include performance measures in GAML generation.
- For prompts that generate GAML drafts, I have used the word "save" rather than "memorise", because "saving" is akin to the process of a developer saving and then later modifying a GAML file in the IDE. This approach was preferred over "memorising" multiple GAML file drafts; however, it could be argued that "memorise" be used instead, to be consistent with the other Mermaid.js diagram draft prompts. I do not believe there would be a difference in LLM performance.
- Splitting the final implementation prompt into two prompts could result in fewer stubbed actions/reflexes, fewer missing experimental factors, fewer missing output charts etc. by instructing it to: 1) list all the mistakes/shortcomings of the GAML code it has generated in {key-gamlScriptDraft4}. 2) fix the mistakes and implement the shortcomings it identified in prompt 1.

Although I invested significant effort and time to not introduce errors in the scripts, some inevitably crept in due to the trial-and-error nature of refining. I did not realise these until after the testing was completed. As testing takes a long time, and I was out of time, I was unable to make the 3 formatting corrections at the top of the list.

5. Gemma 3 12B

5.1 Initial Testing & Parameter Tuning

I wanted to compare a recently released model with the Mistral NeMo model to understand the current state of small/medium-sized LLMs that I can run locally (or, in my case, on a virtual machine). Gemma 3[9] is an open-weight LLM model family recently released by Google. The 12B parameter variant of the model was particularly promising, showing good benchmark performance on the LMArena[10] Overall Leaderboard; to the best of my research, it is the highest ranked open-weight model that is confirmed to have sub-15B parameters, as of 25/7/25. 15B parameters is an approximate limit for my VM to run a quantised version of a model with a reasonable context length in T4 GPU memory and have a responsive conversation.

I used the Quantization-Aware Training (QAT) version of the 12B model[11]. QAT is a method used in deep learning model training to reduce model size and speed up inference without sacrificing a lot of accuracy[12]. This is achieved by simulating the effects of quantisation during training; the model simulates quantisation (e.g. INT8 or INT4) operations (e.g. rounding) during forward propagation, but importantly, backpropagation still uses full-precision gradients. The QAT version utilised ~15GB of memory (8.9GB for the model itself + remaining for a 32K context window).

Initial testing of the model showed promise as it was able to generate sizeable amounts of syntactically correct Mermaid.js and GAML code, however, the model struggled to produce a cohesive, complete source code that was semantically correct (i.e. methods were omitted from class diagrams, methods were implemented but not called, or placeholder methods were generated). I have provided evidence of this initial testing, which can be found in the `test_dump` folder (with timestamps from 8th July). It is important to note that initial testing was conducted while the scripts were being refined, due to time constraints.

Initial testing was conducted without changing any of the hyperparameters. The model ‘s params file[13] states the following default values:

- `temperature`: 1
- `top_k`: 64

[9] https://deepmind.google/models/gemma/gemma-3/
[10] https://lmarena.ai/leaderboard
[11] https://ollama.com/library/Gemma3:12b-it-qat
[12] https://www.ibm.com/think/topics/quantization-aware-training
[13] https://ollama.com/library/Gemma3:12b-it-qat/blobs/3116c5225075

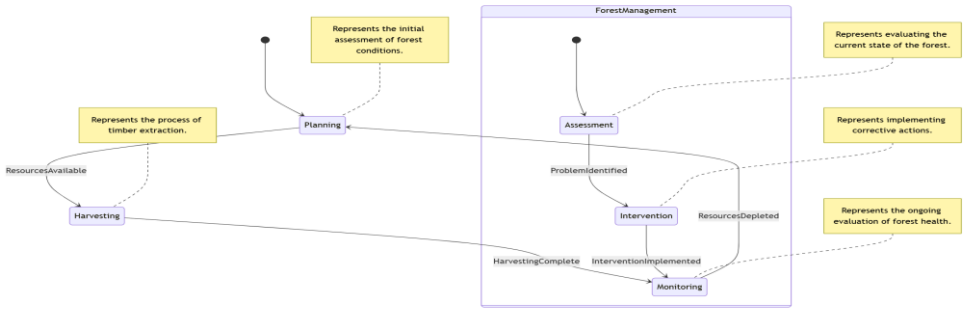
- `top_p`: 0.95

All other hyperparameters use the default values specified by Ollama (please see the default values in the Spring Boot API documentation[14], unfortunately, there isn't a similar documentation for their Python API).

I then proceeded to reduce the temperature of the model from 1 to 0.6, by following the steps outlined in the appendix. Lowering the temperature makes the model likely to produce more deterministic output by picking "safer" tokens[15]. My intuition led me to the notion that the extra determinism has the following advantages (although thorough testing and academic cross-referencing would be necessary to verify this):

1. LLMs are more likely to emit all the methods given in the prompts, as they are "safe" tokens
2. LLMs are more likely to stick to consistent names, and by extension are more likely to call the methods that have been defined without hallucinating other method names.
3. As I used a GAML "scaffold" in my prompts, the tokens that constitute my template would also be considered "safe" tokens, thus my scaffold would be followed more closely.

Reducing the temperature yielded some promising results, such as larger state diagrams (an example from an epidemic flu test scenario, is provided in Figure 1). However, the problems mentioned in initial testing were still present, albeit less severe. I have provided evidence of testing at `test_dump/gemma3_12b_bitqat_0.6temp_latest_test_20250710-151923.json`. Please see a state diagram generated during this test in Figure 1.



[14] https://docs.spring.io/spring-ai/reference/api/chat/ollama-chat.html
[15] https://www.promptingguide.ai/introduction/settings

Figure 1: UML State Diagram (`test_dump/gemma3_12b_bitqat_0.6temp_latest_test_20250710-151923.json`) (for a predator-prey scenario; "Forest Manager" agent)

Consequently, I decided to reduce the top_k and top_p values from 64 to 40 and from 0.95 to 0.8 respectively. The motivation for this was that tightening the nucleus cutoff (top_p) and k-sampling cutoff (top_k) "forces" the model to stick very closely to its highest-probability next tokens (i.e. the completions it's most confident in)[16] [17]. In practice, I expect the higher-probability paths to include the tokens necessary to invoke each method once it has defined, for example. I expect the lower-probability tails to contain the creative detours or omissions (e.g. skipping a method call, inventing an unexpected method). By dropping the top_p and top_k values, the tails are pruned away, and the model must pick from an even smaller set of "trusted" tokens. As a result, the model is more likely to follow through on explicit instructions provided by the user (e.g. "call every method you just declared"). To summarise, I used the following hyperparameter settings:

- `temperature`: 0.6
- `top_k`: 40
- `top_p`: 0.8

The results from testing these settings (located in the `test_dump` folder) looked promising, and I was running out of time, so I decided to proceed with these settings for final testing of the "Streamlining EABSS with Generative AI: Part 3 - Small/Medium Model" script.

5.2 Final Testing & Comparisons with Mistral NeMo

I used the Mistral NeMo model (from Streamlining EABSS with Generative AI: Part 2), to compare the performance of the tuned Gemma 3 12B model and collect more test samples. For the NeMo model, I provided the Python EABSS automation bot with the following hyperparameter tunings via CLI arguments (which are the same as those from Streamlining EABSS with Generative AI: Part 2):

- `temperature`: 0.4
- `repeat_penalty`: 1.05

Please note: The .json filenames for Mistral NeMo test evidence does not contain the hyperparameter values like the Gemma 3 12B ones, because I created the Gemma 3

[16] ttps://www.promptingguide.ai/introduction/settings
[17] https://ivibudh.medium.com/a-guide-to-controlling-llm-model-output-exploring-top-k-top-p-and-temperature-parameters-ed6a31313910

12B model with a custom Modelfile that contains the hyperparameter settings, so I didn't have to set them via the EABSS automation CLI arguments every time I ran the script.

For the remainder of this section, 'NeMo' and 'Gemma 3' refer to the tuned versions of the Mistral NeMo and Gemma 3 12B models.

I ran each test prompt scenario (predator-prey, epidemic flu, Conway's Game of Life and Bass Diffusion) once for each model with the Streamlining EABSS with Generative AI: Part 3 – Small/Medium Model Script. It takes approximately 1 hour to run-through the script on both models (NeMo can take 10-15 minutes longer in some cases). All copies of the tests can be found as JSON files in the `final_tuned_nemo` and `final_tuned_gemma3` folders. As JSON will escape things like new lines & tabs in the LLM output, I recommend unescaping, with a tool like https://www.freeformatter.com/json-escape.html to save time.

Observations from testing are as follows;

5.2.1 Debate Simulation

The ability to generate non-stereotypical debates using the current prompts is limited for both models. The refinements made to the debate prompts did not yield the expected results; this may be due to the relatively low temperature settings used. This could be solved by making a minor change to the Python EABSS automation to gradually reduce the temperature as the script progresses. Doing so could also allow for even more creative {key-context} generation while keeping the standard of GAML generation high.

5.2.2 Devising Experimental Factors, Outputs & Objectives

Both models can struggle with devising the most suitable experimental factors & outputs to verify the hypotheses and meet the objectives - this is probably the single biggest weakness of both models and propagates down to class diagram and GAML generation. I discuss a potential solution in 4.3.

5.2.3 Citing Academic Papers

Gemma 3 is slightly better, albeit not perfect, at citing papers; a good example from a Conway's Game of Life scenario, is provided in Figure 2 below.

Model Type	Description	Harvard Style Reference
Social Model: Social Network Analysis (SNA)	Utilizes graph theory to analyze relationships and interactions within the microbial community, identifying key influencers and patterns of information flow. Focuses on network structure and its impact on community dynamics.	Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications*. Cambridge university press.
Behavioural Model: Reinforcement Learning (RL)	Agents learn optimal strategies through trial and error, receiving rewards or penalties based on their actions. Models adaptive behaviour and decision-making in response to environmental feedback.	Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
Psychological Model: Cognitive Dissonance Theory	Models the psychological discomfort experienced by agents when their actions contradict their beliefs, leading to behavioural adjustments to reduce dissonance.	Festinger, L. (1957). *A theory of cognitive dissonance*. Stanford university press.
Technical Model: Cellular Automata (CA)	Provides a framework for simulating spatial and temporal dynamics based on simple rules governing the state of individual cells and their interactions with neighbours.	Wolfram, S. (2002). *A new kind of science*. Addison-wesley.

Figure 2: Implementation models and papers cited by Gemma 3 (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-203044.json`*) (Conway's Game of Life scenario)*

5.2.4 Markdown Generation

Gemma 3 appears to be better than NeMo at generating Markdown code, as there were occasions where NeMo could not create the correct separator between the heading and body of tables, the Gemma 3 model was more consistent at this.

5.2.5 Use Case Diagram Generation

NeMo covers all UML use cases in the use case diagrams across 3 out of 4 test scenarios. On the other hand, Gemma 3 consistently covers UML use cases in the use case diagrams across all test scenarios (except for a renamed use case in one scenario). One interesting observation was that both models independently devised extra use cases relevant to the actors and context. Please find a (colourful!) example below in Figure 3 from an epidemic flu scenario. Both models are good at generating syntactically valid use case diagrams.

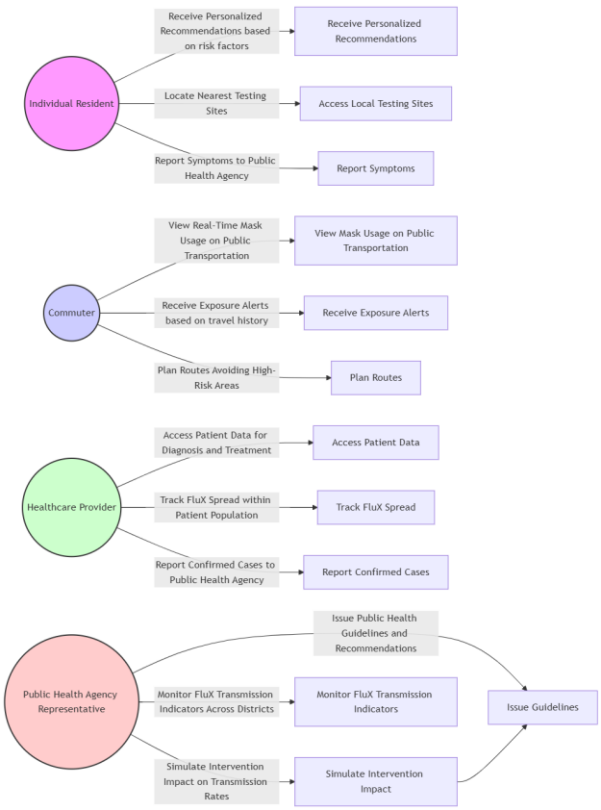


Figure 3: UML use case diagram generated by Gemma 3 (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-180957.json`) (epidemic flu scenario)

5.2.6 Class Diagram Generation

Syntactically, only one class diagram generated by the Gemma 3 model (a Bass Diffusion scenario at `final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-215140.json`) required minor fixes to use a valid relationship symbol. The Gemma 3 model generated classes for every actor and physical environment category except the "Patient Home" in the Bass Diffusion test scenario. One physical environment category, "Resource Availability" was renamed as "Resource Provider" in the class diagram generated by the Gemma model for the Conway test scenario (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-203044.json`).

Conversely, NeMo-generated diagrams had slightly more (but still minor) syntax issues; such as defining enums inside attributes rather than creating a separate class and annotating it with `<<enumeration>>`. The fixes do not take a long time to implement. Semantically, NeMo appears more promising because of the generated class diagrams being larger on average (by number of lines and classes). The NeMo model generated classes for every actor and physical environment category; one strange instance was

the “Habitat Density” physical environment category which did not have its own `class` block but was instead referenced when creating a relationship, Mermaid.js rendered this as an empty class. This can be observed at `final_tuned_nemo/mistral-nemo_latest_test_20250722-070116.json`.

Please note: I checked whether NeMo and Gemma 3 implemented every physical environment and actor category, I did not have the time to look in detail if the specific physical environment sub-categories were accounted for. If the actor or physical environment category names were plural e.g. “CoralPolyps”, I considered a class with a singular name (in this case “CoralPolyp”) as an implementation of that actor/physical environment category.

Both models have demonstrated ability to form relationships, and NeMo demonstrated this ability even with a large number of classes. In Part 2 of this project, I added relationship syntax reminders and subsequently believed I had solved the issue of generating accurate relationships, however testing in Part 3 shows there still is some progress to be made (an example being the NeMo model, Bass diffusion test scenario at `final_tuned_nemo/mistral-nemo_latest_test_20250721-203513.json`). It is important to state that the ArtificialLab is not perfectly generated by either model; experimental factors and/or outputs can be omitted from the class by both LLMs. Both LLMs can also omit some methods in that correspond to UML use case. An example of class diagram generated by NeMo during an epidemic flu test, has been provided in Figure 4.

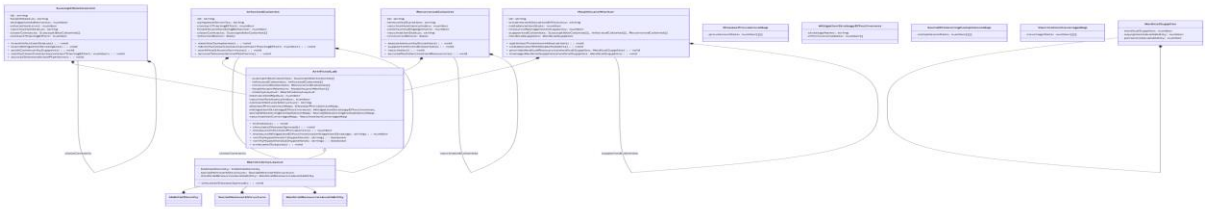


Figure 4: UML class diagram generated by NeMo (`final_tuned_nemo/mistral-nemo_latest_test_20250722-122313.json`) (epidemic flu scenario)

5.2.7 State Diagram Generation

Gemma 3 generated some interesting state diagrams that demonstrated its ability to “think” about how the actor fits within the context of the problem and what potential

flows through the state diagram could look like. I have included an example of this in Figure 5. The Gemma 3 model generates syntactically valid code aside from this in 2 of 4 examples and requires only minor corrections. Conversely, NeMo has been observed sharing a single start state, `[*]`, between actors in 3 of 4 test scenarios (exception being `final_tuned_nemo/mistral-nemo_latest_test_20250722-070116.json`). As I look at the results from Part 2 of this project at the time of writing, this is a persisting issue.

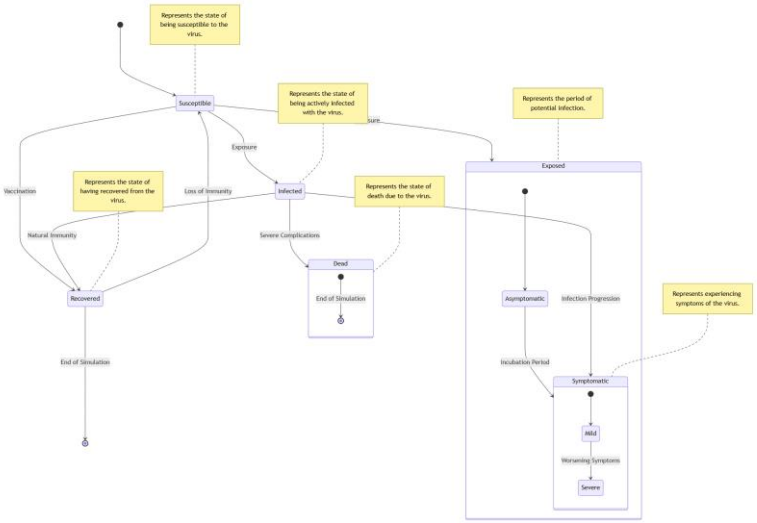


Figure 5: UML State diagram generated by Gemma 3 (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-180957.json`) (for "IndividualResident" actor in epidemic flu scenario)

5.2.8 Sequence Diagram Generation

For a reason unknown, both models struggle with using `actor` instead of `participant` in sequence diagrams, and activations & deactivations of actors are not always present. Only one Gemma 3-generated sequence diagram (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-180957.json`) had a syntax issue; it tried to implement activating/deactivating, but some participants are deactivated early, and this is a quick fix. No sequence diagram generated by Gemma 3 accounted for all UML use cases. Two sequence diagrams generated by NeMo accounted for all UML use cases; in the epidemic flu scenario (`final_tuned_nemo/mistral-nemo_latest_test_20250722-122313.json`) and Bass Diffusion scenario (`final_tuned_nemo/mistral-nemo_latest_test_20250721-203513.json`). Some issues that were understood to be solved in Part 2 still exist; implementing notes in the correct places (a very quick fix), and the use of dotted lines *and* solid lines in the

sequence diagrams (again, a very quick fix). I did not have the time to examine the order of interactions in each diagram.

5.2.9 Overall Mermaid.js Diagram Generation

Overall, I think that Gemma 3 is better than NeMo at generating syntactically correct Mermaid.js code; however, this is probably because it generates diagrams that are generally less detailed. The logical content of Gemma 3-generated diagrams can vary, but it can be said that NeMo performs better at devising the logic for aspects such as relationships in class diagrams, interactions in sequence diagrams and covering use cases in sequence diagrams. This edge provides NeMo with an advantage in generating GAML code; being able to synthesise relationships means more methods can be implemented, and by extension, a more comprehensive GAML script can be written. Modifying the `temperature` of the Gemma 3 model could facilitate better class diagram generation.

5.2.10 GAML Generation

As seen in Figures 6 and 7, the reflex generated by Gemma 3 is syntactically correct and uses the "ask" statement to perform interspecies interactions. The Predator species generated by the Gemma 3 model was missing relevant methods, such as reproducing, a use case of the Predator actor. On the other hand, the NeMo actions encapsulate more logic - more elaborate conditional statements as well as methods that are important for the Conway's Game of Life scenario (reproducing, dying, reviving) - however, the generated code is not syntactically correct.

```
reflex hunt {
    if (preyPopulation > 0) {
        ask Prey {
            if (distance(self, here) < territorySize) {
                energyLevel <- energyLevel + 10;
                preyPopulation <- preyPopulation - 1;
            }
        }
    } else {
        energyLevel <- energyLevel - 5;
    }
}
```

Figure 6: An example reflex generated by Gemma 3 (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250721-190342.json`) (from "Predator" species in predator-prey scenario)

```
action reproduce() {
  if (alive) {
    if (neighbourCells.neighbours.count(x -> x.alive && x.growthRate > growthRate * reproductionRate) > 0) {
      alive <- false;
    } else if (neighbourCells.neighbours.count(x -> x.alive && x.growthRate <= growthRate * reproductionRate) > 0) {
      alive <- true;
      reproductionRate <- reproductionRate * 1.1;
    }
  }
}

reflex die() {
  alive <- false;
}

reflex revive() {
  if (alive) {
    alive <- false;
  } else if (interventionAgent.interventionType == "Revive") {
    alive <- true;
    reproductionRate <- reproductionRate * 1.5;
  }
}
```

Figure 7: Example action and reflexes generated by NeMo (`final_tuned_nemo/mistral-nemo_latest_test_20250722-070116.json`) (from "CoralPolyp" species in a Conway's Game of Life scenario)

Empirically speaking, NeMo implements more GAML species from the class diagrams; this may be due to the higher parameter count or a better decoding capacity of NeMo due to mistuning of the Gemma 3 model hyperparameters. NeMo is better at generating the GAML species from the physical environment categories than Gemma 3 (as shown in Table 1 below). One possible remedy is to raise the top_p value rather than reduce it – a wider nucleus stops premature truncation, helping with thoroughness. One interesting research question pertaining to understanding why NeMo can generate more logic is: *Does NeMo demonstrate a higher degree of latent chain-of-thought reasoning?*

Gemma 3 has been observed outputting stubbed actions/reflexes. Conway's Game of Life scenario testing (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250722-203044.json`) is a good example of this. This was a problem that I encountered during some preliminary testing with NeMo (in the `before_final` folder), however, NeMo generated stubs containing lots of pseudocode and the occurrence of stubs reduced in the final NeMo testing (this is evident in the `final_tuned_nemo` folder).

As a result of its tendency to generate more species from classes and generate more concrete implementations of actions and reflexes, NeMo has shown that it can generate larger GAML scripts (all were 250+ lines, and the Bass Diffusion GAML script was almost 400 lines long). Needless to say, large GAML scripts do not necessarily guarantee that all the logic is correct, however, it does show the ability to generate GAML code of such size.

Therefore, I recorded the number of methods generated for each LLM-test scenario combination, in Table 1 below, to better examine the semantic correctness of the generated GAML scripts. Each line in a table cell represents a species (either an actor or physical environment category). An arrow is used to separate the number of actions/reflexes in that species that have a sizeable implementation (left) from the number of methods that are stubbed or have some non-ideal implementation that does not meet the requirements of that method (right). The order of actors and physical environments in each cell is deliberate, this table was constructed by going through each GAML script top-to-bottom and investigating each species that corresponds to an actor or physical environment category. Some methods were awarded .5 (if I was unsure whether the reflex/action was complete) and the scores are approximations, because to be entirely sure, all corrections must be implemented in the code and then experiments must be run (this was unfeasible given my time frame). I have also made a brief note on how important the missing/incorrect implementations were inside parentheses.

Table 1:

	NeMo	Gemma 3
Predator-prey	ACTOR: 4 => 1 (IS USED TO CALCULATE OUTPUT) ACTOR: 1 => 1 (NOT IMPORTANT) ACTOR: 2 => 1 (NOT IMPORTANT) PHY ENV: 1 => 0 PHY ENV: 1 => 0 ACTOR: 0 => 2 (FIRST IS USE CASE AND SECOND IS USED TO CALCULATE THE SAME OUTPUT AS FIRST ACTOR)	ACTOR: 1=>1 (IS A USE CASE) ACTOR: 2 => 0 ACTOR: 1 => 1 (OR 0.5 => 1 BECAUSE OF BASIC IMPLEMENTATIONS OF USE CASES) ACTOR: 0 => 0
Flu	ACTOR: 3 => 1 (SEMANTICALLY IMPORTANT IN TRANSITION FROM SUSCEPTIBLE -> INFECTED) ACTOR: 4 => 0 ACTOR: 4 => 0 ACTOR: 4 => 0.5 (SIMILAR LOGIC AS ANOTHER METHOD THAT EFFECTS	ACTOR: 2 => 0 ACTOR: 1 => 1 (IS A USE CASE METHOD) ACTOR: 1 => 1 (IMPORTANT FOR EPIDEMIC FLU WITH RECOVERY) ACTOR: 1 => 2 (1 IS A USE CASE METHOD, OTHER IS RELATED TO HYPOTHESES TESTING BUT IN THE WRONG SPECIES)

	EXPERIMENTAL FACTOR VALUE) ARTIFICIAL LAB [SHOULD NOT BE IMPLEMENTED]: 0 => 6 (SHOULD BE A GLOBAL REFLEX TO UPDATE EXPERIMENTAL FACTORS, VERIFY HYPOTHESES CODE IS INCORRECT, OTHER METHODS ARE NOT NECESSARY)	PHY ENV: 1 => 1 (IMPORTANT FOR HYPOTHESIS) PHY ENV: 0 => 1 (EXPERIMENTAL FACTOR)
Conway's Game of Life	ACTOR: 3 => 0 ACTOR: 3 => 0 ACTOR: 2 => 0 ACTOR: 2 => 0 PHY ENV: 1 => 0 PHY ENV: 1 => 0	ACTOR: 0.5 => 1 (IS A USE CASE METHOD) ACTOR: 0 => 2 (BOTH ARE USE CASE METHODS) ACTOR: 0 => 2 (BOTH ARE USE CASE METHODS) ACTOR: 0 => 2 (BOTH ARE USE CASE METHODS) PHY ENV: 0 => 0 PHY ENV: 0 => 0
Bass	ACTOR: 4 => 0 (OR 3.5 => 0 DEVICE REVIEWS ARE READ BUT NOT ACTED UPON) ACTOR: 2 => 2 (1 IS A USE CASE METHOD) ACTOR: 3 => 1 (OR 3 => 0.5) (COMMENT TELLING USER TO COPY & PASTE CODE FROM SOMEWHERE ELSE IN FILE) (NOT IMPORTANT) ACTOR: 2 => 2 (OR 2 => 1.5) (1 IS A USE CASE METHOD) PHY ENV: 2 => 0 PHY ENV: 2 => 0	ACTOR: 4 => 1 (OR 3.5 => 1; EXAMPLE FROM SCAFFOLD HAS BEEN INCLUDED AND ONE METHOD BODY DOES NOT HAVE A NAME) ACTOR: 0.5 => 0 (OR 0 => 0.5. IS A USE CASE METHOD WITH SOME STUBS) ACTOR: 0.5 => 0 (OR 0 => 0.5. IS A USE CASE METHOD WITH SOME STUBS) ACTOR: 0 => 1 (IS A UML USE CASE METHOD)

In the generated GAML scripts, there were a number of calls to actions that were not actually implemented, this was more common for the NeMo model. Please find these detailed in Table 2 below, it is important to note that a number of the methods listed are for performing UML use cases.

Table 2:

	NeMo	Gemma 3
Predator-Prey	4 calls to unimplemented calculateEthicalImpactScore Call made to contributeToNutrientCycling but not implemented 1 Call made to unimplemented calculateNewTerrain	N/A
Flu	TrackInfectionStatus and learnMitigationStrategies are implemented but not called. Both should be reflexes. notifyCloseContact called twice but not implemented. requestVaccination called but not implemented. allocateTreatments called but not implemented. shareDataAndDiscussStrategies, requestCollaboration, allocateMedicalSupplies and requestMedicalSupplies are called but not implemented.	N/A
Conway's Game of Life	N/A	N/A
Bass	displayExperienceDetails called but not implemented. CalculateEngagementMetrics called in 3 places but not implemented.	N/A

	retrievePeers called but not implemented. RetrieveGamingConsolePrice called but not implemented. PurchaseVRDevice called but not implemented. CalculateInfluenceScore called but not implemented.	

My explanation for the results above is that NeMo attempts to be more comprehensive with GAML script generation (by generating more methods from the class diagrams), thus there is a higher chance that calls are made to unimplemented actions. Gemma 3 seems to be better at connecting species via method calls (i.e. it generates fewer methods and less complete method bodies when compared to NeMo, but it also connects species better in the few methods it does implement). I believe that small/medium-sized LLMs do not yet know exactly when to use `action` or `reflex` - there are instances of actions being implemented but not called (because they should really be reflexes) - manually refactoring some of these actions into reflexes would alleviate this problem.

Few experimental factors and outputs were implemented correctly in the `experiment` blocks without corrections (across testing for both models), hence it was difficult for the hypotheses to be verified, and objectives were not always met.

I believe the shortcomings above can be alleviated with further improvements to the implementation prompts. I included these improvements in the list in 4.3.

5.3 Correcting & Testing Sample GAML Models

After completing testing, I moved onto correcting and running two GAML scripts; the predator-prey model generated by Gemma 3 (`final_tuned_gemma3/gemma3_12b_itqat_0.6temp_40topk_0.8topp_latest_test_20250721-190342.json`) and Bass Diffusion model generated by NeMo (`final_tuned_nemo/mistral-nemo_latest_test_20250721-203513.json`).

Please find the UML diagrams in the figures below. When necessary, I use “before” and “after” diagrams; the “before” diagrams are the ones generated by the LLM, with only the necessary syntax corrections to get the diagram to render (and are given inside

parentheses in the figure captions). I have created an “after” diagram to the best of my ability if the “before” diagram must be modified to meet the diagram requirements specified in the prompts. I have provided the code generated from these “after” diagrams and these can be found as `.mermaid` files in the `final_tuned_nemo` and `final_tuned-gemma3` folders.

5.3.1 Predator-Prey Model

At the end of this subchapter, I have also provided screenshots of the predator-prey experiment running in the GAMA IDE[18].

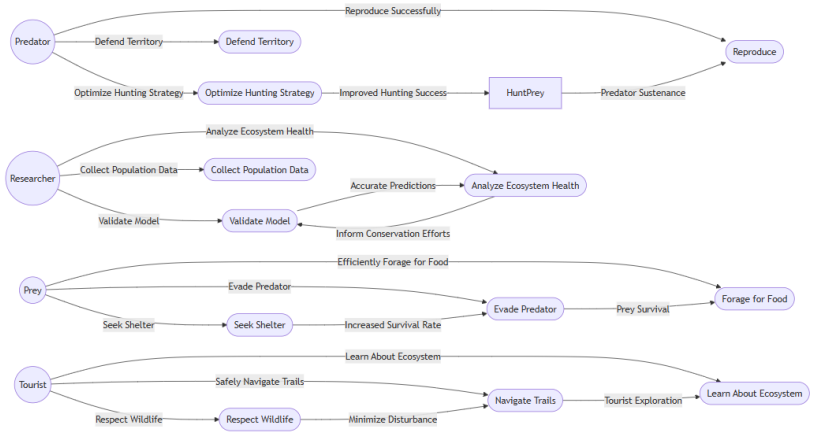
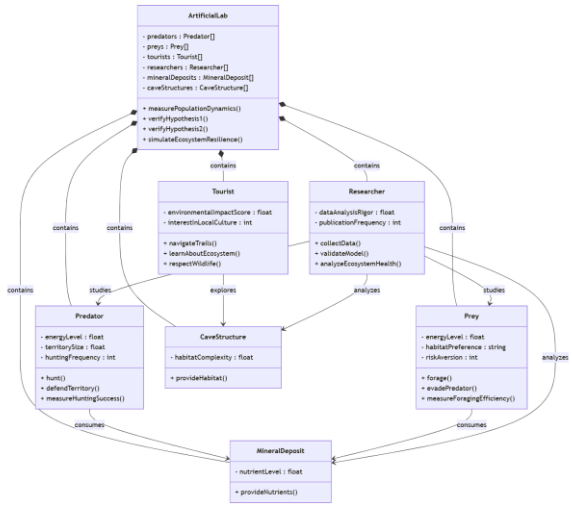
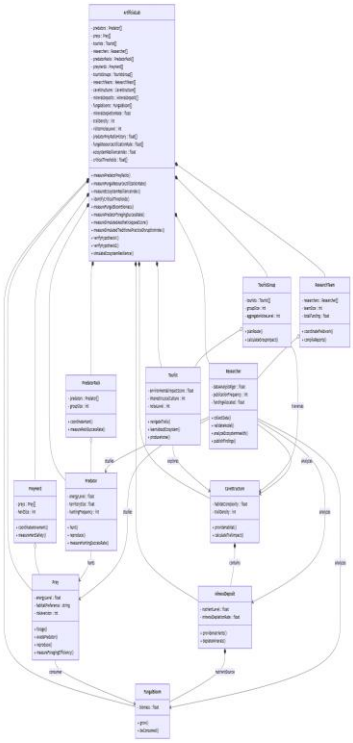


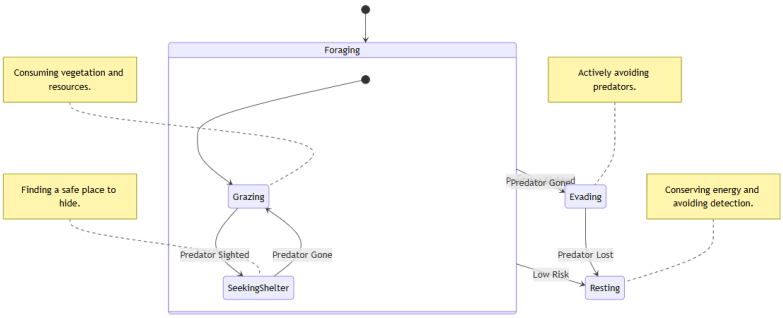
Figure 8: UML Use Case Diagram (syntax corrections made in “before”: whitespace removed from use case variable names). No “after” required, all conditions are satisfied.



[18] https://gama-platform.org/



Figures 9 and 10: UML Class Diagram Before & After



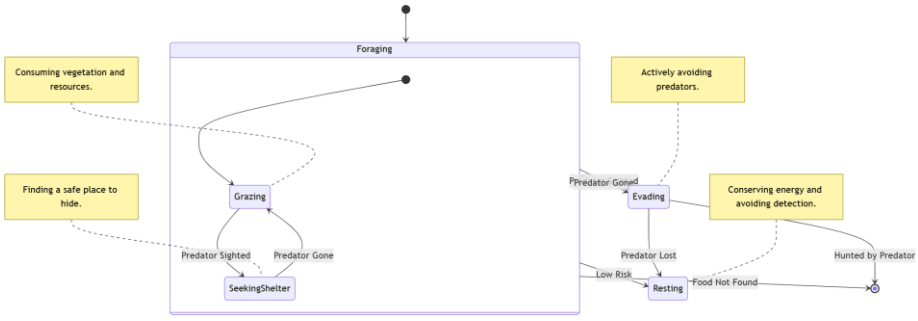
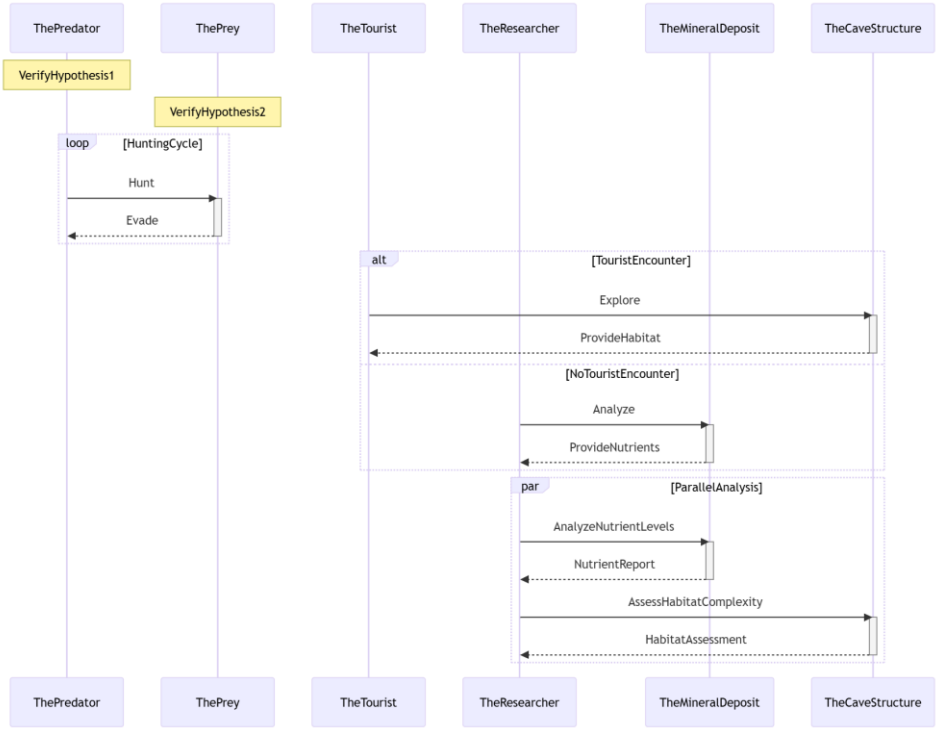
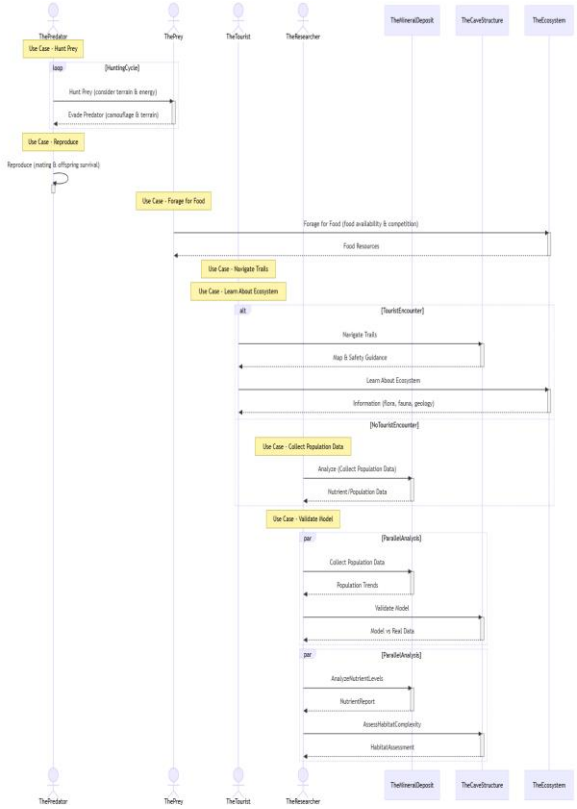
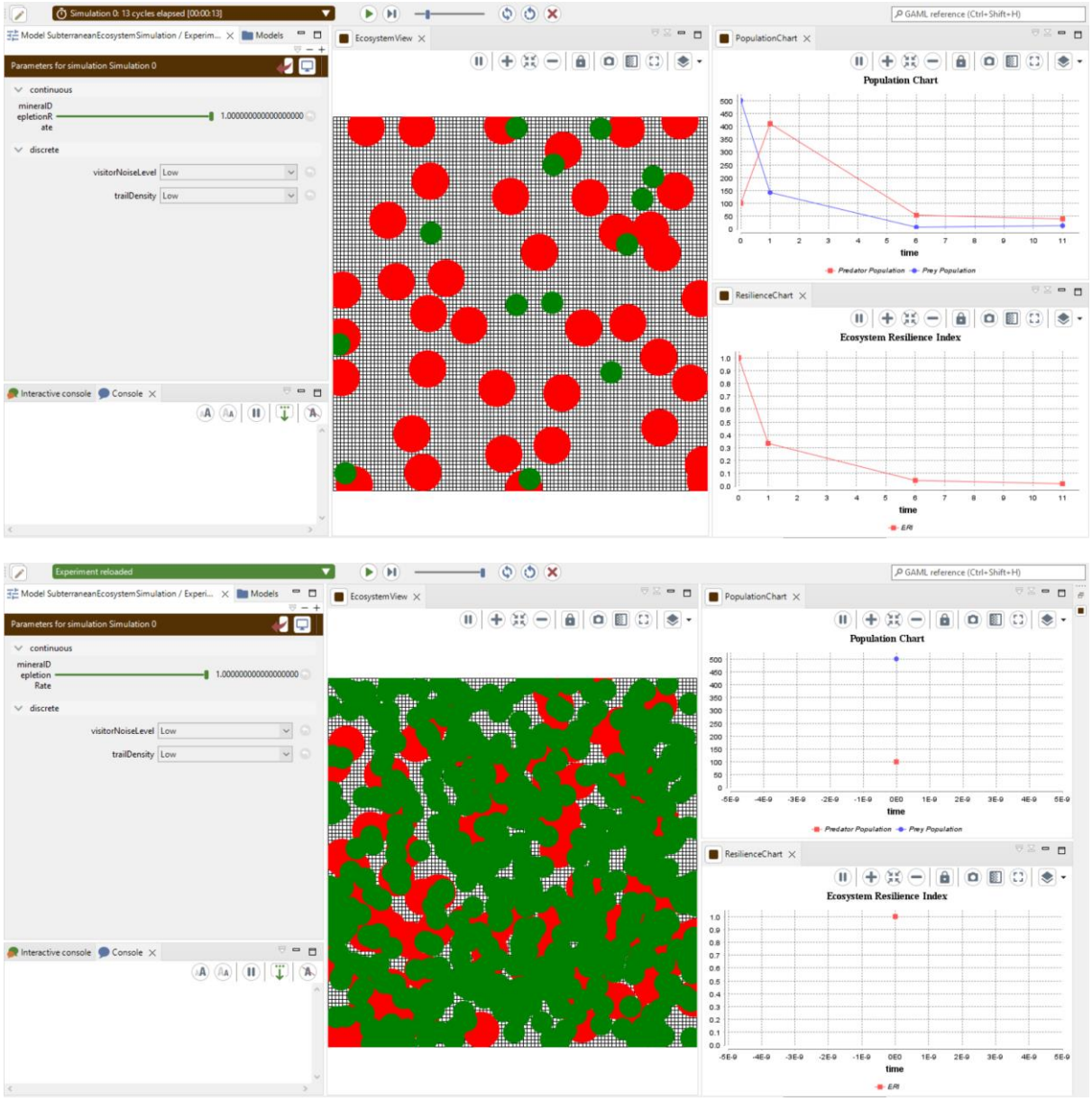


Figure 11: UML State Diagram for "Prey" Actor Before & After (syntax corrections made in "before": remove `state` blocks surrounding notes) (note: a couple of labels are positioned over each other).





Figures 12 and 13: UML Sequence Diagram Before & After



Figures 14 and 15: Screenshots of working GAML code (post-corrections)

When correcting the GAML script generated by the LLM, I focused on the implementing the UML use cases, experimental factors and output, rather than trying to fulfill "after" diagrams, because of the sheer amount of detail. The working GAML script can be found at `final_tuned_gemma3/predator_prey.gaml`. This GAML script demonstrates that the Streamlining EABSS with Generative AI: Part 3 – Small/Medium Model Script, together with the tuned Gemma 3 model, can provide a foundation to produce working agent-based social simulation models.

5.3.2 Bass Diffusion Model

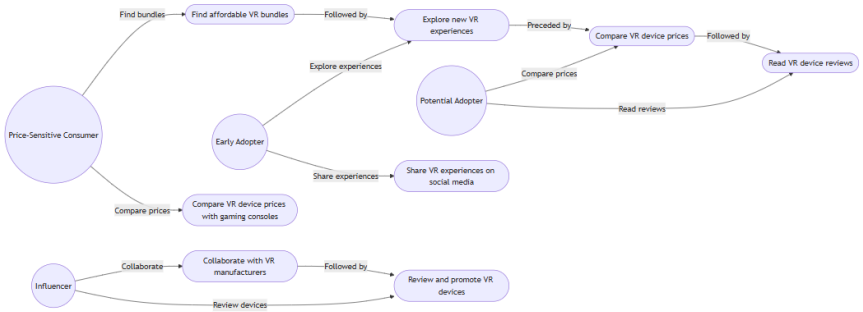
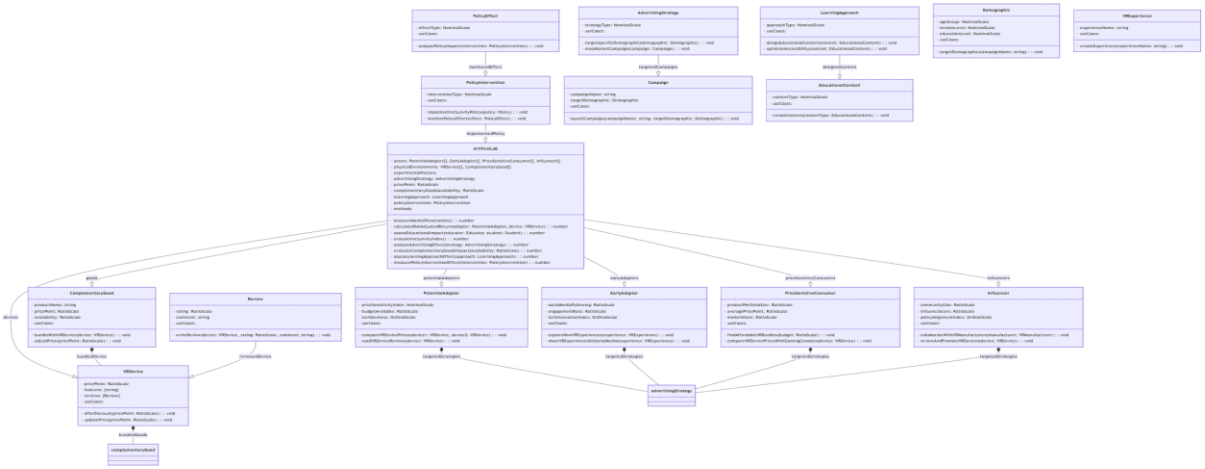
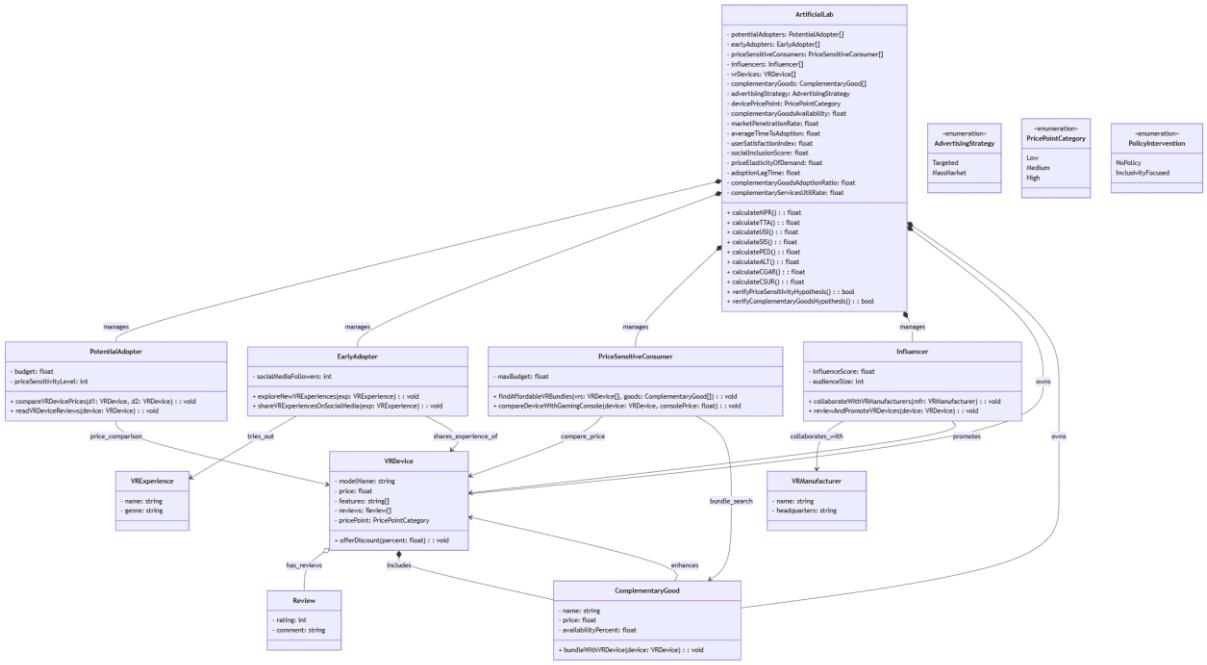
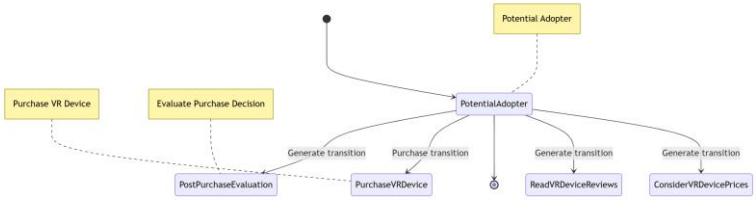


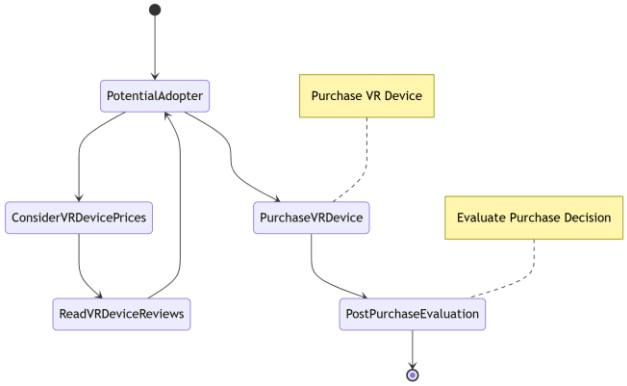
Figure 16: UML Use Case Diagram. No "after" required, all conditions are satisfied.



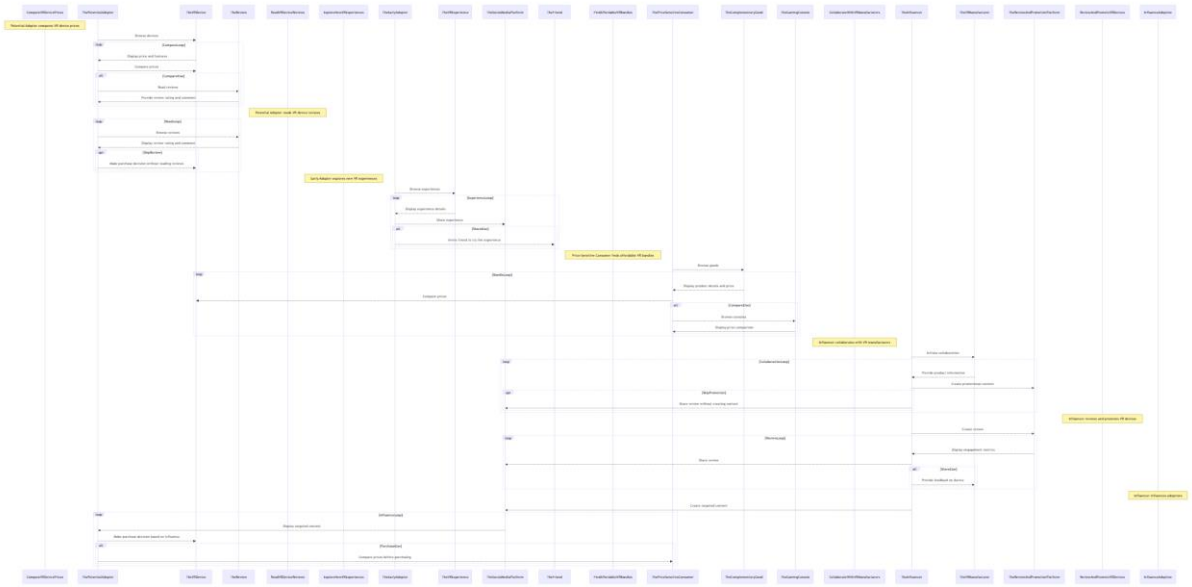


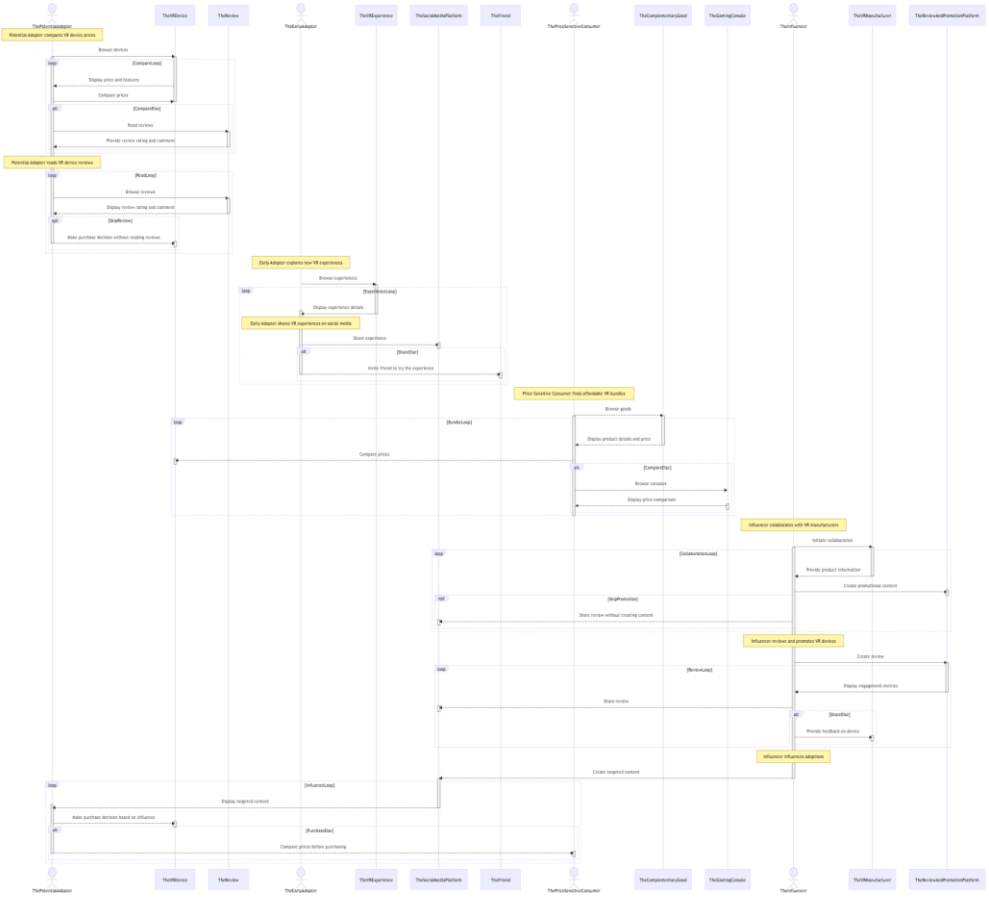
Figures 17 and 18: UML Class Diagram Before & After (syntax corrections made in "before": remove the possible nominal and ordinal values from attribute types)





Figures 19 and 20: UML State Diagram Before & After for the "PotentialAdopter" actor





Figures 21 and 22: UML Sequence Diagram Before & After (syntax corrections made in "before": remove loop name from `end` statements)

I have made significant progress on getting the model to a runnable state, by focusing on the UML use cases, experimental factors and output rather than the “after” diagrams. At the moment, an out-of-bounds error occurs while accessing a matrix that stores prices of VR bundles. Unfortunately, I did not have time to make the fix; however, I have added print statements to aid debugging. The GAML script for this model can be found in the `final_tuned_nemo/bass.gaml` file.

Once the script is fixed, there will be two runnable GAML files based on the GAML generated from locally run LLMs & the Streamlining EABSS with Generative AI: Part 3 - Small/Medium Model script.

5.4 NeMo v Gemma 3 12B: Which is Better?

To summarise, I cannot determine a clear “winner” between the Gemma 3 12B and Mistral NeMo models at this stage, although this is subjective as it depends on what is prioritised in terms of LLM output. Testing results are mixed (this may be due to a small sample size despite the 4 different test scenarios) and there isn't a clear winner. NeMo is more likely to account for all UML use cases in sequence diagrams, tends to generate larger class diagrams and more comprehensive GAML code in terms of actors and physical environment entities. However, Gemma 3 12B tends to generate Mermaid, Markdown and GAML code with less syntax errors.

5.5 Important Comments

The new implementation prompts are better at generating syntactically valid GAML experiment blocks, which was an issue in “Streamlining EABSS with Generative AI: Part 2”. More thorough comparisons between the results from Part 2 and Part 3 script testing could potentially uncover some interesting insights. The Part 3 scripts are the first version to use the smaller decomposed implementation prompts and verification is necessary to check if full parity has been reached yet with the previous Part 2 script. ***Performing this verification is crucial and should be the next immediate step in this project.***

If Gemma 3 12B with better prompts and better tuning does not yield improvements, testing the 27B variant of the Gemma 3 model is still a valuable use of resources – the model capacity of the lower-parameter-count 12B model could be a bottleneck. If this happened to be true, transitioning to the 27B parameter model would provide additional parameters that could capture the complex patterns and relationships that the 12B variant simply cannot.

I performed comprehensive testing while refining the Streamlining EABSS with Generative AI: Part 3 – Small/Medium Model Script, to identify common shortcomings across models before the final testing. All my claims and observations above are much

more heavily weighted on the final testing, however, I have still included evidence of this round of testing in the `before_final` folder for more insight, should you wish to inspect them.

I believe the test evidence in `final_tuned_gemma3`, `final_tuned_nemo` and `before_final` folders contain a lot of information that I have not had the chance to evaluate, and I therefore recommend inspecting them. This may sway the discussion above in 5.4. Better still, more testing of the scripts would yield a sample size to make more reliable & specific claims about exactly which EABSS aspects each model is good at.

Evidence of all other testing that was conducted at different stages of refining the scripts can be found in the `test_dump` folder. This includes evidence of testing conducted on various tunings of Gemma 3, OpenAI models etc.

6. OpenAI Models

Using the Python EABSS automation script written for ChatGPT models and OpenAI API credits, I completed a series of tests with the new Streamlining EABSS with Generative AI: Part 3 - Advanced Model Script on two different OpenAI models. I did not get the chance to evaluate the models thoroughly; however, I have included some insights below.

6.1 GPT-4.1 mini

I used the `gpt-4.1-mini-2025-04-14` version of the GPT-4.1 mini model[19] [20] with temperature set to 0.9 and top_p set to 0.9 (similar to the EABSS v1.0.1 script[21]). Copies of the run-throughs can be found in the `final_gpt_4.1_mini` folder. The model generated longer GAML scripts than the o4-mini model described below. While generating longer GAML scripts is not a guarantee of completeness with respect to the memorised keys, I expect the differentiator to be how well implemented each action/reflex is - unfortunately, I did not have time to check this in detail, but a quick look showed that both models generated some stubs.

It takes 10-15 minutes for this model to run the Streamlining EABSS with Generative AI: Part 3 – Advanced Model Script.

---

[19] https://platform.openai.com/docs/models/gpt-4.1-mini
[20] https://openai.com/index/gpt-4-1/)
[21] https://github.com/PeerOlafSiebers/abss-with-chatgpt/blob/main/supplementary-material-jasss-28-3-2/main/eabss-prompt-scripts/eabss-prompt-script_v1.0.1.txt

6.2 o4-mini

I used the `o4-mini-2025-04-16` version of the o4-mini reasoning model[22] [23]. Copies of the run-throughs can be found in the `final_o4_mini` folder. The o4-mini model does not allow custom temperature and top_p values, hence the run-throughs were performed without providing any CLI arguments to the Python EABSS automation script. The o4 mini model generates much more creative contexts that are richer in detail.

It takes 20-25 minutes for this model to run the Streamlining EABSS with Generative AI: Part 3 – Advanced Model Script.

7. Gemini 2.5 Pro

Using the Python EABSS automation script written for the Gemini models and the free tier of the Gemini API, I managed to complete one test of Gemini 2.5 Pro[24] [25]. The version of the script used is not the final version of the Streamlining EABSS with Generative AI: Part 3 - Advanced Model Script, however, it is relatively close and can give a good idea of the capabilities of the model. The generated GAML script was quite comprehensive, I have provided a copy of the test at `test_dump/gemini-2.5-pro_run_20250721-084127.json`. The free tier does limit the number of requests per day[26], which reduced the number of tests I could make.

8. Attempts to Test Other LLMs

I tried to test the o3-mini reasoning model[27] [28] by OpenAI, via their API; however, it asked me to verify my identity, which I was reluctant to do given the time constraint.

I also tried running the 27B parameter, QAT variant of Gemma 3 via Ollama[29], which was very slow due to its size (the total memory required to run the model with a 32K context window was 25GB) - only 60% of the model fit on the GPU. Some layers were offloaded to slower RAM memory and processed by CPU. Testing this model was abandoned after it took more than 12 minutes to go through the first 3 prompts.

---

[22] https://platform.openai.com/docs/models/o4-mini
[23] https://openai.com/index/introducing-o3-and-o4-mini/)
[24] https://ai.google.dev/gemini-api/docs/models#gemini-2.5-pro
[25] https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/
[26] https://ai.google.dev/gemini-api/docs/rate-limits
[27] https://platform.openai.com/docs/models/o3-mini
[28] https://openai.com/index/openai-o3-mini/
[29] https://ollama.com/library/gemma3:27b-it-qat

Furthermore, I tried to test the Magistral 24B parameter reasoning model by Mistral AI via Ollama[30], using the single T4 Nvidia GPU attached to my cloud virtual machine. Unfortunately, it was not possible to get even a single response in 10 minutes - the model and context window occupied ~21GB, and ⅓ of this had to be stored in slower RAM, as the GPU memory was fully occupied. I also tried the Phi4-reasoning-plus model[31] by Microsoft, a 14B parameter reasoning model, and faced the same issue. To alleviate this issue, I attempted to raise my GPU usage quota to 2 GPUs per machine on several occasions, however this was not possible, even when I applied in different Google Cloud regions.

9. Open WebUI

Open WebUI[32] is a free and open-source, user-friendly web-based front-end interface for interacting with large language models (LLMs) locally. It bridges an LLM server behind the scenes (which runs the model) with the user. The intuitive front-end interface design is similar to the ChatGPT web interface and is more suitable for the average user than using a CLI. The tool is actively maintained and has become a popular interface; it has a community of over 270,000 members as per their website.

Web search functionality[33] coupled with features such as Mermaid.js rendering[34] make Open WebUI a good tool for this project. The ability to get web results, such as academic citations, beyond the model training cutoff dates and with perhaps less hallucinations seems valuable, as does the ability to visualise generated diagrams all in one place. The tool also supports RAG, which could be useful if the Streamlining EABSS with Generative AI project were to take such a direction to augment the capabilities of models (for example, during code generation). Furthermore, Open WebUI supports model switching; the user can switch LLMs mid-conversation, and context is retained[35]. Finally, Open WebUI has a code editor similar to ChatGPT Canvas, and language support of the editor could be extended to Mermaid.js/Markdown/GAML (there is a reddit post discussing this: https://www.reddit.com/r/LocalLLaMA/comments/1i3as1m/openwebui_canvas_implementation_coming_soon/).

[30] https://ollama.com/library/magistral
[31] https://ollama.com/library/phi4-reasoning:plus
[32] https://openwebui.com/
[33] https://docs.openwebui.com/category/-web-search/
[34] https://docs.openwebui.com/features/code-execution/mermaid#using-mermaidjs-in-open-webui
[35] https://docs.openwebui.com/features/workspace/models#model-switching

Figure 23: An example of Mermaid.js rendering inside an Open WebUI chat.

I have recorded a demonstration video showing the Open WebUI interface and web search functionality, which can be found inside this folder.

I have also included instructions to use Open WebUI in the appendix of this document. Please note that Open WebUI requires a sign-up to establish an administrator user to manage other users and settings. Per the Open WebUI documentation, all data, including login information, is stored locally on your device[36].

10. Fine-tuning Protocol

I drafted a fine-tuning protocol document that discusses steps to perform supervised fine-tuning on the LLMs. It covers aspects such as the format of training data (and how to mimic the key memorisation technique used in the EABSS scripts in fine-tuning prompts), along with appropriate file formats to save training data and other considerations. It is currently in draft form and should be reviewed before adding to this folder, as there may be other legal/ethical considerations and technical steps that should be included, however, it is a good starting point.

11. Appendix

11.1 Manually Use LLMs with Ollama

1. To start the Ollama server, run the following command in the CLI:
   `OLLAMA_CONTEXT_LENGTH=<context length e.g. 32768> ollama serve`

[36] https://docs.openwebui.com/getting-started/quick-start/#:~:text=Privacy%20and%20Data%20Security%3A%20All,or%20by%20being%20made%20public

2. Run a model and interact with it manually (you will probably have to open another terminal): `ollama run <modelname>`

11.2 Using Python EABSS Automation Scripts

1. To start the Ollama server, run the following command in the CLI:

```
OLLAMA_CONTEXT_LENGTH=<context length e.g. 32768> ollama serve
```

2. Now, the EABSS automation script can be run in another terminal:
    a. To run the automation script for models served via Ollama, run the following command (square brackets contain optional CLI arguments):
    ```
    python eabss_automation_bot.py --model <model> --prompt-filepath
    streamlining_eabss_3_small_medium_model_script.json [--temperature <temperature> --top_p <top_p> --verbose]
    ```
    b. To run the automation script for ChatGPT, run the following command (square brackets contain optional CLI arguments, some arguments such as temperature are not accepted by all models):
    ```
    python chatgpt_eabss_automation_bot.py --model <e.g. o4-mini-2025-04-16> --prompt-filepath
    streamlining_eabss_3_advanced_model_script.json [--temperature 0.9 --top-p 0.9 --verbose]
    ```
    c. To run the automation script for Gemini, run the following command (square brackets contain optional CLI arguments, some arguments such as temperature may not be accepted by all models):
    ```
    python gemini_eabss_automation_bot.py --model <e.g. gemini-2.5-pro> --prompt-filepath
    streamlining_eabss_3_advanced_model_script.json [--temperature 0.9 --top-p 0.9 --verbose]
    ```

11.3 Changing LLM Hyperparameters (e.g. Temperature) using Ollama Modelfile

I have included an end-to-end example that changes the temperature of the gemma3:12b-it-qat model below, which is based on the guide at https://www.gpu-mart.com/blog/custom-llm-models-with-ollama-modelfile. I have introduced an intermediate step between 1) and 2).

1. Create a new Modelfile based on the existing Modelfile: `ollama show gemma3:12b-it-qat --modelfile > gemma3_12b_itqat_0.6temp.modelfile`
2. Make the necessary hyperparameter changes
3. Replace `FROM` in newly created `gemma3_12b_itqat_0.6temp.modelfile` by following the instructions at the top of the original Modelfile.

4. Create a new model from the Modelfile (you may choose self-descriptive model names that include the parameter values): `ollama create gemma3_12b_itqat_0.6temp --file gemma3_12b_itqat_0.6temp.modelfile`
5. List all on-device models (you should see the newly created model): `ollama list`
6. To start the Ollama server, run the following command in the CLI: `OLLAMA_CONTEXT_LENGTH=<context length e.g. 32768> ollama serve`
7. Run the newly created model and test it manually in another terminal: `ollama run gemma3_12b_itqat_0.6temp:latest`

To run the Python EABSS automation script with the newly created model (square brackets contain optional CLI arguments):

1. First make sure to start the Ollama server, run the following command in the CLI: `OLLAMA_CONTEXT_LENGTH=<context length e.g. 32768> ollama serve`
2. Run the automation Python script in another terminal
   `python eabss_automation_bot.py --model gemma3_12b_itqat_0.6temp:latest --prompt-filepath <prompts_file_path>.json [--verbose]`

Note that you do not need to provide the hyperparameter settings via the CLI arguments when running the Python EABSS automation script if you have used this approach. This removes the overhead associated with remembering to set the correct hyperparameter settings.

11.4 LLM Fine Tuning Tutorials

- Gemma 3: https://docs.unsloth.ai/basics/tutorials-how-to-fine-tune-and-run-llms/Gemma-3-how-to-run-and-fine-tune
- DeepSeek: https://docs.unsloth.ai/basics/tutorials-how-to-fine-tune-and-run-llms/deepseek-r1-how-to-run-locally
- Jupyter notebooks for fine-tuning a variety of models can be found at: https://github.com/unslothai/notebooks/tree/main/nb
- Also, some in-depth analysis into Gemma 3 published by the same company can be found at: https://unsloth.ai/blog/Gemma3

11.5 Using Open WebUI (with LLM server running in Cloud VM)

Below, I use the Ollama LLM server running in Google Cloud as an example:

1. In cloud, start Ollama: `OLLAMA_HOST=0.0.0.0:11434 OLLAMA_CONTEXT_LENGTH=<context_length> ollama serve`
2. In cloud, in another terminal:
   a. Activate Python virtual environment

    b. Install Open WebUI: `pip install open-webui`
    c. Start Open WebUI: `open-webui serve`
3. On local machine, SSH into the cloud instance: `gcloud compute ssh <instance-name> --zone=<zone-name> -- -L <8080_or_whichever_port_cloud_exposes_openwebui>:localhost:8080`
4. In the local machine's browser, go to `localhost:8080`, you should see the Open WebUI interface.

11.6 Using Open WebUI (with LLM server running on Physical Machine)

1. In local machine, start Ollama: `OLLAMA_CONTEXT_LENGTH=<context_length> ollama serve`
2. On local machine in another terminal:
    a. Activate Python virtual environment
    b. Start Open WebUI: `open-webui serve`
3. In the local machine's browser go to `localhost:<8080_or_whichever_port_cloud_exposes_openwebui>`, you should see the Open WebUI interface.

11.7 Miscellaneous

- I have found a tool to help evaluate responses from different LLMs that run via Ollama which can be found here (I did not have time to it test myself, but it could be useful): https://github.com/dezoito/ollama-grid-search
- An interesting post that talks about different hyperparameter settings: https://www.reddit.com/r/LocalLLaMA/comments/17vonjo/your_settings_are_probably_hurting_your_model_why/
- OpenAI are apparently releasing an open-source model (https://techcrunch.com/2025/07/11/openai-delays-the-release-of-its-open-model-again/). It might be worth testing (if and) when it is released (original release date was this summer, however, it has been postponed).