Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

SCALABLE, RULE-BASED, PARALLEL DISCRETE EVENT BASED AGENTIC AI SIMULATIONS

Atanu Barai[1], Stephan Eidenbenz[1], and Nandakishore Santhi[1]

[1]Los Alamos National Laboratory, Los Alamos, NM, USA

ABSTRACT

We introduce a novel parallel discrete event simulation (PDES)-based method to couple multiple AI and non-AI agents in a rule-based manner with dynamic constraints while ensuring correctness of output. Our coupling mechanism enables the agents to work in a co-operative environment towards a common goal while many sub-tasks run in parallel. AI agents trained on vast amounts of human data naturally model complex human behaviors and emotions easily – this is in contrast to conventional agents which need to be burdensomely complex to capture aspects of human behavior. Distributing smaller AI agents on a large heterogeneous CPU/GPU cluster enables extremely scalable simulations tapping into the collective complexity of individual smaller models, while circumventing local memory bottlenecks for model parameters. We illustrate the potential of our approach with examples from traffic simulation and robot gathering, where we find our coupling of AI/non-AI agents improves overall fidelity.

1 INTRODUCTION

Artificial intelligence (AI) is transforming industries and driving breakthroughs across science, energy, economics, and security. While AI systems can perform some tasks once limited to humans, most models operate in isolation, unable to harness collective knowledge and parallel synergy, hindering innovation.

The growing complexity of problems in science and other domains demand solutions that leverage the strengths of multiple AI and non-AI systems, all coupled together. While AI agents excel in repetitive data processing, content generation, and adaptive decision making, non-AI agents such as human experts, and rule-based deterministic systems offer domain expertise, practical constraints, and ethical/societal considerations that the AI systems can't yet fully encapsulate. On the other hand rule-based agents cannot easily reflect complex human behavior necessary to model social scenarios whereas the AI agents inherently can do so. Working together, these systems can create more adaptable, context-aware systems solutions that can address a broad spectrum of modeling challenges by pooling knowledge, sharing information, and coordinating actions that would be beyond the reach of any individual agent model. Thus, these types of systems promise enhanced problem solving capabilities and greater adaptability to complex environments including autonomous transportation, drug-discovery, education, socioeconomic studies etc.

The rise of large language models (LLMs) (Bommasani et al. 2021) has made structured collaboration between AI and non-AI agents increasingly desirable. While LLMs excel at tasks like human-like responses, code generation and text summarization, they often struggle with complex reasoning that demands goal-directed, organized, multi-step logic-based processes. *Agentic AI* systems address this gap by integrating diverse agents and structuring workflows to emulate human-like iterative reasoning.

However, developing a robust coupling methodology to enable multi-agent systems that also include AI agents is challenging due to several factors. Issues such as communication methodologies, timing/coordination among the agents, conflict resolution, handling biases, security considerations, and most importantly extreme scalability to handle a very large number of agents, each with high hardware-resource demands, must be addressed to ensure an effective collaboration strategy.

This paper introduces a robust parallel discrete event simulation (PDES) based methodology to couple AI and non-AI agents into rule-based, cohesive, parameter-rich multi-agent systems. Here the PDES framework serves as an underlying infrastructure enabler for agentic AI addressing communication, coordination, synchronization, and scalability issues, lifting the burden from the user. It helps break problems into smaller, manageable steps, involving both AI and non-AI agents to solve each stage in a causal event-driven manner. Using the open-source *Simian* (Santhi, Eidenbenz, and Liu 2015) PDES engine, agents are coupled and deployed through a user-transparent parallelization mechanism which effectively uses the message passing interface (MPI) library. Each simulation agent is implemented as an *entity* object assigned to an MPI rank to be executed in parallel. Through a *reqService* process, future events are scheduled at the corresponding *entity* to be processed by a user-defined *event-handler*. Although here we use Simian PDES to couple AI/non-AI agents, any other discrete event simulation framework may as well be utilized.

Modern AI models like Llama3 (Dubey et al. 2024) and GPT4 (OpenAI et al. 2024), require scarce monolithic graphics processing units (GPUs) with very large compute and memory capacity for parallel computation due to their large-scale training on web data resulting in billions of parameters, making *local inference* of large models costly and out of reach for many researchers. A PDES-based methodology addresses this problem by enabling the parallel deployment of thousands of smaller AI agents coupled with many non-AI agents to work together to solve a particular problem with less powerful and inexpensive hardware. Non-AI agents can run on the CPUs hosting the GPUs, thus enabling resource efficiency.

To demonstrate our approach more thoroughly, we simulate (i) a transportation and (ii) a gathering problem by coupling small language model (SLM) based AI agents with non-AI rule-enforcing agents. We classify models with 16B parameters or fewer as SLMs and larger ones as LLMs. This distinction is based on current technology, where 16B and smaller models can run on consumer-grade GPUs using parameter quantization. The general simulation approach by coupling AI/non-AI agents is the following:

1. We break down the simulation scenario into multiple more manageable, yet smaller steps,
2. SLM agents in parallel make human-like decisions to solve a step based on current conditions,
3. Solution space might be constrained by providing SLM agents with a list of options to choose from,
4. Non-SLM agents collate and then inspect SLM responses, performs domain-specific deterministic calculations, and provides feedback in the form of follow-on prompts to the distributed SLM agents,
5. Collective model runs under the umbrella of a distributed-memory parallel discrete event simulation.

These simulations might be performed by an LLM or rule-based non-AI agent acting on their own. But most large models require GPU acceleration with very large local memory to run, which is in turn very expensive and often scarce. In contrast, in our approach, we use SLMs that require far less powerful GPUs with low local memory requirements, and can even run on integrated GPUs on the CPU-fabric. Our method can optionally make multiple SLMs work together to get more nuanced responses (Shen et al. 2024) for the individual steps. It also has a simple verification step built in as the non-AI agent checks intermediate SLM answers based on rules, or otherwise constrained by a pre-vetted options list; thus making the overall system behavior much more trustworthy and human-like.

Incorporating probabilistic AI models such as the language models inherently adds rich non-deterministic characteristics to the simulation. Thus, simulation scenarios with emergent non-deterministic or somewhat random agent decision requirements can benefit the most from our proposed coupling. Indeed, we show at trace level detail, how our collaborative approach improves performance in the traffic simulation and gathering problems.

The rest of the paper is organized as follows: Section 2 summarizes the relevant related works and sets this work apart from those. Section 3 covers the necessary background concepts used in this paper. Section 4 provides an overview of our coupling methodology and shows how one simulates some common scenarios from different domains by incorporating both AI and non-AI agents through PDES. Section 5 goes over the simulation results. Finally, Section 6 concludes the paper.

2 RELATED WORKS

Coupling several models trained on varied topics into multi-agent systems has been an active field of research to improve the overall quality of AI model responses. The interplay of multiple agents require seamless integration of heterogenous agents bringing unique challenges regarding communication, coordination and interaction-rules. Wooldridge discussed various aspects of multi-agent systems including architectures and applications (Wooldridge 2009). Researchers have also extensively studied those challenges, opportunities and limitations of integrating AI and non-AI entities (Alonso et al. 2001) including multi-agent deep-reinforcement learning.

Researchers showed how AI-based robots can collaborate with human agents in industrial settings through continuous learning of human behavior (Baratta et al. 2023). Shen *et al.* (Shen et al. 2024) proposed a model collaboration approach where off-the-shelf models learn to choose among themselves which one will handle what portion of all the input tokens. In another approach named Mixture of Experts (Xue et al. 2024) all experts are trained simultaneously on the same/similar data. These are sub-networks of the mixture that can't be used standalone where the feed-forward layers in a transformer is decomposed into modules called *experts*.

Among prompt-based methods to enhance an LLM's reasoning ability, chain-of-thought (Kojima et al. 2024) activates intermediate reasoning steps for providing a response which promotes a more complex reasoning capability. Other methods that aim at improving inference performance include meta prompting (Zhang, Yuan, and Yao 2024), abstraction (Zheng et al. 2024) and self-verification (Gero et al. 2023). Similarly, tree-search approaches like Monte-Carlo Tree Search also break the task into smaller sub-tasks and improve the performance by sampling simpler, individual, and intermediate reasoning steps (Zhang et al. 2024) but employs a structured way in doing so.

In terms of using LLMs in simulation, Diamantopoulos *et al.* (Diamatopoulos et al. 2024) proposed using LLM in blockchain simulation for dynamic simulation scenario generation and malicious code modeling. Liu *et al.* proposed using LLMs in transportation simulation (Liu, Yang, and Yin 2024) to represent drivers. Giabbanelli (Giabbanelli 2023) and Wu *et al.* (Wu et al. 2023) also proposed smart agent based modeling and simulation methodologies where the LLMs are leveraged in agent-based modeling.

Although these approaches try to enhance multi-model interaction and overall system fidelity, they lack incorporating rule-based non-AI components with domain expertise, thus making the overall system less trustworthy and prone to hallucinations. In our work, we aim to couple off-the-shelf models in a rule-based way to work collaboratively towards a common solution by breaking down the problem into simpler sub-tasks. It also inherently enables deployment of hundreds of simpler agents through MPI support. Decomposing the problem also enables exploring new PDES-based algorithms to solve a particular problem using less powerful AI models that are easier to train and deploy. Thus our approach not only increases overall system fidelity it also aims to address the parallel deployment bottlenecks by shifting that burden to carefully designed, specialized PDES engines.

Coupling AI and non-AI agents is also possible using existing frameworks such as LangChain (Chase 2022). But, chaining multiple AI agents or tasks serially leads to latency and inefficiency. PDES engines inherently manage multiple concurrent processes, enabling simultaneous agent execution by running independent tasks in parallel.

3 BACKGROUND

3.1 Parallel Discrete Event Simulation

Parallel discrete event simulation (PDES) refers to the execution of a single discrete event simulation program on multiple processors (Fujimoto 1990) so that the simulation runs more efficiently in parallel than an equivalent traditional sequential simulation. In a PDES, a simulated system is decomposed into smaller independent subsystems, and it is assumed that changes in system state occur at discrete points of simulated time. In a popular implementation style, each subsystem would then maintain its own event list with timestamps, containing the events that are scheduled to occur within that subsystem in the future.

The occurrence of events (execution of subsystems) further schedules a set of actions which may include performing calculations, updating state variables of the simulation and generating new events to advance the simulation. The simulation runs until all events are processed. In sequential DES, timestamps denote when an event will occur changing the system. The main loop of the simulation engine repeatedly processes the event with smallest timestamp and removes the event from event list. In parallel DES if an event with larger timestamp ($E_{t-large}$) is executed before a smaller one ($E_{t-small}$); ($E_{t-large}$) might change a state variable used by ($E_{t-small}$) thus making a future event to affect a past event, causing a *causality* violation. PDES tries to execute the events in parallel avoiding causality errors (in conservative schemes) or detecting causality errors for later recovery (in optimistic schemes). This parallel execution along with clever problem decomposition enables PDES to run very large-scale simulations on supercomputers (Fujimoto et al. 2003; Perumalla 2007) effortlessly.

3.2 Language Models

Language models are a class of generative AI systems designed to understand, synthesize, and manipulate human language (Chang et al. 2024). They are trained on vast amount of mostly human-generated textual data to learn the statistical relationship between words, phrases, and contexts. These models are capable of predicting and generating new text sequences based on given input text prompts – although they face many challenges such as difficulty in understanding complex language, unseen words, structured applications such as mathematics, logic, reasoning etc.

Early language model such as an *n*-gram (Brown et al. 1992) (e.g. bigram, trigram) were based on simple statistical modeling techniques for predicting the probability of the next word based on preceding words. Later, with the rise of deep learning and GPUs, researchers proposed recurrent neural network (RNN) based language models capable of capturing temporal relationship among the words in a sentence (Mikolov et al. 2010). These models were good in tasks like language recognition and translation but still had problems like vanishing gradient (Bengio, Simard, and Frasconi 1994) and inability to capture very long-range relationships among the words. Researchers adopted long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and gated recurrent units (GRUs) (Cho et al. 2014) to extend RNN based models to address these issues but they were still limited in handling very large sequences of data.

The introduction of the self-attention based transformer architecture (Vaswani et al. 2017) for language models revolutionized this field by enabling training on very large web-scale data in parallel and capturing long-rage relationships in the text. Built on transformer architectures, large language models like BERT (Devlin et al. 2019) and GPT (Alec Radford and Karthik Narasimhan and Tim Salimans and Ilya Sutskever) set new standards for solving a wide range of NLP tasks. These models learn broad language representations and can be fine-tuned for specific tasks with smaller, labeled datasets. Because these models are capable of in-context learning, they can generate coherent and contextually relevant responses making them suitable for interactive applications in diverse scenarios.

4 METHODOLOGY

4.1 Coupling Technique

We use the open-source *Simian* PDES engine designed for large-scale computing environments to couple multiple AI and non-AI agents. We use Simian's Python implementation as it works well with mainstream machine learning libraries such as TensorFlow, PyTorch, llama.cpp, and Transformers. Simian uses *entities* as objects that contain event handling functions and automatically distributes among the various MPI ranks for parallel processing ensuring causality. As shown in Figure 1, each AI and non-AI agent is implemented as an ***entity*** of the Simian engine enabling them to inherently run in parallel. The *entity* class has several methods like *attachService, reqService, createProcess, startProcess* to handle various events and processes.

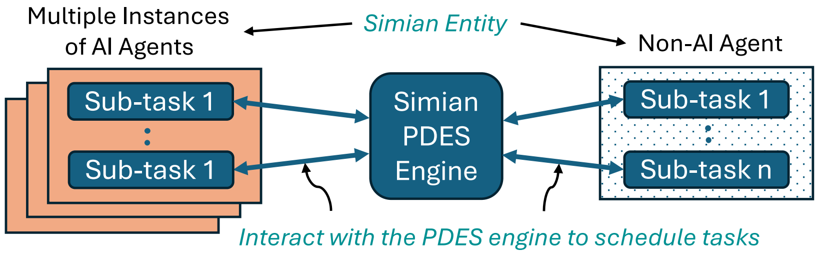


Figure 1: AI / non-AI agents are coupled using the Simian PDES semantics at runtime.

In our work we use the *reqService* method to schedule future events (subtasks) at the corresponding *entity* which effectively inserts new events into that *entity's* event priority list at the desired timestamp.

Figure 2 shows an example of how AI/SLM and non-AI agents are defined as Simian *entities*. We present the gathering simulation in geometry domain where multiple AI-based robotic agents with their own different speeds try to gather. In the figure, a Simian PDES engine is initiated in line 5. Two classes *SLMAgent* and *NonAIAgent* are defined in lines 9 and 28 where both inherit the PDES engine's *Entity* class. In lines 50 to 52, one instance of *NonAIAgent* class and multiple instances of *SLMAgent* class are added to the PDES engine as *entities* using the *addEntity* function while also assigning specific names and numbers to those *entities* for identification. In line 54, the first event of the simulation (*calculate_optimal_gathering_position*) is scheduled at timestamp 0 on the non-AI entity which calculates the *geometric median* of given coordinates. Further events are generated and registered to the corresponding *entities* using the *reqService* method in lines 25, 39, 43, and 48. The simulation starts with *simianEngine.run ()* at line 56 where it starts executing the registered events based on their virtual timestamp. In line 6 and 16 we define the system and user prompts that are passed to language model inference API *create_chat_completion* (line 21) along with other parameters such as context size, chat format etc. where the temperature defines the randomness of the output of language models giving us control over how much human-like behavior we want to incorporate. The language model in the *SLMAgent* is initiated in line 14 where we pass the model path, number of model layers to be offloaded to any available GPU, context size during model inference, and chat format. The response from the language model is parsed and necessary information in extracted which is later passed to further events when registering them using the *reqService* method. Simulation ends when no further events are available in Simian's event queue for processing.

4.2 Problem Formulation

To solve a problem using SLM involving PDES, we (re)formulate it into a multi-step task by breaking the problem into simpler sub-tasks that might have one of a finite number of possible solutions identifiable by an AI-agent. This way, the AI-agents are constrained to a certain finite solution space, thereby boosting chances of success in each sub-task. When we use more sophisticated models like *Qwen* (Yang et al. 2025), for simpler tasks the model is often able to provide a correct response. In that case we don't constrain the model response by providing it with relevant choices to choose from. We use non-AI agents in tandem with SLMs to perform some sub-tasks. The non-AI agents also may be used to verify and correct the responses of an SLM in cases of extreme deviations from acceptable responses. On the other hand, deterministic mathematical sub-tasks that SLMs often fail to provide correct answer to, can be handled by the non-AI entities. Each sub-task is designated as an *event* of the PDES simulation. Each *event* is associated with a Simian *entity*. When an event is requested, it is added to the corresponding entity's event list along with timestamp information. When the simulation runs with MPI enabled, these entities can opportunistically run methods in parallel and execute independent events with the current timestamps in their event lists.

In many scenarios, it is advantageous to considerably constrain an SLM's solution space – this helps reduce its tendency to hallucinate or wander-off into irrelevant contexts. As mentioned, one effective strategy we adopted was to provide the SLM with a set of multiple choices to choose from. Furthermore, the many non-AI agents in the simulation may also cross-check the SLMs' outputs, and then collectively work to further dynamically constrain the various AI agent's solution search-spaces.

```
# Gathering simulation involving Small Language Model
agent_speed = [10, 15, 20, 25, 30]
init_pos = [[0, 108], [0, 462], [335, 571], [543, 285], [335, 0]]
startTime, endTime, minDelay, useMPI, mpiLib,  simProgress  = 0, 100000, 0.0001, True, "/path/to/libmpich", 1
simianEngine = Simian("Gathering_Agents_Dynamic_Gathering_Positions", startTime, endTime, minDelay, ..)
msg_prompt = [{"role":"system", "content":"You are an AI agent moving in a two-dimensional Euclidean space."},
    {"role": "user", "content": ""},]

class SLMAgent(simianEngine.Entity):
    def __init__(self, baseInfo, *args):
        super(SLMAgent, self).__init__(baseInfo)
        self.position = init_pos[self.num]
        self.max_speed = agent_speed[self.num]
        self.slm = Llama(model_path="Qwen2.5-7B-Instruct-1M-f16.gguf", n_gpu_layers=-1, n_ctx=20000)
    def choose_next_step(self, optimal, *args):
        msg_prompt[1]["content"] = f"You are located at position ({self.position[0]}, {self.position[1]}). Your
        goal is to go to position ({optimal[0]}, {optimal[1]}). You can move maximum {self.max_speed} units in
        each step. What should be your position in the next step? Verify that the distance between your new
        position and old position didn't exceed {self.max_speed} units. Strictly follow the following format to
        provide your answer in integer coordinates in the last line of your response. New_Position:(.., ..)."
        response = self.llm.create_chat_completion(messages = msg_prompt, temperature=0.1)
        parse_pos_x_y_from_response()
        reached_optimal = True if math.dist(new_position, optimal) < self.max_speed else False
        #Schedule update_non_slm_copy_of_agent_pos on "Non-AI-Agent" entity.
        self.reqService(simProgress, "update_non_slm_copy_of_agent_pos", (self.num, self.position,
     reached_optimal),
        "Non-AI-Agent", 0)

class NonAIAgent(simianEngine.Entity):
    def __init__(self, baseInfo, *args):
        super(NonAIAgent, self).__init__(baseInfo)
        self.optimal_pos, self.reached_optimal = None, [False] * agentCount
        self.agent_positions = [([None] * 2) for _ in range(agentCount)],
    def update_non_slm_copy_of_agent_pos(self, data, *args): # data = (agent_num, new_pos, reached_optimal?)
        slm_agent_num, self.reached_optimals[slm_agent_num] = data[0], data[2]
        self.agent_positions[slm_agent_num][0] = data[1][0] # agent[num][0], x coordinater
        self.agent_positions[slm_agent_num][1] = data[1][1] # agent[num][1], y coordinate
        # After updating all agent locations (should happen in same virtual time), schedule next event
        if slm_agent_num == 0: # Only one agent needs to request this
            self.reqService(simProgress,"calculate_optimal_gathering_position",self.agent_positions,"Non-AI-
     Agent",0)
    def calculate_optimal_gathering_position(self, positions_lst, *args):
        if all(self.reached_optimals): #Checks if all agents reached optimal position
            for i in range(agentCount):
                self.reqService(simProgress, "dump_stats", None, "SLM-Agent", i)
        else: #Calculates geometric median
            self.reached_optimals
            self.optimal_position = CACL_GEOMETRIC_MEDIAN()
            for i in range(agentCount):
                self.reqService(sim_progress, "choose_next_step", self.optimal_position, "SLM-Agent", i)

simianEngine.addEntity("Non-AI-Agent", NonAIAgent, 0)
for i in range(agentCount):
    simianEngine.addEntity("SLM-Agent", SLMAgent, i)
#Schedule first event at virtual time 0
simianEngine.schedService(0, "calculate_optimal_gathering_position", init_pos[:agentCount], "Non-AI-Agent", 0)

simianEngine.run()
simianEngine.exit()
```

Figure 2: Code snippet from our simulator to solve the gathering problem involving multiple AI agent entities and a non-AI agent entity. Here the non-AI agent mainly perform structured mathematical side-calculations that SLMs are not so good at natively.

4.3 Case Studies

In this subsection we elaborate how we simulate two scenarios to demonstrate our coupling methodology. We simulate three-lane highway traffic movement and robot gathering. In both cases, SLM-based AI agents brings in non-deterministic, human-like characteristics in the simulation while non-AI agents apply constraints to the randomness. In many ways, one may argue that the chosen problems are representative of some of the harder problems for non-AI models, or the current generation of language models to simulate on their own, and hence serve as representative tough-case scenarios to showcase the efficacy of our PDES driven technique. Next we describe our approach in each case in more detail. We also solved few other problems in arithmetic, graph-theory, and combinatorics domains which are beyond the scope of this paper.

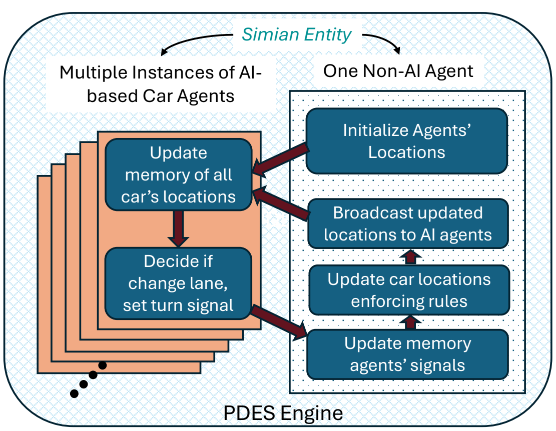
4.3.1 Traffic Simulation

We setup the simulation considering that there are n number of AI-based agents representing n cars traveling in a 3-lane road. Figure 3a shows how we break down the simulation into multiple steps and assign some tasks to the SLM-based agents to get human-like responses. Each car has its own maximum speed and has knowledge about the locations of other cars near it on the road using PDES messages. We define driver characteristics (e.g. aggressive driver or driver who prefers the right lane) as the *system prompt* of the SLM agent. Based on the driver characteristics and information about the cars near to the car in subject, SLM prompt is generated and the AI agent is asked to make lane change choice. It is asked to reason and think step by step to make the decision. The response from the SLM is parsed, and based on the choice it makes, turn signals are activated. All agents then broadcast their turn signal to the non-AI agent through the *Simian* engine. Following steps are repeated to perform the simulation except for the initialization step.

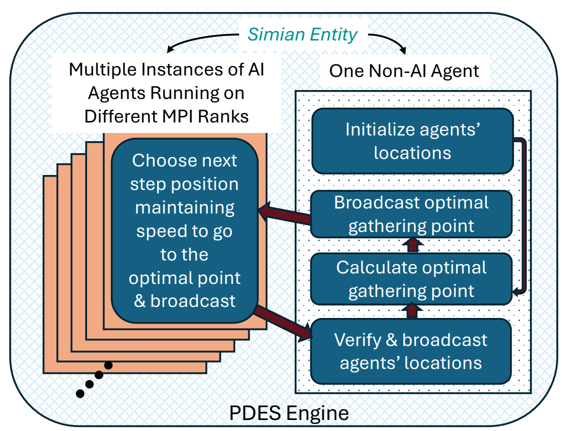
(a) Initial positions, lanes, and speeds of the cars and driver characteristics/goals are set.
(b) AI agent decides on turn signal in order to change the lane based on driver's goal.
(c) Turn signal is broadcasted to the non-AI simulation entity which with some degree of freedom decides whether it is safe to change lane and updates car positions accounting for max speeds.
(d) Updated positions are broadcast to all SLM entities from the non-AI entity using the PDES engine

4.3.2 Agents Gathering Problem

We define the problem as n number of autonomous AI agents on a 2D space (Euclidean plane) trying to gather starting from their initial positions. Each agent has its own maximum speed, thus limiting the number of units it can move in any direction in a given time-step. The optimal gathering position (geometric



(a) Simulation process of traffic simulation involving multiple SLM-based AI and non-AI agents.



(b) Simulation process of multiple agents trying to gather in a 2D space involving SLM.

Figure 3: Shows how the simulations are broken down into multiple subtasks and distributed among the agents to reflect real world simulation dynamics under the umbrella of PDES engine. Each subtask is registered to corresponding Simian entity's event list leaving the burden of inter-process communication and simulation progression to the PDES engine.

Table 1: CPU, GPU and Memory Configuration of a compute node.

CPU	RAM	GPU	GPU Memory
Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz	125GB	Tesla_V100S	32GB

median) is calculated and updated by non-AI agent at each time-step. Step by step, our simulation method is as below:

(a) to start the simulation, the initial positions of the agents are assigned from preset positions.
(b) the non-AI agent is scheduled to calculate the *geometric median* as optimal gathering position.
(c) the SLMs belonging to each agent are asked to choose the new position it should move to in next step following *maximum speed* guideline.
(d) the non-AI agent verifies whether the new position provided by the SLM follows *maximum speed* guideline. If not, it sets the correct coordinates.
(e) the new positions of the agents are passed to another function of the non-AI agent in incremental timestamp to further adjust the optimal gathering position.

All of these subtasks are repeated until the agents gather together. We deploy *n* SLMAgent Simian entities (Figure 2, line 9) as *n* gathering agents. These entities run in parallel by deploying multiple instances of SLMs on different MPI ranks. The steps are illustrated in Figure 3b where multiple instances of SLM-based agents run parallelly to determine new agent position.

4.4 Open Source Generic PDES-AI Framework

Our Simian PDES based AI/non-AI hybrid agents simulator has been designed in a generic manner with a view to enable its use in complex, real-world reasoning problems of broader research interest, more diverse than the relatively simple, yet representative case-studies presented in this paper. We intend to open-source the entire PDES-AI simulator framework code for research purposes.

5 EXPERIMENTS AND RESULTS

5.1 Hardware Setup

We used a cluster of six x86 computer nodes, each with an Nvidia Volta GPU attached to it, over a single network to run the MPI enabled parallel discrete event simulations. Per node hardware requirements limited our scaling studies with efforts underway to fix this in future. Table 1 shows configuration of each node.

To demonstrate the scaling gain from deploying the models in multiple ranks for parallel execution, we use a cluster of six AMD EPYC_7402 @2.8GHz CPUs each with 503GB of memory attached. When we offload the models to the attached GPUs listed in Table 1 to accelerate token generation, GPU memory runs out and thus multiple MPI ranks can't be assigned to single nodes to show strong scaling. Larger memory size of the CPU nodes allows us to assign multiple MPI ranks on a single node avoiding overflow. We take the gathering problem to show simulation scaling with MPI rank increase. We assign 18 CPU threads for the SLM inference within each MPI rank. We used *gathering* simulation to get scaling data.

5.2 SLM Models

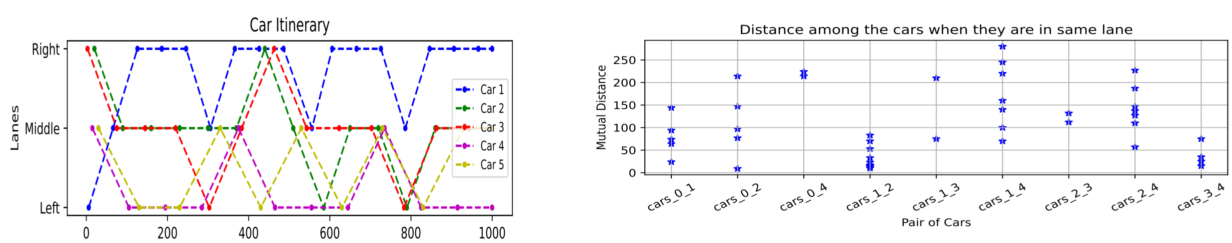
We use *instruct* version of the open-weight model - Qwen-2.5 (Yang et al. 2025) - which have been fine tuned for a question-answer scenario. We use the smaller 14B parameter versions of these models in our experiment since the larger models of 70B+ parameters require a large amount of local video-ram memory and more powerful GPUs to accelerate that are stipulated to be unavailable. We use 8-bit quantized version (16GB size) of this model in GPT-Generated Unified Format (GGUF) since the FP32 version does not fit in our GPUs. Using such quantized, smaller models further enabled running the initial experiments on an Apple M3 MacBook with Metal Shading Language (MSL) acceleration and 64GB total RAM.

To run inference on these models we use *llama-cpp-python* (Betlen) Python library. Llama-cpp-python is an open-source, lightweight, cross-platform supported, community driven C++ accelerated LLM inference library. It supports quantization and is especially optimized to run efficiently on devices with limited resources. Thus it enables running the models locally even on CPUs without GPU acceleration ensuring on-device privacy where user data never leaves the user premises. Line 10 in Figure 2 shows how a model object is initialized using *llama-cpp-python*. The path to the model location and preferred context size is provided to the object-constructor along with the preferred chat format for the model. Line 21 shows an example how a response can be generated from a language model.

5.3 Simulation Results

Figure 4a shows the itinerary of the cars from simulation. All the cars starting from their initial locations, go forward until 1000 distance marker. We observe that depending on speed, different car takes different number of steps to reach destination. In this simulation, we don't strictly enforce collision avoidance mechanism enabling simulation flexibly. As a result it is possible that sometimes based on SLM response, the cars may try to move to same lane without consideration for other cars, potentially causing collisions. However, during our simulations, as seen in Figure 4b we did not observe any collisions. An example response from the SLM-based entity is listed below:

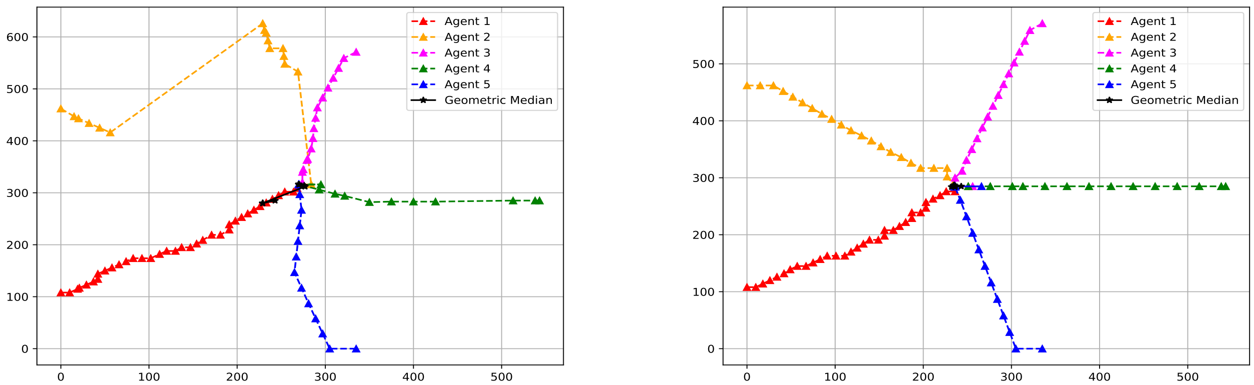
```
System Prompt: You are a car driver travelling in a three-lane road where all cars are
     moving forward. Your goal is to stay in right lane. If there is a slow car ahead
    in the same lane as yours, then try to change lane and overtake that car.
User Prompt: Your are currently at left lane, position 6, driving at a speed of 60
    feet per second. Other cars on the road are: Car 2: Travelling at 70 feet per
    second in right lane at position 20. Car 3: Travelling at 80 feet per second in
    right lane at position 3. Car 4: Travelling at 90 feet per second in middle lane at
     position 15.  Car 5: Travelling at 100 feet per second in middle lane at position
    30. Which of the following option would you choose to meet your goal? Option 1:
    Stay in the same lane. Option 2: Move to the middle lane. Think step by step.
Response:
 Option: 2.
Step-by-step reasoning:
1. The goal is to stay in the right lane.
2. Currently, the driver is in the left lane, driving at 60 feet per second.
3. In the right lane, Car 3 is ahead at position 3 and driving at 80 feet per second,
    which is faster than the driver's speed.
4. In the middle lane, Car 4 is ahead at position 15 and driving at 90 feet per second
    , which is also faster than the driver's speed.
5. To change to the right lane, the driver would need to overtake Car 3, but since Car
     3 is faster, it is not possible to overtake safely.
6. To change to the middle lane, the driver can overtake Car 4 as it is slower than
    the driver's speed (60 feet per second compared to Car 4's 90 feet per second).
```



(a) Itinerary of five cars.

(b) Mutual distance between cars while in the same lane.

Figure 4: Simulation of three lane traffic with five cars, with AI agents proxying for human driver behavior.



(a) SLM response not verified by the Non-AI agent

(b) SLM response verified by the AI agent

Figure 5: Comparing the paths the five SLM-based robots take when the SLMs are is asked to gather given different step size for different agents. The left and right figures respectively show the paths they take when the non-AI agents either *do-not* or *do* verify and correct the SLMs' response.

```
7. After changing to the middle lane, the driver can continue moving towards the right
   lane, where there is no car in front that is slower than the driver.
8. Therefore, the best option to eventually reach the right lane is to first move to
   the middle lane.
```

If we didn't use the SLM agents, it would be difficult and would have required complex deterministic logic to produce a model with human-like behavior for car drivers.

For the *Robot Gathering* problem, Figure 5 shows the path of the agents in various conditions. We choose the starting positions of the agents as the vertices of a pentagon for better visualization. These vertices are listed in Figure 2's line 3. The maximum distance each agent can travel in a step is listed in line 2. In the Figure 5a, without applying constraints and verifying SLM response, the robots tend to deviate from their path often making large jumps exceeding the predefined *maximum distance* it can travel in each step. We observe that overall 32% times the AI agent make such large jump. On the other hand, Figure 5b shows that when these jumps are detected by the non-AI agent, it can ask the SLM to provide correct *new position* or even it can itself correct the error; thus avoiding the jumps. Overall smoothness and quality of the agent trajectories is observed to be better with our coupling strategy.

5.3.1 Scaling

Figure 6 shows that with the increase of MPI ranks, the simulation time decreases while the average number of tokens generated by the language models per minute also increase. The incremental improvements are more when the number of MPI ranks are low, since more agent *entities* need to be assigned on the same MPI rank (or compute node). In the *geometric* simulation scenario which is shown in the plot, there were 6 agents in total: 5 SLM AI agents, and 1 non-AI rule enforcing agent. In all our scenario based experiments, strong scaling in simulation time and tokens/minute has been observed up to as many ranks as there are instantiated agents, since the AI agents in particular are almost always compute/memory-resource bound. One scaling limitation of using PDES is that the simulation becomes more GPU-dependent when the number of requests to the SLM increases significantly while the CPUs sit idle. Thus careful problem decomposition into AI and non-AI components is crucial in maximizing resource utilization to maintain scaling. Still, we show that we can get results similar to LLMs using only SLMs through our strategy.

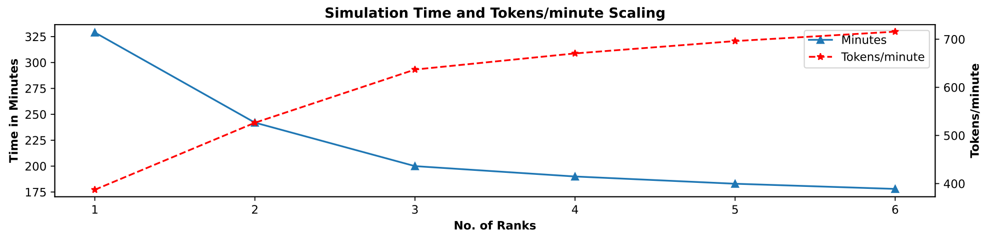


Figure 6: Simulation scaling with MPI ranks

6 CONCLUSION

This paper presents a PDES-based methodology to couple multiple AI and non-AI agents in a rule-based way to work towards a common goal. With the support of MPI, careful problem decomposition, and multiple-choice constraints, the AI and non-AI agents run in parallel on CPU/GPU heterogenous hardware to solve each sub-task of a more complex parent-task. We demonstrated how problems from different domains can be efficiently solved by our structured, PDES strategy involving multiple AI and non-AI agents. Moreover, the intermediate results are cross-checked using non-AI agents (potentially involving even more complex sub-simulations), promoting verifiability of AI model responses and thereby ensuring trustworthiness of AI agent simulations. By coupling AI and non-AI agents using PDES ensures incorporation of human-like agent components especially in social simulation aspects. We will be open-sourcing our code for the research community to enable further progress in these directions.

ACKNOWLEDGMENTS

Atanu Barai was supported by the DoE ASC program. Stephan Eidenbenz and Nandakishore Santhi were supported by funding from the DoE Office of Science, ASCR program. This work is approved for unlimited release with LA-UR-25-23562.

REFERENCES

Alonso, E., M. D'Inverno, D. Kudenko, M. Luck, and J. Noble. 2001. "Learning in Multi-agent Systems". *The Knowledge Engineering Review* 16(3):277–284.

Baratta, A., A. Cimino, M. G. Gnoni, and F. Longo. 2023. "Human Robot Collaboration in Industry 4.0: a Literature Review". *Procedia Computer Science* 217:1887–1895.

Bengio, Y., P. Simard, and P. Frasconi. 1994. "Learning Long-term Dependencies with Gradient Descent is Difficult". *IEEE Transactions on Neural Networks* 5(2):157–166.

Andrei Betlen. "Python Bindings for Llama.cpp". https://github.com/abetlen/llama-cpp-python, accessed 02.04.2025.

Bommasani, R., D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, *et al*. 2021. "On the Opportunities and Risks of Foundation Models". *arXiv preprint arXiv: 2108.07258*.

Brown, P. F., P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. 1992. "Class-based n-gram Models of Natural Language". *Computational Linguistics* 18(4):467–479.

Chang, Y., X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, *et al*. 2024. "A Survey on Evaluation of Large Language Models". *ACM Transactions on Intelligent Systems and Technology* 15(3):1–45.

Harrison Chase 2022, October. "LangChain". https://github.com/langchain-ai/langchain, accessed 06.05.2025.

Cho, K., B. van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111. Doha, Qatar.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. 2019. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *arXiv preprint arXiv: 1810.04805*.

Diamatopoulos, G., G. Theodoropoulos, N. Tziritas, and R. Bahsoon. 2024. "Towards LLM Augmented Discrete Event Simulation of Blockchain Systems". In *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '24, 75–76. Atlanta, GA, USA.

Dubey, A., A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, *et al*. 2024. "The Llama 3 Herd of Models". *arXiv preprint arXiv: 2407.21783*.

Fujimoto, R. M. 1990, October. "Parallel Discrete Event Simulation". *Communications of the ACM* 33(10):30–53.

Fujimoto, R. M., K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. 2003. "Large-scale Network Simulation: How Big? How Fast?". In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.*, 116–123. Orlando, FL, USA.

Gero, Z., C. Singh, H. Cheng, T. Naumann, M. Galley, J. Gao *et al*. 2023. "Self-Verification Improves Few-Shot Clinical Information Extraction". *arXiv preprint arXiv: 2306.00024*.

Giabbanelli, P. J. 2023. "GPT-Based Models Meet Simulation: How to Efficiently use Large-Scale Pre-Trained Language Models Across Simulation Tasks". In *2023 Winter Simulation Conference (WSC)*, 2920–2931 https://doi.org/10.1109/WSC60868.2023.10408017.

Hochreiter, S., and J. Schmidhuber. 1997. "Long Short-Term Memory". *Neural Computation* 9(8):1735–1780.

Kojima, T., S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. 2024. "Large Language Models are Zero-shot Reasoners". In *Proceedings of the 36th International Conference on Neural Information Processing Systems*. New Orleans, LA, USA.

Liu, T., J. Yang, and Y. Yin. 2024. "Toward LLM-Agent-Based Modeling of Transportation Systems: A Conceptual Framework". *arXiv preprint arXiv: 2412.06681*.

Mikolov, T., M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. 2010. "Recurrent Neural Network Based Language Model". In *Interspeech*, Volume 2, 1045–1048.

OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, *et al*. 2024. "GPT-4 Technical Report". *arXiv preprint arXiv: 2303.08774*.

Perumalla, K. S. 2007. "Scaling Time Warp-based Discrete Event Execution to 104 Processors on a Blue Gene Supercomputer". In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, 69–76. Ischia, Italy.

Alec Radford and Karthik Narasimhan and Tim Salimans and Ilya Sutskever. "Improving Language Understanding by Generative Pre-training". https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, accessed 02.04.2025.

Santhi, N., S. Eidenbenz, and J. Liu. 2015. "The Simian Concept: Parallel Discrete Event Simulation with Interpreted Languages and Just-in-time Compilation". In *2015 Winter Simulation Conference (WSC)*, 3013–3024 https://doi.org/10.1109/WSC.2015.7408405. Source code available at https://simian.lanl.gov.

Shen, S. Z., H. Lang, B. Wang, Y. Kim, and D. Sontag. 2024. "Learning to Decode Collaboratively with Multiple Language Models". *arXiv preprint arXiv: 2403.03870*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, *et al*. 2017. "Attention is All You Need". In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Volume 30. Long Beach, CA, USA.

Wooldridge, M. 2009. *An Introduction to Multiagent Systems*. 2nd ed. John Wiley & Sons.

Wu, Z., R. Peng, X. Han, S. Zheng, Y. Zhang, and C. Xiao. 2023. "Smart Agent-Based Modeling: On the Use of Large Language Models in Computer Simulations". *arXiv preprint arXiv: 2311.06330*.

Xue, F., Z. Zheng, Y. Fu, J. Ni, Z. Zheng, W. Zhou *et al*. 2024. "OpenMoE: An Early Effort on Open Mixture-of-Experts Language Models". *arXiv preprint arXiv: 2402.01739*.

Yang, A., B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, *et al*. 2025. "Qwen2.5 Technical Report". *arXiv preprint arXiv: 2412.15115*.

Zhang, D., X. Huang, D. Zhou, Y. Li, and W. Ouyang. 2024. "Accessing GPT-4 level Mathematical Olympiad Solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B". *arXiv preprint arXiv: 2406.07394*.

Zhang, Y., Y. Yuan, and A. C.-C. Yao. 2024. "Meta Prompting for AI Systems". *arXiv preprint arXiv: 2311.11482*.

Zheng, H. S., S. Mishra, X. Chen, H.-T. Cheng, E. H. Chi, Q. V. Le *et al*. 2024. "Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models". In *The Twelfth International Conference on Learning Representations*. Vienna, Austria.

AUTHOR BIOGRAPHIES

ATANU BARAI is a Postdoctoral Research Fellow with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory. He obtained a PhD from New Mexico State University in Computer Engineering. His research interest include scalable modeling and simulation, computer architecture and performance analysis, and artificial intelligence. His email address is abarai@lanl.gov.

STEPHAN EIDENBENZ is a Computer Research Scientist with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory. He obtained a PhD from the Swiss Federal Institute of Technology, Zurich (ETHZ) in Computer Science. His research interests include cyber security, computational codesign, communication networks, scalable modeling and simulation, quantum computing, and theoretical computer science. His email address is eidenben@lanl.gov

NANDAKISHORE SANTHI is a Computer Research Scientist with the Information Sciences Group (CCS-3) at Los Alamos National Laboratory. He holds a PhD in Electrical and Computer Engineering from the University of California San Diego. His areas of research interests include parallel discrete event simulation, performance modeling of HPC systems, applied mathematics, communication systems, quantum computing, and computer architectures. His email address is nsanthi@lanl.gov.