End of Internship Report: Streamlining the Coding Bottleneck in Simulation Modelling with Generative AI

Sener Topaloglu

Supervisor: Prof. Peer-Olaf Siebers

September 22, 2024

Abbreviations

AI Artificial Intelligence

CUDA Compute Unified Device Architecture

DROP Discrete Reasoning Over Paragraphs

EABSS Engineering Agent-Based Social Simulation

GAMA GIS Agent-based Modelling Architecture

GAML GAMA Markup Language

GPT General Purpose Transformer

GPU Graphics Processing Unit

JASSS Journal of Artificial Societies and Social Simulation

LLM Large Language Model

MMLU Massive Multitask Language Understanding

NLP Natural Language Processing

RAG Retrieval Augmented Generation

UML Unified Modelling Language

Contents

1 Introduction

The domains of Operations Research and Social Simulation rely on intricate simulation modelling to understand and experiment with complex systems. Applications of simulation models span both academia and industry, to simulate a multitude of tasks, including disaster response and organisational change management. Currently, implementing such models consists of manual coding, a technical and time-consuming task that introduces a significant bottleneck in model development.

The aim of this project was to investigate how generative Artificial Intelligence (AI) can translate high-level, natural language descriptions of systems, their components, and the rules governing their dynamics into executable GAML scripts for the GIS Agent-based Modelling Architecture (GAMA) simulation engine.

In addition, I was asked to build a library of reusable design patterns, in the form of prompts and advanced prompt engineering techniques, which will serve as building blocks to develop elaborate simulation models. This approach not only simplifies the process for developers but also allows for more efficient collaboration between domain experts and model developers, who may not possess sufficient technical understanding to write syntactically correct, executable simulation models.

This project represents a significant step towards the broader initiative of reducing the time and effort required for modelling across frameworks that may not share the same syntax and semantics as GAMA Markup Language (GAML), such as the AgentPy Python library. By developing tools that automate the coding process, we aim to make agent-based simulation more accessible and scalable. It is expected that the foundational work completed in this project will be extendable to support many different simulation engines and use cases from epidemiology to economics and everything in between. The ultimate goal is to integrate AI-driven model generation more deeply into the development workflow, leading to a future where complete models can be generated with very minimal human intervention.

2 Project Structure

This section outlines the systematic approach undertaken in this research project, categorised into five major phases (work packages): preparation, design, implementation, testing and authoring the final report. Each work package consists of specific sub-tasks that collectively contribute to achieving the project objectives. A visual representation of my approach can be found in section 2 of the appendix.

3 Work Package 1: Preparation

The preparation phase was foundational and involved a thorough exploration of various components essential for this project. The main focus was on identifying suitable models, fine-tuning strategies, and learning the notion of agent-based simulation modelling.

Given the project's focus on the generation of GAML scripts from natural language descriptions and Unified Modelling Language (UML) diagrams, preparation for the internship required a comprehensive understanding of multiple domains: agent-based simulation modelling, the GAMA platform, and the intricacies of generative AI models, specifically in the context of script generation. This chapter outlines the steps taken to acquire the necessary knowledge and skills, detailing the preparatory work undertaken to ensure a solid foundation for the internship.

3.1 Open Source Models

Given the project's focus on leveraging generative AI for script translation, it was essential to develop an understanding of the current Large Language Models (LLMs). As I had previous experience of implementing decoder-only General Purpose Transformer (GPT) models, I furthered my knowledge by reviewing a survey[1] that covered a range of other architectures underpinning existing LLMs.

With a solid understanding of the theoretical aspects of generative AI, I proceeded to experiment with existing LLMs offerings. I shortlisted various models which I considered to be well-documented and widely used – I have previously experienced many compatibility issues due to frequent (breaking) changes to many of the libraries/frameworks that exist in the deep learning ecosystem. Naturally, a strong user support base is advantageous in such situations. I then conducted narrowed my shortlist to models that have commonly been used as base models for similar tasks, such as Python/XML code generation; Hugging Face (https://huggingface.co/) model repository allows users to filter for common models.

Each shortlisted model was compared against several benchmarks that quantify model performance in various areas that I deemed relevant to the project; Massive Multitask Language Understanding (MMLU), Discrete Reasoning Over Paragraphs (DROP) and EvalPlus. MMLU[2] is a benchmark to evaluate general language understanding in zero-shot and few-shot settings. DROP[3] is an evaluation where models must extract relevant information from English-text paragraphs before executing discrete reasoning steps on them. EvalPlus[4] is a code generation benchmark that measures the quantity

[1] Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large language models: A survey. arXiv:2402.06196

[2] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring massive multitask language understanding. arXiv:2009.03300.

[3] Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., & Gardner, M. (2019). DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. arXiv:1903.00161.

[4] Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (2024). Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems, 36.

and quality of synthesised code. Please refer to Table 1 below to see each model's performance in the respective benchmarks.

Table 1: Benchmark comparison of shortlisted models

	Context Window	MMLU (5 shot)	DROP (3-shot, F1)	EvalPlus
Llama3 - 8B	8k	66.6	58.4	29.3
Mistral - 7B	8k	62.5	54.4	23.8
Gemma - 7B	8k	64.3	56.3	28.7

References: https://ai.meta.com/blog/meta-llama-3/ (MMLU, DROP) and https://evalplus.github.io/leaderboard.html (EvalPlus).
Please note: DROP scores referenced were measured by Meta FAIR (creators of Llama models), as creators of the other models in the table did not publish such scores.

Finally, I empirically evaluated each models' performance in generating GAML code, this assessment was necessary because the EvalPlus benchmark does not cover GAML code generation. Model performance (latency) was also evaluated empirically, because no existing leaderboard has recorded all the shortlisted LLMs running on the same hardware (the closest leaderboard to cover all models can be found at https://huggingface.co/spaces/ArtificialAnalysis/LLM-Performance-Leaderboard).

My test data includes the following prompts, and all testing was conducted in a zero-shot manner:

1. Prompt 1: How can I define the experiment attributes, actions, and behaviours in a model in GAML (GAMA Markup Language)?

2. Prompt 2: Give me a GAML code snippet for a grid structure named "vegetationCell" with a size of 50x50, considering a neighbourhood of 4 cells around the current cell. The attributes include "maxFood","foodProd", "food" a colour attribute based on food level, and a list of neighbours at a distance of 2 cells.

3. Prompt 3: Generate a code snippet that defines a list of string called "listOfString" composed of three elements "A", "B" and "C".

At the end of this process, I choose *Llama3-8B* as the base model to meet the specific needs of the project.

3.2 Fine-Tuning Approaches: Supervised & Retrieval Augmented Generation (RAG)

I considered both popular, industry standard approaches to fine-tuning models; conventional fine-tuning and RAG.

Conventional fine-tuning involves supervised training of a base model on a task-specific

dataset (that is representative of the kind of input the model will encounter when deployed).

RAG, on the other hand, integrates retrieval mechanisms into the generation process, allowing the model to access knowledge bases that are external to the model, such as a corpora of text documents relevant to the area of interest. Embedding vectors are used to represent tokens of text in each document and the retrieval mechanism matches queries with the most appropriate content (which has the closest embeddings to the input prompt).

I decided that I would design and implement both fine-tuning approaches to compare them in the context of modelling development.

3.3 Conceptual Knowledge: Agent-Based Simulation

To understand the context of the project, it was imperative to develop a thorough understanding of ABM. ABM is a powerful computational technique used in the study of complex systems, particularly those involving human interactions and decision-making. In ABM, individual entities (agents) with distinct behaviours interact within an environment, leading to emergent phenomena which may be non-linear and difficult to predict. For example, consider the simulation of a traffic flow in a city. In this model, each car is represented as an individual agent with distinct behaviours such as speed, lane-changing preferences, and response to traffic signals. These cars interact with each other and with the road network (the environment). When traffic density increases, agents may slow down, change lanes, or reroute based on local conditions. The emergent phenomenon in this case is traffic congestion. While the behaviour of each individual car is simple and predictable, the overall traffic pattern that arises from just a single car slowing down can become hard to predict.

I gained understanding of social simulation and agent-based simulation from the book Simulating Social Complexity[5] (recommended by my supervisor).

3.4 Implementation Knowledge: Agent-Based Simulation

Familiarity with GAMA (an open-source platform for building agent-based models) and it's supported language, GAML, was essential for the project's success. I used the approach of "learning by doing" and created several GAML models, including:

1. Predator-prey simulation model: simulates the interactions between predator and prey populations in a common environment (habitat). This model required the implementation of distinct agent types (predators and prey), each with their own set of behaviours (e.g., hunting, fleeing, reproducing). The simulation demonstrated how individual interactions could lead to population dynamics, such as oscillations in predator and prey numbers, illustrating the non-linear feedback loops common in ecological systems. I enjoyed learning how parameter changes affect the dynamics in this scenario, for example, population cycles of both species tend to oscillate, however, increasing the rabbit population too much can change this dynamic as they compete with each other.

[5]Edmonds B and Meyer R (2017), Simulating Social Complexity: A Handbook - 2e

2. Flu-Dispersion Model: This model simulates the spread of an infectious disease (such as influenza) within a population. The natural language input described the movement of agents, transmission probabilities, and health states (e.g., susceptible, infected, recovered). This test assessed the model's capacity to handle scenarios with complex interactions between agents and environmental variables, such as varying infection rates or differing movement patterns.

I also reviewed the GAMA documentation and official forum to understand its syntax and learn best practices for coding simulation models. I am very grateful to my supervisor, who gave me a walkthrough of the GAMA IDE and signposted me in the right direction at the start of my journey learning GAMA. The hands-on experience gained through these exercises was instrumental in bridging the gap between theoretical knowledge and practical application.

3.5 Engineering Agent-Based Social Simulation (EABSS)

The learning process began with an in-depth review of the foundational EABSS framework paper[6]. The paper provides a comprehensive methodology for developing agent-based models that capture the complexities of social systems, emphasising modular design, agent behaviour specification, and the importance of model validation.

4 Work Package 2: Design

In this phase, I advance onto designing two distinct pipeline architectures (for supervised fine-tuning and RAG), followed by feasibility testing design that will allow me review my hypotheses:

1. Models will not require fine-tuning to produce satisfactory results in the analysis phase of EABSS (i.e. assuming a particular role to understand the problem statement).

2. Fine-tuned models will perform better on UML generation than their counterparts. In the context of this project, UML generation refers to the generation of MermaidJS scripts; MermaidJS is a JavaScript-based tool that enables users to create UML diagrams through simple, text-based syntax.

4.1 Fine-tuning pipeline design

In the supervised fine-tuning pipeline, data is processed into the correct format (for Llama models this is a pair of *instruction* and *output* string values for each entry in the training dataset), then, the processed dataset is stored as a list of JSON records. Just before training, the JSON file is read by a Python script and converted into a Pandas dataframe object.

The RAG pipeline makes use of the ChromaDB in-memory vector database, to index embeddings of both natural language inputs from the user, a key advantage of using

[6]Siebers, P.O. & Klügl, F. (2017). What software engineering has to offer to agent-based social simulation. In Edmonds, B. and Meyer, R. (Eds). Simulating Social Complexity: A Handbook - 2e. Springer.

ChromaDB is that it is pre-packaged with a driver to match user prompts with the appropriate embeddings – saving development related overhead. Various embedding generators (models) exist, I chose the *all-MiniLM-L6-v2* model embedding due its compatibility with LangChain and low memory footprint. LangChain was utilised to connect the retrieved information with the base language model. Finally, I developed a Streamlit-based web application to communicate with this package.

During the whole process, I placed priority on using open-source libraries and frameworks.

4.2 Test experiment design

Test experiments were structured to cover a range of scenarios, with increasing levels of complexity, to comprehensively assess the model's adaptability, accuracy, and reliability. The three test cases below were selected based on their relevance to the core domains of Operations Research and Social Simulation and are a mix of classical simulation problems and comparatively modern use cases.

1. Predator-prey model (as described in section 3.4): simulates the interactions between predator and prey populations in a common environment (habitat). This model required the implementation of distinct agent types (predators and prey), each with their own set of behaviours (e.g., hunting, fleeing, reproducing). The simulation demonstrated how individual interactions could lead to population dynamics, such as oscillations in predator and prey numbers, illustrating the non-linear feedback loops common in ecological systems. I enjoyed learning how parameter changes affect the dynamics in this scenario, for example, population cycles of both species tend to oscillate, however, increasing the rabbit population too much can change this dynamic as they compete with each other.

2. Ant pheromone diffusion model: simulates the behaviour of ants in a foraging scenario, focusing on the influence of pheromone trails on their movement patterns. This exercise required the implementation of individual ant agents, each with behaviours such as moving, sensing pheromones, and depositing pheromones. The simulation highlighted the emergent properties of collective behaviour, such as the formation of optimal foraging paths.

3. Adaptive architecture in museum exhibitions model: consisting of 2 types of artefacts: large wall-mounted smart content windows that move with visitors and a smart partition wall that creates a dynamic and flexible exhibition environment by continuously analysing visitor movement, making real-time decisions, and physically re-configuring the space to optimise the experience.

Each test model will be generated using design patterns[7] (natural language prompts) that align with the EABSS framework.

[7]Siebers, P. O. (2024). Exploring the Potential of Conversational AI Support for Agent-Based Social Simulation Model Design. arXiv preprint arXiv:2405.08032.

5 Work Package 3: Implementation

This section outlines the key steps in implementing the necessary components for the successful deployment of the AI models. Starting with running base LLMs, I proceeded to implementing the fine-tuning pipelines and curating task-specific datasets. Finally, I deployed the fine-tuned models for experimentation.

5.1 Running Base LLMs

It was necessary to choose a platform to serve base LLMs. Ollama, an open-source, lightweight server optimised for local deployment of language models was chosen. Ollama's built-in repository provides easy access to many popular base models, including Llama3, which are guaranteed to be compatible with the server. Furthermore, Ollama supports running custom models using Modelfile schemas for quick experimentation. Consideration was also given to vLLM, another open-source inference framework designed for serving LLMs, however, I regarded the lack of documentation and support for various models as a major drawback.

One issue I faced was the inability to run base models on-device. My device (running Windows 10, with Intel i5-1035G1 CPU @ 1.00GHz and NVIDIA GeForce MX350 GPU) has 16GB RAM and I was not able to run any model without facing high latency (typically >5 minutes after just a few hundred tokens into conversation). Consequently, I made use of virtual machines hosted in Google Cloud to execute and run models.

5.2 Curating and Creating Fine-tuning Dataset

The success of generative AI models in specific applications depends heavily on the quality and relevance of the fine-tuning datasets used during the model optimisation phase. The first step in curating the fine-tuning dataset involved a comprehensive evaluation of the chosen base model's performance in three key areas:

1. Ability to generate syntactically correct GAML scripts based on natural language input and MermaidJS representations of UML was evaluated.

2. Ability to produce accurate UML (MermaidJS) diagrams from textual descriptions. UML visual representations are an industry standard in modern software development. Evaluation was focused on syntax correctness, logical representation and the ability to generate UML features such as branching in state diagrams, and inheritance in class diagrams.

3. Conversational competence within the EABSS Framework, for example, ability to understand and respond accurately to queries such as generating sequence diagrams from class and state diagrams generated in the steps prior.

I curated (part of 5.2.1) and synthesised (remainder of 5.2.1 and onwards) datasets for fine-tuning, each described in the relevant subsection below.

5.2.1 GAML Code Generation Datasets

Since errors were discovered in the GAML scripts generated from the base Llama3 model, it was important to training the model to generate valid GAML scripts from the following inputs:

- Natural language prompts related to general GAML knowledge. The dataset for this task was retrieved from Hugging Face[8] and processed by me.
- Specific simulation scenario prompts, that resemble scenarios the final LLM may face in production.
- UML Mermaid JS sequence diagrams derived from state and class diagrams, to further link UML representations and GAML code.

The integration of these datasets enabled a multi-faceted approach to code generation, providing the model with diverse input types.

5.2.2 UML Class, Sequence, and State Diagrams

Diagrams were extracted from prompts describing natural language scenario prompts, serving as an intermediate representation between textual descriptions and code. Class diagrams were used to represent the structure of a system, sequence diagrams to capture dynamic interactions, and state diagrams to depict the possible states of system components. Sequence diagrams were also generated from MermaidJS scripts representing class and state diagrams generated in prior stage of dataset creation. The inclusion of these diagrams aimed to help the model learn the syntactic and semantic connections between UML and GAML.

5.2.3 JASSS Keywords and Abstracts

Journal of Artificial Societies and Social Simulation (JASSS) is a leading publication in the field of social simulation. Keywords and abstracts from the journal were used to imbue the model with domain-specific terminology and contextual knowledge. This domain knowledge helped the model better understand and generate simulation-related content.

5.3 Implementing fine-tuning

A review of state-of-the-art methodologies revealed that llama.cpp was a widely used C++ library for LLM inference. Various python libraries expose a llama.cpp API, as well as handling the underlying model architecture, data loading, and training routines. Fine-tuning requires careful calibration of several key parameters to ensure that the model converges effectively and generalises well to the target tasks. The primary parameters considered during the fine-tuning process were:

- Batch size: the number of samples processed before the model updates its weights. Smaller batch sizes can lead to more noisy updates and slower convergence, however it is found that smaller sizes eventually generalise better, whereas larger batch sizes allow for faster training with the risk of overfitting and poorer generalisation performance due to learning patterns in the data too precisely.

[8] https://huggingface.co/datasets/Phanh2532/GAML-Data

- Learning rate: controls the step size at each iteration while moving towards a minimum of the loss function. A learning rate scheduler was also employed to dynamically adjust the rate during training, starting higher to speed up convergence and then reducing as the model approached optimal performance to reduce the possibilty of divergence. Leading academics in the field of theoretical machine learning have noted it is the single most important hyper-parameter worth tuning[9].

- Number of epochs: refers to how many times the entire training dataset passes through the model. Determining the appropriate number of epochs involved evaluating model performance at different stages to avoid overfitting while ensuring sufficient training exposure to the dataset.

- Optimiser: the choice of optimiser significantly impacts the efficiency of the fine-tuning process. Common optimizers such as Adam, AdamW, and SGD were tested, with AdamW being selected due to its adaptive learning rate properties and effective weight decay mechanism, which improved convergence while mitigating overfitting.

The fine-tuning process was carried out using a Python script that integrates with the Unsloth API. This setup allowed highly efficient Low Rank Approximation (LoRA) fine-tuning that is highly configurable at no extra cost. LoRA modifies the attention layers of a transformer model, instead of updating the entire weight matrices during fine-tuning. Weight matrices are decomposed into two smaller matrices, where one is of a lower rank, thus reducing the computational cost of training. A visual representation of LORA can be found in Figure 2 below.

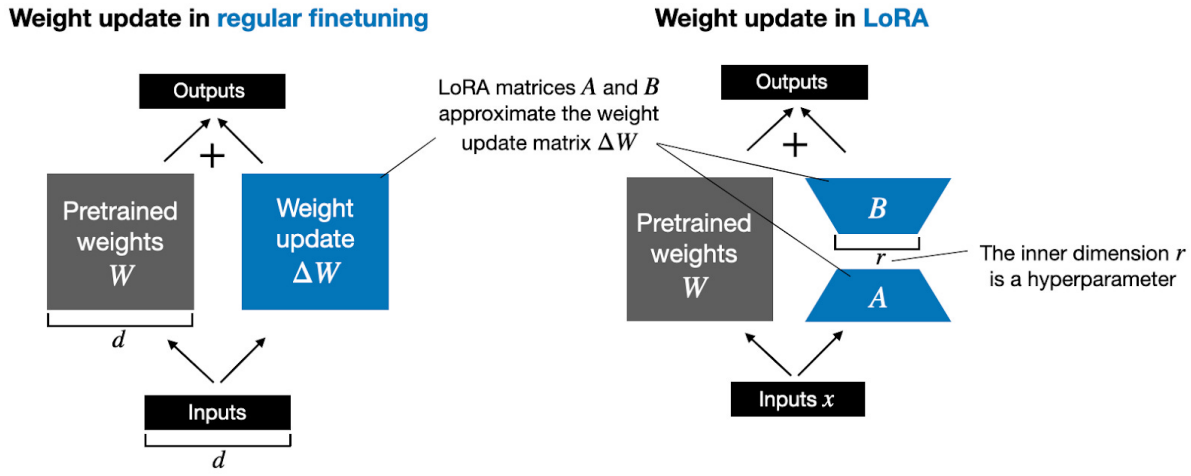

Figure 1: LORA (https://magazine.sebastianraschka.com/p/lora-and-dora-from-scratch)

The fine-tuning script was written such that each trained weight could be represented using a 8-bit or 16-bit floating point value. 16-bit values allow for more accurate representation of the weights, however, it is taxing on computational resources. In

[9]Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural networks: Tricks of the trade: Second edition (pp. 437-478). Berlin, Heidelberg: Springer Berlin Heidelberg.

contrast, less accurate 8-bit quantisation significantly reduces memory usage and increases inference speed, making it highly suitable for rapid experimentation. Given this trade-off, 8-bit quantisation was employed during the initial experimentation phases - this allowed us to quickly assess model performance on new datasets and configurations. Once a stable and promising model configuration was identified, 16-bit quantisation was used for the final fine-tuning stages to enhance the accuracy of code generation.

The output of the fine-tuning process were .gguf (Grok Generation Universal Format) files that encapsulate the quantised weights, metadata, and architectural information of a fine-tuned model in a format that is compatible (and efficient to load and execute) with Ollama.

5.4 Running fine-tuned LLMs

The .gguf files are then referenced in a Modelfile (a schema to create instances of models that is compatible with the Ollama server). Using the Ollama CLI, models are then instantiated (`ollama create <new-model-name> --file Modelfile`) and executed (`ollama run <new-model-name>`), ready for use.

6 Work Package 4: Testing

The testing phase of this project focuses on evaluating the performance of fine-tuned generative AI models across several critical dimensions. The primary objective is to assess how well these models can comprehend and reason through provided briefs, generate logical and syntactically accurate UML diagrams, and subsequently produce semantically and syntactically correct GAML code.

6.1 Llama3

6.1.1 RAG

Testing a proof-of-concept RAG pipeline (which used a corpora of news reports) revealed significant performance limitations. The pipeline was exceedingly slow, and thus unable to provide timely responses. Moreover, it frequently failed to generate correct answers to questions, even when those answers were directly contained within the provided documents. Furthermore, the RAG pipeline's ability to generalise was poor – it could not handle questions outside the scope of its training documents. As a result, the pipeline under-performed in terms of both accuracy and generalisation, making it unsuitable for this project's needs.

I also believed it would not be feasible to create a corpora of documents that would fulfill all the project criteria; accurate UML code generation and GAML script generation, and possess domain knowledge. RAG relies on a large quantity of documents to work adequately - we do not have enough examples of simulation model documentation that conforms to EABSS at hand, and it is difficult to create high numbers of quality synthetic data (consider multi-page developer documentation).

6.1.2 Supervised fine-tuning

Supervised fine-tuning on Llama3, in contrast, delivered better results. The fine-tuning process was incremental, with each training/testing iteration focusing on progressively more complex tasks. Initially, the model was fine-tuned to generate UML diagrams (in MermaidJS) from natural language prompts. Subsequent iterations involved teaching the model to generate GAML code from natural language prompts. I noticed infrequent hallucinations by the model, but did not consider them serious as every model has the potential to show this behaviour.

6.2 Llama3.1

During testing, Meta released Llama3.1 – an updated version of the Llama3 model. Shortly after, I made the switch to using Llama3.1 with the expectation that a larger context window would improve handling of long-form content. I also believe the gains in reasoning and code generation benchmark performance could potentially reduce the likelihood of hallucinations. Please find the table below to compare benchmark performance of Llama3.1 and Llama3[10].

Table 2: Benchmark comparison of Llama3.1 and Llama3

	Context Window	MMLU (5 shot)	DROP (3-shot, F1)	EvalPlus
Llama3.1 - 8B	128k	37.1	59.5	72.8
Llama3 - 8B	8k	36.2	58.4	70.6

Despite the change of model, the design and implementation of the supervised fine-tuning pipeline remained largely the same. Key differences were dependency upgrades to fix breaking changes in the fine-tuning script and an Ollama server upgrade, in order to run the model. Based on the poor results obtained with the RAG-fine-tuned Llama3, I opted not to fine-tuning Llama3.1 using RAG, opting for traditional fine-tuning only. The model was trained on the final iteration of training data, which introduced UML diagram prompts (for GAML code generation) and a collection of questions pertaining to abstracts of papers published in JASSS, to equip it with domain-specific knowledge.

However, despite the improvements, hallucinations persisted, even when the input prompts were part of the training data. This issue has been documented as quantisation causing disturbances to the sophisticated knowledge structure during pre-training[11]. Behaviour similar to that documented below was observed several times throughout the testing process.

```
sb_topaloglu1@instance-20240729-100526:   $ ollama run
fine-tuned-model-2:latest
>>> Step-by-step, work through the following task list in the given order
```

[10] https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md

[11] Huang, W., Ma, X., Qin, H., Zheng, X., Lv, C., Chen, H., ... & Magno, M. (2024). How good are low-bit quantized llama3 models? an empirical study. arXiv:2404.14047.

```
during the entire conversation.  Got it?  Say "yes" or say "no".
['task', ['create a new species called "human" from the basic template,
and add an action "talk" to it.', 'define the action "talk" so that when
it is executed in a specific location named "town", a message appears on
the console of the user interface:  "The human talks in town".', 'add
another action named "walk" to the species "human", with no argument and
with the following code snippet inside it :  "do walk speed:  0.5;"',
'define an initialization block that will be executed when a new human is
created, which consists of positioning the agent at random within a
specific polygon called "town"', 'define an experiment named "experiment"
that contains:', 'a display named "main_display" that has a camera
looking down on the "town" polygon.', 'an output device called "console"
for displaying messages in the console window.', 'launch this simulation
by running the main experiment "experiment", where we will see humans
wandering around and speaking in town.'],
```

6.3 Mistral-NeMo

Ultimately, I decided to switch to the Mistral-NeMo model for the last iteration of testing. Mistral-NeMo was released after I made the initial decision to use the Llama family of models, hence I became aware of its existence in the later stages of the project. The model holds several advantages over Llama3.1, including a higher number of parameters (12 billion vs. 8 billion) and a larger context window (up to 128k tokens). These characteristics make it more suited for handling complex, context-rich tasks like GAML script generation and UML diagram creation. Additionally, I noticed Mistral-NeMo has garnered positive feedback from users in various online communities.

In comparison to Llama3.1, Mistral-NeMo provided better overall performance in general question answering ability (MMLU 5 shot score is higher by 1.4%) and following empirical evaluation to check for code generation, it was ultimately selected as the final model for this project. I reduced my reliance on benchmark scores and the model selection process was mostly empirical.

Please find some examples of generated UML diagrams, generated using a single-prompt, without background knowledge, in the appendix A3.

I then generated GAML scripts, using basic text prompts as you can find in the appendix.

I did not train Mistral-NeMo using RAG due to disappointing results in Llama3, and unfortunately, I did not have the time to successfully fine-tune the model using supervised methods, although I did create a script for this purpose.

The very short remainder of time (last couple of days) of testing was testing using the EABSS script (kindly provided by my supervisor). In order to submit the design patterns (which were an important deliverable), I did a lot of isolated testing (rather than running the whole EABSS script, I executed parts on example scenarios - similar to the ant foraging scenario in the appendix), this allowed me to refine the existing design

prompts and develop my own. Unfortunately, my assumption that running the EABSS script would return similar results to lots of isolated tests run in succession was wrong. I made this assumption due to the long (128k token) context length of the model, and while context length was sufficient to run the EABSS script, the script introduced lots of additonal information. From my experience running the EABSS script for testing (I was only able to do one run because the script took several hours to complete, I expand on this later in section 8.1) the model becomes over-concentrated on the additional information introduced. The result is that UML representations are often syntatically correct MermaidJS code albeit not logically coherent, please see the use case diagram in the EABSS script run for this. This issue also transcends into GAML code generation. Nevertheless, I propose 2 different design patterns to implement models in GAML:

1. Command the model to generate the GAML script in one attempt, followed by a prompt commanding the model to adjust the generated code.
2. Command the model to generate the GAML script in parts (e.g. first global code block then species implementations, then experiment code block), followed by a prompt commanding the model to adjust the generated code.

7 Work Package 5: Report Writing

The final work package of this research project focused on consolidating the findings and documenting the entire process. The objective was to produce a detailed and coherent report that captures the progress made, challenges faced, and lessons learned during the internship. Additionally, I outlined the next steps in pursuing the broader objective of automating the simulation modelling workflow.

The report was structured to ensure clarity and logical progression from one section to the next. Writing the report was an iterative process, involving constant refinement for clarity and technical accuracy. Key aspects such as data interpretation, test results, and the fine-tuning pipeline were revisited multiple times. I am very grateful to my supervisor, who took the time to review and provide feedback on my report.

One challenge I faced writing the report was presenting the limitations of the models, such as the hallucinations observed during Llama3 testing, without detracting from the project's overall contributions. This was achieved by referencing existing literature on LLM quantisation and discussing potential avenues for improvement, thus framing the limitations as areas for future research.

8 Outcomes

The internship resulted in several successful outcomes, both in terms of research results and my personal development. Under the supervision of Prof. Siebers, I successfully researched the feasibility of various open-source LLMs, including Mistral and Meta Llama models for generating GAML simulation model scripts. While I experimented with both traditional fine-tuning methods and RAG pipelines, to produce more accurate, syntactically valid scripts, it was found that conventional fine-tuning on smaller datasets provided more reliable answers without losing ability to generalise. It

was noticed that fine-tuning the Llama3/Llama3.1 models on large datasets resulted in strong hallucinations that rendered the model unusable. A key learning for me personally was realising that empirical evaluation has a strong place – performance of models and techniques do not always translate to successful outcomes in a practical sense. A major technical achievements was the creation of reusable design patterns for conversational AI. These patterns were developed in close collaboration with my supervisor and conform to the EABSS Framework. The most interesting discovery during the internship for me was that there is potential to translate JavaScript (MermaidJS) code into GAML code. The generalisable nature of the developed patterns and processes means they can be adapted to different simulation engines with ease.
Along with these successful outcomes, there are some undesired outcomes that must be improved upon. As mentioned earlier, running EABSS scripts is an extremely long process because of the memory required to store long context windows (which are needed considered the size of the input (and output generated) from the prompts. Running the single EABSS script test I had the chance to do, after finalising my design prompts, took over 4 hours as the inference process broke down 3/4 of the way into the script – this was exacerbated by the lack of a powerful GPU in the virtual machine which hosted the Ollama server. A learning for me was that I would perform such tasks on much more powerful devices, and I would need to budget accordingly. After spending the final 10 days trying to the best of my ability, to get a running and operationally successful Mistral-NeMo model, it was unpleasant to watch the model not perform as well as I anticipated; and it was difficult to leave the final GAML code with bugs, as I didn't have the time to debug the script.
Nevertheless, we are in a position to automate parts of the EABSS development process, namely; preparation, analysis, design (partially – generation of state diagrams and its relevant documentation), and conclusion. Design and implementation are phases that LLMs typically struggle with, due to their lack of ability to generate syntatically correct, logical MermaidJS and GAML code. The final Mistral-Nemo model has shown promise to generate GAML code from MermaidJS UML representations, but can contain errors that are difficult to debug. I have identified a solution for this in the section below.

8.1 Outlook

A shortcoming of the current model is the ability to reliably generate syntatically valid and coherent use case diagrams, class diagrams and GAML scripts from UML representations. To counteract this, I have written the design patterns to be restrictive and prevent undesired output (such as XML representations of models), however this is not a reliable approach. This can be improved by collecting more fine-tuning data. By increasing the volume of domain-specific examples, we can enhance the model's ability to accurately generate GAML scripts from UML representations – this is particularly important since the fine-tuning dataset used for Llama3/Llama3.1 did not have to be very large since the number of parameters was less - did not need to be large to affect model behaviour enough.

Another area for improvement involves providing additional domain-specific knowledge to the model during fine-tuning. By using academic texts related to simulation modelling, we can imbue the model with deeper contextual understanding. I am not sure to what extent this is necessary for the Mistral-NeMo, in my run of the EABSS

script, it did not have an issue referencing academic works. If such fine-tuning is deemed necessary, I would recommend providing more in-depth texts than abstracts – perhaps excerpts from text books, so the model learns like a human does during fine-tuning.

A domain that I could not explore in this project due to time and hardware limitations is Mixture of Experts (MoE) models. MoE models comprise multiple smaller models, each specialised in a sub-domain. For instance, one model could focus on simulation modelling literature, while another could specialise in GAML generation. This architecture is particularly appealing because it enables more efficient handling of domain-specific tasks. Each expert model contributes its specialised knowledge, which can be combined to improve overall performance across diverse tasks. I believe this approach would neutralise the inability to generalise that arises in a single model fine-tuned across various different tasks. References in literature suggest that MoE architectures significantly reduce computational costs while improving model adaptability and task-specific precision[12]. Additional background reading on MoE models can be found on a Hugging Face introductory blog post[13].

As mentioned previously, the tools, methods, and design patterns for natural language-to-script translation, can be adapted easily to different frameworks such as the AgentPy Python library. This cross-compatibility will be essential in extending this research beyond the GAMA platform, potentially improving simulation workflows in fields ranging from epidemiology to environmental modelling.

8.2 Personal Reflection

Personally, this internship was my first experience working in a research-oriented environment. I gained in-depth knowledge about both simulation modelling and generative AI. Through background reading, I developed an understanding of agent-based modelling and its various applications. I very much enjoyed collaborating with my supervisor, Prof. Siebers, and gained a lot of tacit knowledge about agent-based modelling from our discussions. Furthermore, I developed a deeper understanding of the transformer-based architecture that underpins the state-of-the-art LLMs, such as Llama3.1, and the natural language processing techniques necessary to prepare training data for language models. I also had the opportunity to expand my technical skill set, learning GAML as well as the llama.cpp inference library, LangChain and the workings of the Compute Unified Device Architecture (CUDA) API for fine-tuning. During the implementation and testing phases of the project, I encountered breaking changes with various libraries, solving these issues gave me a much deeper understanding of the technologies I used, as well as how they fit together. I learnt to plan and write for an academic audience, and I enjoyed using LaTeXfor the first time.

Overall, my internship experience has reinforced my commitment to pursue a postgraduate research degree in computer science. This experience has enhanced my critical thinking, time management, and communication skills. I have a clearer understanding of how I can contribute to academia, particularly in the fields of AI and

[12]Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... & Sayed, W. E. (2024). Mixtral of experts. arXiv:2401.04088.

[13]https://huggingface.co/blog/moe

Deep Learning.

8.3 Feedback

My internship experience was incredibly enjoyable and very rewarding, and I would definitely recommend this opportunity to my peers.

One small challenge I experienced was accessing powerful computing resources. Running and fine-tuning LLMs can be resource-intensive, and having access to more powerful Graphics Processing Units (GPUs) would have allowed for faster on-device model training and experimentation. There were times when I had to rely on cloud-based virtual machines (with 24GB RAM and a non-NVIDIA GPU) which have additional overhead and are time-consuming to configure. I appreciate there is a strong demand for additional processing power due to the concentration of research in Deep Learning at the moment. Perhaps it would be possible for the department to procure a small number of GPUs if similar projects are planned for the future?

9 Appendix

9.1 A1: Git Repository

https://github.com/senertopaloglu/epsrc-summer-internship
Folders:

- GAML-Data: GAML general knowledge dataset.
- complete-finetune-datasets: "final" form of all the task-specific datasets collated to fine-tune the model.
- debugging-llama: some evidence of llama3.1 hallucination.
- fine-tune-training-losses: recordings of training accuracy data, as evidence of fine-tuning.
- finetune-scripts-unsloth: unsloth fine-tuning scripts for llama3, llama3.1 and Mistral-NeMo (I have made changes to these, particularly to solve dependency conflicts and training parameter tunings).
- mistral-nemo: evidence of isolated training with mistral-nemo.
- news-summaries-pdf: document corpora for RAG proof of concept.
- report: please ignore, this contains a very early version of the report.
- synthetic-dataset-json: contains synthetic datasets in json form.

Files in top-level directory:

- Modelfile_template: a template modelfile (schema) for instantiating models to run in Ollama server.
- finetune-dataset.json: please ignore, this is old.

- mygpt.py: I initially experimented with my own GPT-based model, before I quickly realised it would not work.
- news-summary-{x}.docx: please ignore, these are the .docx equivalent of the news summary PDFs.
- ragapp.py: my RAG pipeline, with Streamlit UI.
- requirements.txt: to quickly retrieve all dependencies with correct versions - ideal if you would like to use a virtual Python environment to run the project.

9.2 A2: Project Planning Diagram

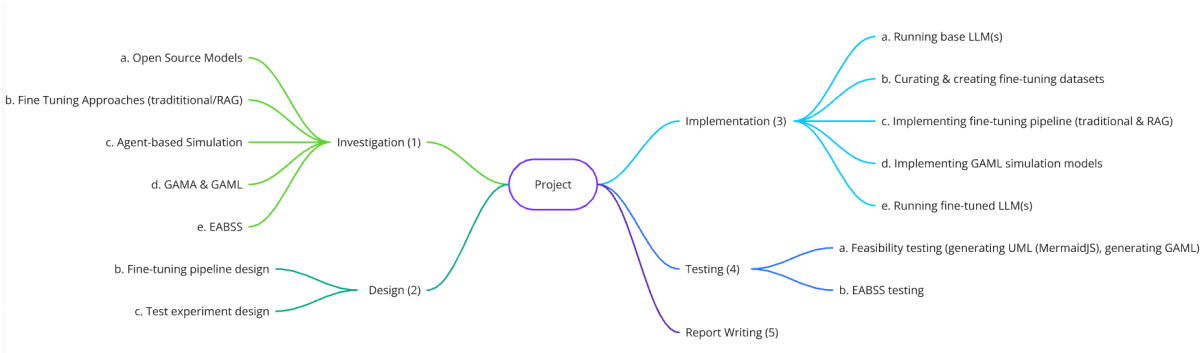

Figure 2: Project phases and their sub-tasks

9.3 A3: Mistral-NeMo Zero Shot UML Diagrams

```
sb_topaloglu1@instance-20240729-100526:   $ ollama run
mistral-nemo:latest
>>> Generate UML state diagrams, using Mermaid JS, for each important
actor in the prompt provided:  1)Prompt:  A cybersecurity firm detects a
threat, analyzes it, and deploys countermeasures to protect client data.
```

9.4 A4: Mistral-NeMo Zero Shot GAML Generation

Please see the relevant PDF inside the Report ZIP folder.

9.5 A5: Design Patterns

Insert input here.

9.5.1 Preparation

1. Work through each task list in the given order during the entire conversation. Answer with just "yes" if you understand or "no", if you don't understand.

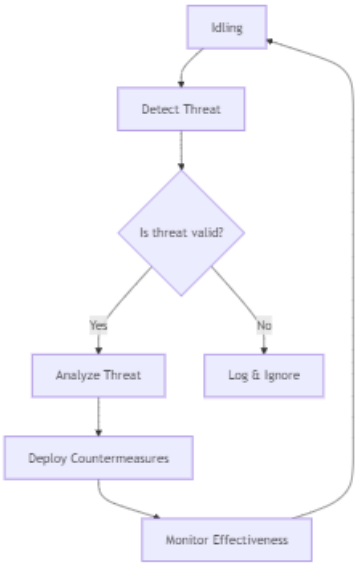

Figure 3: UML state diagram for *CybersecurityFirm* actor

2. Provide ALL RESPONSE CONTENT without asking questions during the entire conversation. DO NOT print any keys (Example: Use "Example" AND NOT "{key-example}") during the entire conversation. Use a "scientific tone" during the entire conversation, unless instructed otherwise. Do you understand? say "yes" or "no".

3. Memorise "Unified Modelling Language" as {key-uml}. Memorise "Agent-Based Social Simulation (ABSS) Study" as {key-studyType}. Got it? Say "yes" or say "no".

9.5.2 Analysis

Problem Statement

1. Take on the "role" of a "Sociologist" with experience in "Agent-Based Social Simulation". Memorise this role as {key-role1}. Do you understand? say "yes" or "no".

2. Define the "topic" of the memorised {key-studyType} as "The goal of this study is to generate IDEAS for using ADAPTIVE ARCHITECTURE in futuristic MUSEUMS within an exhibition room that is visited by ADULTS and CHILDREN. The adaptive architecture consists of 2 kinds of artefacts: (1) 2 large wall-mounted SCREENS on which SMART CONTENT WINDOWS move with the visitors and (2) a SMART PARTITION WALL that creates a dynamic and flexible exhibition environment by continuously analysing visitor movement, making real-time decisions, and physically reconfiguring the space to optimise the experience for everyone. The adaptive architecture artefacts represent AI-DRIVEN INTELLIGENT OBJECTS.". Memorise this topic as {key-topic}. Do you understand? say "yes" or "no".

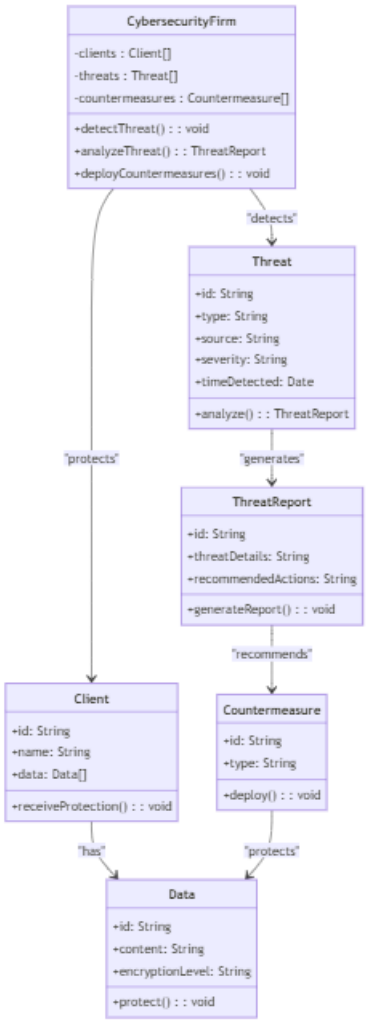

Figure 4: UML class diagram

3. Memorise "Exploratory" as {key-researchDesign}. Memorise "Social Studies" as {key-domain}. Memorise "Human Behaviour" as {key-specialisation}. Do you understand? say "yes" or "no".

4. Using a "scientific and inspirational tone". Define a novel and creative "context" for the memorised {key-topic} in 200 WORDS (if possible), then memorise this context as {key-context}.

5. Define "stakeholders" for the memorised {key-topic}, to participate in a co-creation role-play game. Memorise these stakeholders together with their personas as {key-stakeholders} (you do not need to create names for personas).

6. You will write a Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. Display "Problem Statement" as markdown 'heading level 3'. Structure: 1. display memorised {key-role1}, 2. display memorised {key-topic}, 3. display memorised {key-researchDesign}, 4. display memorised {key-domain}, 5. display memorised {key-specialisation}, 6. display memorised {key-context}, 7.

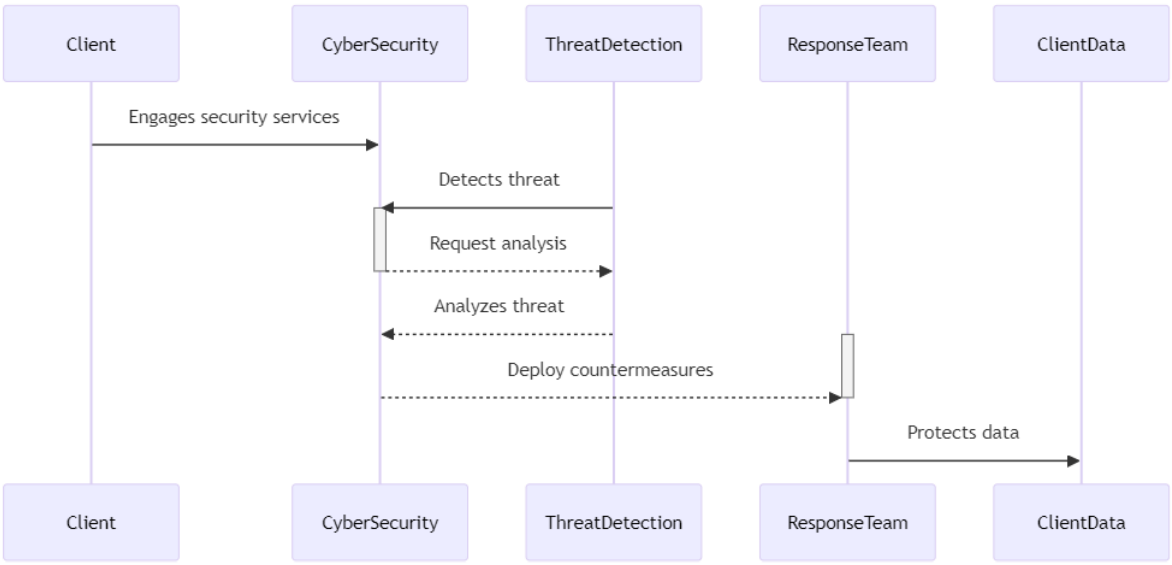

Figure 5: UML sequence diagram (from the prompt, state & class diagrams generated previously).

display memorised {key-stakeholders}. Make sure to replace the keys inside "{}" with their values.

7. List 2 potential aims that satisfy the viewpoints of all participating memorised {key-stakeholders}. Define 5 "keywords" for the memorised {key-studyType} in the context of the memorised key-topic in the form of a comma-separated list. Memorise these 5 keywords as {key-keywords}.

8. Define the "title" for the memorised {key-studyType} in the context of the memorised {key-topic} in 12 WORDS (if possible). Memorise this title as {key-title}.

9. Define the "aim" for the memorised {key-studyType} in the context of the memorised {key-topic} in 40 WORDS (if possible). CONSIDER the memorised {key-potentialAims} in your definition. Use a "scientific tone". Memorise this aim as {key-aim}.

10. Using exactly the same markdown code as what was generated before, append to the end of the document the following sections/chapters: display memorised {key-title}. display memorised {key-aim}. display memorised {key-keywords}. Make sure to replace the keys inside "{}" with their values.

Study Outline

1. Simulate and play a co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential Agent-Based Social Simulation (ABSS) objectives for the study considering the pros and cons. Use a "debating tone". The moderator focuses on 1 novel RANDOM question. Provide the question and the details of the controversial discussion. Agree on 4 potential ABSS objectives that satisfy the view of all participating memorised

key-stakeholders. Memorise these potential ABSS objectives as {key-potentialObjectives}. Did you memorise? state just "yes" or "no".

2. Propose 3 criteria for ranking the 4 potential ABSS objectives to support the decision which objectives to carry forward. Use a "scientific tone".

3. Define 2 "ABSS objectives" for the memorised {key-studyType} in the context of the memorised {key-topic}. CONSIDER the memorised {key-potentialObjectives} in your definitions. List the objectives with 2 relevant performance measures for each objective. Memorise these 2 objectives together with the performance measures as {key-objectives}.

4. Play a new co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential ABSS hypotheses for the study considering the pros and cons. Use a "debating tone". The moderator focuses on 1 novel RANDOM question. Provide the question and the details of the controversial discussion. Agree on a few potential ABSS hypotheses that satisfy the view of all participating memorised {key-stakeholders}. Memorise these potential ABSS hypotheses as {key-potentialHypotheses}. Propose 3 criteria for ranking the 4 potential ABSS hypotheses to support the decision which hypotheses to carry forward. Use a "scientific tone".

5. Define 2 "ABSS hypotheses" for the memorised {key-studyType} in the context of the memorised {key-topic}. The hypotheses MUST not be related to the memorised {key-objectives}. CONSIDER the memorised {key-potentialHypotheses} in your definitions. Memorise these 2 hypotheses AND the performance measures as {key-hypotheses}. Did you memorise? state just "yes" or "no".

6. Play a new co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential ABSS experimental factors for the study considering the pros and cons. Use a "debating tone". The moderator focuses on 1 novel RANDOM question. Provide the question and the details of the controversial discussion. Agree on 6 potential ABSS experimental factors that satisfy the view of all participating memorised {key-stakeholders}. Memorise these potential ABSS experimental factors as {key-potentialExperimentalFactors}. Propose 3 criteria for ranking the 6 potential ABSS experimental factors to support the decision which experimental factors to carry forward. Use a "scientific tone".

7. Define 3 "ABSS experimental factors" for the memorised {key-studyType} in the context of the memorised {key-topic}. You ALWAYS must satisfy the following 2 requirements for defining experimental factors: 1) The experimental factors need to be useful for creating memorised {key-studyType} scenarios. 2) CONSIDER the memorised key-objectives and the memorised {key-hypotheses} for defining the experimental factors. MAKE SURE TO CONSIDER the memorised {key-potentialExperimentalFactors} in your definitions. List the experimental factors with 1 value range for each experimental factor. 1 of them MUST use a 'nominal scale' AND 1 of them MUST use an 'ordinal scale' AND 1 of them MUST use a 'ratio scale'. Memorise these 3 experimental factors together with the value ranges as {key-experimentalFactors}.

8. Play a new co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential ABSS outputs for the study considering the pros and cons. Use a "debating tone". The moderator focuses on 1 novel RANDOM question. Provide the question and the details of the controversial discussion. Agree on 6 potential ABSS outputs that satisfy the view of all participating memorised {key-stakeholders}. Memorise these potential ABSS outputs as {key-potentialOutputs}. Propose 3 criteria for ranking the 6 potential ABSS outputs to support the decision which outputs to carry forward. Use a "scientific tone".

9. Define 3 "ABSS outputs" for the memorised {key-studyType} in the context of the memorised {key-topic}. You ALWAYS must satisfy the following 2 requirements for defining outputs: 1) Some outputs need to be useful for measuring if the memorised {key-objectives} have been satisfied. 2) Some outputs need to be useful for accepting or rejecting the memorised {key-hypotheses}. CONSIDER the memorised {key-potentialOutputs} in your definitions. List the outputs and explain links to the memorised {key-objectives} OR the memorised {key-hypotheses} in 1 concise sentence each. Memorise these 3 outputs together with the links as {key-outputs}.

10. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Study Outline" as markdown 'Heading Level 3 ###'. Then: 1. List: a) the "objective" in the context of the memorised {key-studyType} in 1 concise sentence, b) the "hypothesis" in the context of the memorised {key-studyType} in 1 concise sentence, c) the "experimental factor" in the context of the memorised {key-studyType} in 1 concise sentence, d) the "output" in the context of the memorised {key-studyType} in 1 concise sentence. 2. Display your 3 criteria for ranking the 4 potential ABSS objectives to support the decision which objectives to carry forward (you have done this before, just output your previous answer here in markdown). 3. list memorised {key-objectives}. 4. List the 3 criteria you proposed for ranking the 6 potential ABSS experimental factors to support the decision which experimental factors to carry forward (you have done this before, just output your previous answer here in markdown). 5. List the memorised {key-experimentalFactors}. 6. List the memorised {key-outputs}. Make sure to replace the keys inside "{}" with their values.

Model Scope

1. Now take on the additional "role" of a "Senior Software Developer" with experience in the "Unified Modelling Language (UML)", memorise this role as {key-role2}.

2. Play a co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential some ABSS UML actors for the study considering the pros and cons. Use a "debating tone". The moderator focuses on 1 novel RANDOM question. Provide the question and the details of the controversial discussion. Agree on some potential ABSS UML actors that satisfy

the view of all participating memorised {key-stakeholders}. Memorise these potential ABSS UML actors as {key-potentialUMLActors}. Propose 3 criteria for ranking the potential ABSS UML actors to support the decision which ABSS UML actors to carry forward. Use a "scientific tone".

3. CONSIDER the memorised {key-potentialUMLActors} in your definitions. Choose all the important "ABSS UML actors", if they are all important, choose them all. Remember the memorised {key-topic}. Memorise these UML actors together with a persona description as {key-umlActors}. Did you memorise? simply answer with just "yes" or "no".

4. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Define 15 "real-world elements" with relevance to the memorised {key-topic}. Make sure to replace the keys inside "" with their values. You ALWAYS must satisfy the following 8 requirements for defining real-world elements: 1) Consider what 'real-world elements' are needed to represent in the model scope and to satisfy the memorised {key-aim}. 2) ALL 4 memorised {key-umlActors} MUST BE REPRESENTED. 3) At least 2 Physical Environment elements MUST be present. At least 2 Social Aspect elements MUST be present. At least 2 Psychological Aspect elements MUST be present. At least 2 Misc elements MUST be present. 4) Consider the memorised {key-context}. 5) Consider all nouns in the conversation history. 6) Each element can only be in 1 category. 7) Social Aspect elements MUST describe theories of social behaviour. 8) Psychological Aspect elements MUST describe theories of psychological behaviour. Feel free to be creative and add your ideas. Categorise the 'real world elements' into Actors, Physical Environment, Social Aspects, Psychological Aspects, and Misc. TABLE MUST include 15 rows. Organise all 15 elements into categories and provide a brief explanation. Memorise these 15 elements and explanations as {key-explanations}. List the memorised {key-topic} relevant real-world elements in the form of table rows. Provide a column for Category. Provide a column for Sub-Category. Provide a column with the memorised {key-explanations}. Provide a column with concise justifications in ABOUT 25 WORDS. Memorise this table as {key-modelScope}.

5. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Define 4 models for implementing elements of the memorised {key-modelScope}. Provide 1 social model AND 1 behavioural model AND 1 psychological model AND 1 technical model. Find relevant theoretical models in the SCIENTIFIC LITERATURE. Provide a full EXISTING UP-TO-DATE scientific paper (conference or journal) or book REFERENCE in HARVARD STYLE for each in a separate column. Memorise these 4 model details together with a description and the relevant reference as {key-implementationModels}.

6. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Model Scope" as Markdown 'Heading Level 3 ###'. Then: 1) List memorised {key-role2}, 2) List definitions of: "model scope" in the context of the memorised {key-study} and "UML actor" in the context of the memorised {key-uml}, 3) List each of the {key-potentialUMLActors}, with their personas, 4) Display {key-modelScope}

table, 5) Display {key-ImplementatonModels}. Make sure to replace the keys inside "{}" with their values.

Key Activities

1. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Define 2 "UML user stories" for each of the 4 memorised {key-umlActors} (Example: As an 'actor' I want 'action' so that 'achievement'). Memorise ALL 8 UML user stories as {key-umlUserStories}. Translate the memorised {key-umlUserStories} into UML use cases. Memorise ALL 8 UML use cases as {key-umlUseCases}. List ALL 8 memorised {key-umlUserStories} and ALL 8 corresponding memorised {key-umlUseCases} side by side in two columns inside the table sorted by memorised {key-umlActors}. Memorise this table as {key-umlUseCaseTable}.

2. Generate a script for a 'comprehensive use case diagram' in "Mermaid.js". Use the memorised {key-umlActors} as UML actors. Remove all brackets from the actor names. Use the memorised {key-umlUseCases} as UML use cases. You ALWAYS must satisfy the following 4 requirements for defining the use case diagram: 1) Each UML actor MUST be linked to at least 1 UML use case. 2) Each UML use case MUST be linked to at least 1 UML actor OR MUST be pointing to at least 1 other UML use case. 3) There is no UML actor to UML actor interaction. 4) A UML use case CAN be linked to multiple UML actors. Add relationships with 'detailed descriptors'. Start the script with "graph LR". DO NOT Add subgraphs. Use the following format (Example for actor A((actor))) AND (Example for use case A([activity])) AND (Example for relationship: A –¿—activity— A1). Feel free to be creative and add your ideas. Memorise this Mermaid.js script as {key-mermaidKeyActivitiesScriptDraft}

3. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Key Activities" as Markdown 'Heading Level 3 ###'. Then: 1) list definitions of "user story" in the context of the memorised {key-uml} and "use case" in the context of the memorised {key-uml}, 2) display the memorised {key-umlUseCaseTable}, 3) display {key-mermaidKeyActivitiesScript}. Make sure to replace the keys inside "" with their values.

9.5.3 Design

Archetypes

1. Now, take on the additional third role of a "Marketing Expert" with experience in "Customer Management". Memorise this role as {key-role3}.

2. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Archetypes" as Markdown 'Heading Level 3 ###'. Then: 1) Display memorised {key-role3}, 2) List: definitions of "archetype" and "categorisation schema" in the context of the

memorised {key-job} in 1 sentence. Make sure to replace the keys inside "" with their values.

3. Play a co-creation role-play game in which all the memorised {key-stakeholders} discuss with each other potential archetypes for each of the memorised {key-umlActors} individually. Use a "debating tone". Provide 6 potential archetypes FOR EACH of the 6 memorised {key-umlActors} including 3 criteria to identify them. Agree on 2 potential archetypes FOR EACH of the memorised {key-umlActors} that satisfy the view of all participating memorised {key-stakeholders}. Memorise these potential archetypes as {key-potentialArchetypes}. Use a "scientific tone".

4. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Define 4 categorisation schemata, 1 for each of the 4 memorised {key-umlActors}. You ALWAYS must satisfy the following 5 requirements for defining categorisation schemata: 1) Each of the 4 tables must be based on memorised {key-umlActors} behaviour, preferences, characteristics, demographics, habits, and the likelihood of actions. 2) Each of the 4 tables MUST contain 3 characteristic rows. 3) Characteristics inside a table MUST use 1 'nominal scale' AND MUST use 1 'ordinal scale' AND MUST use 1 'ratio scale'. 4) Characteristics inside a table MUST provide value ranges for these scales. 5) Table columns: Actor Category, Individual Characteristic, Scale, Value Range. CONSIDER the memorised {key-potentialArchetypes} in your definitions. Memorise ALL 4 categorisation schemata as {key-categorisationSchemata}. Define 4 models for implementing elements of the memorised {key-modelScope}. Provide 1 social model AND 1 behavioural model AND 1 psychological model AND 1 technical model. Find relevant theoretical models in the SCIENTIFIC LITERATURE. Provide a full EXISTING UP-TO-DATE scientific paper (conference or journal) or book REFERENCE in HARVARD STYLE for each in a separate column. Memorise these 4 model details together with a description and the relevant reference as {key-implementationModels}.

Agent & Object Templates

1. Take on the additional "role" of a "Senior Software Developer" with experience in the "Unified Modelling Language". Memorise this role as {key-role4}.

2. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Agent and Object Templates" as Markdown 'Heading Level 3 ###'. Then: 1) List memorised {key-role4}, 2) List: definitions of "class", "class diagram", "state chart", "state variable" in the context of the memorised {key-uml}.

3. Generate a script for a 'comprehensive class diagram' in "Mermaid.js". From the memorised {key-modelScope} use the Actor and Physical Environment Categories for class names. Define a class for each of these. Add more classes. IN ADDITION Add collective classes for individual actors where appropriate (Example: bird ¿ flock. Example: grape ¿ bunch). DO NOT use the examples. DO NOT create abstract classes. DO NOT create classes with the same name. Delete all getter

and setter operations. Add additional attributes and operations. DO NOT define relationships. Call the Main class ArtificialLab. Create only 1 ArtificialLab class. Define arrays for ALL Actor objects and ALL Physical Environment objects as attributes INSIDE the ArtificialLab class. Include MULTIPLE "summary statistics" operations for testing memorised {key-objectives} AND memorised {key-hypotheses} INSIDE the ArtificialLab class. Remove ALL lines from the script that contain "//". Feel free to be creative and add your ideas. Memorise this mermaid.js script as {key-mermaidClassDiagramScriptDraft}.

4. INCREASE COMPLEXITY of {key-mermaidClassDiagramScriptDraft}. Add additional attributes. Add additional operations. Add additional relationships between classes. Provide CONNECTIONS between classes. Critically REFLECT and IMPROVE the script based on your reflection. Find and remove any mermaid.js script errors. Memorise this mermaid.js script as {key-mermaidClassDiagramScript}.

5. For EACH INDIVIDUAL of the 4 memorised {key-umlActors} generate a script for a 'comprehensive state machine diagram' in "Mermaid.js". Define their states and state transitions between these states. Add text to the transitions to describe what they represent (Example: 's1 –¿ s2: Generate A transition'). Consider the start state (Example: '[*] –¿ s1'). Consider stop state (Example: 's1 –¿ [*]'). Add a comment as line 0 with the actor's name (Example: '%% Name: Actor'). You ALWAYS must satisfy the following 2 requirements for defining the state machine diagram: 1) ALL states MUST have AT LEAST 1 entry transition AND 1 exit transition. 2) Provide a memorised {key-uml} note for every individual state, explaining the related state (Example: 'note left of [actual state] : Informative text'). Memorise this mermaid.js script as {key-mermaidStateMachineDiagramsScriptDraft}.

6. INCREASE COMPLEXITY of {key-mermaidClassDiagramScriptDraft}. If necessary; add additional attributes, add additional operations, add additional relationships between classes. Provide CONNECTIONS between classes. Critically REFLECT and IMPROVE the script based on your reflection. Find and remove any mermaid.js script errors. Memorise this mermaid.js script as {key-mermaidClassDiagramScript}.

7. For EACH INDIVIDUAL of the memorised {key-umlActors}, generate a script for a 'comprehensive state machine diagram' in "Mermaid.js". Use "stateDiagram-v2". Define their states and state transitions between these states. Add text to the transitions to describe what they represent (Example: 's1 –¿ s2: Generate A transition'). Consider the start state (Example: '[*] –¿ s1'). Consider stop state (Example: 's1 –¿[*]'). You ALWAYS must satisfy the following 2 requirements for each state machine diagram: 1) ALL states MUST have AT LEAST 1 entry transition AND 1 exit transition. Memorise this mermaid.js script as {key-mermaidStateMachineDiagramsScriptDraft}.

8. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Iterate through the memorised {key-mermaidStateMachineDiagramsScript} and define up to 3 variables FOR EACH diagram for keeping track of continuous changes of agent and object states

(often a level of something: Example 'tiredness level'). Create a "state variables table" with all state variables (columns: state machine diagram, variable, unit, definition of variable. Example: State machine shopper, satisfaction level, scale 1-10, represents the satisfaction level). Do NOT include the example. Memorise this state variables table as {key-stateVariablesTable}.

9. Create a Markdown table for the following (DO NOT use "¡br¿", IGNORE ALL space limitations): Create a "state transitions table" with all state diagram transitions (columns: actor, start state, end state, type of transition, detail). Detail MUST be 1 concise sentence. Possible TYPE OF TRANSACTION: timeout, condition, rate. Memorise this state transitions table as {key-stateTransitionsTable}.

Interactions

1. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Interactions" as Markdown 'Heading Level 3 ###'. Then: 1) Display definition of the term "sequence diagram" in the context of the memorised {key-uml} in 1 concise sentence.

2. Generate a script for a 'comprehensive sequence diagram' in "Mermaid.js". Use the memorised {key-mermaidClassDiagramScript} for identifying all relevant ACTORS and OBJECTS. Define interactions between the 'different actors' and 'actors and objects' FOR ALL memorised {key-umlUseCases}. Use cases should be presented as NOTES on the vertical axis above each use case representation. Actors and 7 objects should be as lifelines on the horizontal axis. EXCHANGE "participant" with "actor" for ALL ACTORS (Example: actor example). DO NOT use aliases. Present ACTIVATIONS and DEACTIVATIONS for actors and objects on the LIFELINES. Each use case should be connected to the corresponding sequence of events. Add the prefix "The" to all ACTOR and OBJECT names. IGNORE the "ArtificialLab". Memorise this mermaid.js script as {key-mermaidSequenceDiagramScriptDraft}.

3. Generate a script for a 'comprehensive sequence diagram' in "Mermaid.js". Use the memorised {key-mermaidClassDiagramScript} for identifying all relevant ACTORS and OBJECTS. Define interactions between the 'different actors' and 'actors and objects' FOR ALL memorised {key-umlUseCases}. Use cases should be presented as NOTES on the vertical axis above each use case representation. Actors and 7 objects should be as lifelines on the horizontal axis. EXCHANGE "participant" with "actor" for ALL ACTORS (Example: actor example). DO NOT use aliases. Present ACTIVATIONS and DEACTIVATIONS for actors and objects on the LIFELINES. Each use case should be connected to the corresponding sequence of events. Add the prefix "The" to all ACTOR and OBJECT names. IGNORE the "ArtificialLab". Memorise this mermaid.js script as {key-mermaidSequenceDiagramScript}.

9.5.4 Implementation

Approach 1

1. Using the information provided, I require a COMPLETE, FULLY IMPLEMENTED GAMA Markup Language (GAML) (NOT XML) simulation script for the key topic to run in the GAMA simulation engine. A reminder: {key-topic} is "applying the predator-prey cycle.The predator-prey model simulates the interactions between two species: predators and preys, within a grid-like environment where vegetation cells represent areas where grass grows. Preys feed on the grass, which regenerates at each simulation step, while predators hunt and consume preys to gain energy. Both species follow specific behaviors: they move to neighboring cells, consume available resources (preys or grass), and lose energy over time. If their energy drops too low, they die. However, if they maintain sufficient energy, they can reproduce, contributing to population dynamics.". The GAML script must start with "model" keyword, so the model starts with "model <¡APPROPRIATE-MODELNAME>". Make sure to generate an "experiment {...}" block, with "output {...}" and "display {...}" blocks inside it. Make sure to initialise any parameters used in the "experiment" block, in the "global {...}" block, if you think a modelling specialist would like to alter its value. Use "species" to define actors/systems/species which you may have seen inside {key-mermaidClassDiagramScript}. Use interactions in {key-mermaidSequenceDiagramScript} to connect the respective species. DO NOT implement the artificial lab. To implement actions made by species, use "action" if action is not expected to be conducted at each timestep or "reflexes" if action is expected to be conducted at each timestep. Begin to generate the GAML script, remember it must be syntatically correct, valid (for example using "<-" to initialise variables inside species) and be FULLY IMPLEMENTED (all species, actions and reflexes). Make sure to add comments to the code. Memorise this as {key-gamlScriptDraft}.

Approach 2

1. Using the information provided, I require a COMPLETE, FULLY IMPLEMENTED GAMA Markup Language (GAML) (NOT XML) simulation script for the key topic to run in the GAMA simulation engine. A reminder: {key-topic} is "applying the predator-prey cycle.The predator-prey model simulates the interactions between two species: predators and preys, within a grid-like environment where vegetation cells represent areas where grass grows. Preys feed on the grass, which regenerates at each simulation step, while predators hunt and consume preys to gain energy. Both species follow specific behaviors: they move to neighboring cells, consume available resources (preys or grass), and lose energy over time. If their energy drops too low, they die. However, if they maintain sufficient energy, they can reproduce, contributing to population dynamics.". First: The GAML script must start with "model" keyword, so the model starts with "model <¡APPROPRIATE-MODELNAME>".

2. Then, generate a "global {...}" block, if there are any important global variables, or variables you think a modelling specialist would like to be able to alter its value.

3. Building upon the generated GAML script, generate "species" blocks for each actors/systems/species which you may have seen inside {key-mermaidClassDiagramScript}. Use interactions in

{key-mermaidSequenceDiagramScript} to connect the respective species. DO NOT implement the artificial lab. To implement actions made by species, use "action" if action is not expected to be conducted at each timestep or "reflexes" if action is expected to be conducted at each timestep.

4. It is important to be able to run experiments. Generate the following at the end of the generated GAML script; an "experiment {...}" block, with "output {...}" and "display {...}" blocks inside it, this will allow the user to run experiments. Make sure to use any variables that have been initialised in the "global" block, if you think a modelling specialist would like to alter its value.

5. Make sure every species, reflex and action in {key-gamlScriptDraft} is fully implemented. For example if a species named "species1" has a action/reflex named "move()" make sure "move()" is implemented. Make sure any function calls or references to species have implementations. REFLECT and IMPROVE the script based on your reflection. Find and remove any GAML errors. Make sure to add comments to the code. Memorise this script as {key-gamlScript}.

9.5.5 Conclusion

1. Now, you will write a new, different Markdown document using the memorised keys (separate each section using headers). Only show the final, resulting markdown file code from this prompt. First, output "Conclusion" as Markdown 'Heading Level 3 ###'. Then: Write a 300 WORD (if possible) conclusion of the entire conversation history. Provide 3 paragraphs, testifying whether the aim has been achieved, answering the questions related to the objectives and hypotheses, providing 2 identified limitations of the current work, and proposing 2 ideas for future work, based on these limitations. Also mention what the memorised {key-gamlScript} achieves and how it fits into the hypotheses and objectives. Memorise this conclusion as {key-conclusion}.

9.6 A6: Example

Please see the relevant PDF inside the Report ZIP folder, for the documented EABSS script test using design patterns (with implementation approach 1).