EE 451
Course Project Proposal
Due: October 20th, 2024
Kae Sawada
SID: 6598172388

Parallelization of Scientific Realtime Generative AI Response Verification System for Continuous Quality Assurance in Safety-Critical Software

Introduction

Integrating Generative AI into safety-critical systems requires continuous, rigorous quality assurance to mitigate risks of decision-making errors. Manual verification methods are expensive and time-consuming, making automation essential. This project proposes an autonomous verification system utilizing multiple Large Language Models (LLMs) as a parallelized panel of verifiers.

This study examined the feasibility of improving verification latency while maintaining decision accuracy/correctness by applying task-level and data-level parallelism. Efficiency was enhanced through optimized task mapping and minimizing lock contention to reduce inter-process communication overhead. Leveraging Task Dependency Graph analysis and Amdahl's Law, the study conducted theoretical and experimental evaluations to quantify performance improvements, focusing on metrics such as speedup and reduced latency.

The code developed for this project can be viewed at https://github.com/ksawada0/chronoarbitor.

System Architecture Diagram

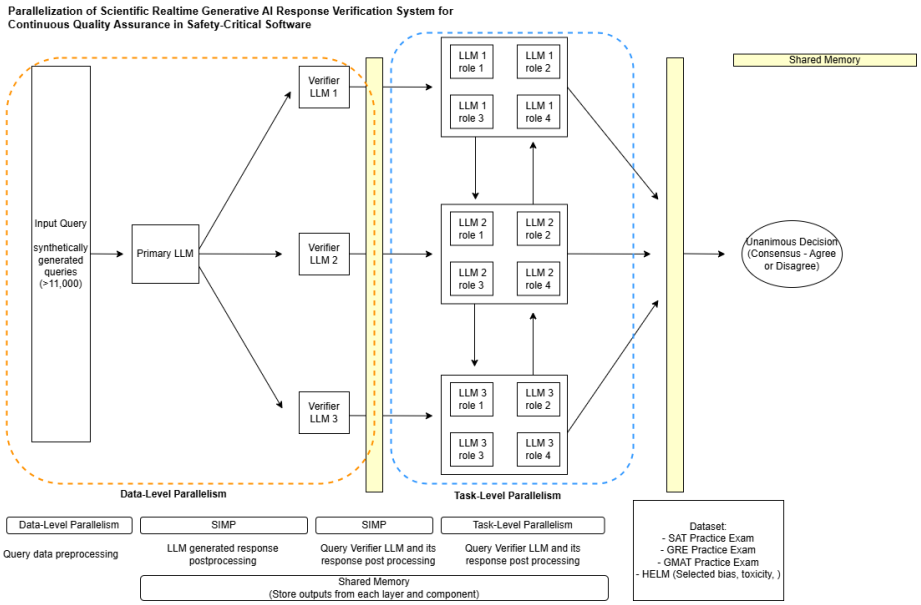

Figure 1 2The diagram shown summarizes the system architecture of the verification process simulation pipelines. The system consists of 3 stage simulation, input layer that comprises of data ingestion and the output of the LLM under test. The second layer consumes the identical prompt that primary LLM's consumed, and independently produces its. In the third layer, the three LLM will play various roles to mimic a panel of experts and non-experts. The final output is a Pass/Partial Pass/Failure, determined by the number of agreements

Problem Statement

The proposed verification system is required to continuously and efficiently assess the alignment of Generative AI outputs. Manual verification is costly and impractical, making fast, automated solutions essential for ensuring safety and reliability.

With increasing interest in multi-agent and multi-LLM architectures, this project focuses on identifying efficient query mechanisms and applying parallelization techniques to improve performance and scalability.

A preliminary prototype evaluation revealed the following observations and solutions:

- **Observation**: Tasks within the same layer are independent and can be executed in parallel
    - Solution: Parallelize tasks within a layer using multiple workers
- **Observation**: Some tasks, like phi-Engineer, take significantly longer than others (see appendix for a sample raw logging of some response time assessment)
    - Solution: Start these high-latency tasks early to prevent them from becoming bottlenecks (task-level scheduling to give phi-Engineer higher priority)
- **Observation**: Models like *gemma* and *mistral* are faster overall.

    - Solution: Group tasks for gemma and mistral into a batch and assign them to workers in parallel. This allows for slower tasks like phi-Engineer do not idle the system.
- **Observation**: Latency variability across runs suggests external factors (e.g., server load, network latency).
    - Solution: Dynamically assign tasks to workers based on their current load or estimated task duration.

Hypothesis

By parallelizing independent tasks within each layer (each level of the TDG), by dynamically prioritizing/scheduling high-latency tasks* by querying the language model-role earlier, we expect to reduce latency by 20–30%.

* e.g. Querying slower language models like phi-Engineer earlier

Parallelization Strategy Used

- Application of MapReduce concept (See
- Construction of Task Dependency Graph and its analysis yielded:
    - Revealed that the original design does not support straightforward SIMD parallelization (see figure 1)
    - Redesigned the program to improve parallel performance while maintaining a "blind experiments" approach.
    - Measured an average LLM response time of 12.33 seconds across 737 calls to the Ollama server.
- Given the number of independent tasks in layer 2 and layer 3 gives an opportunity for task parallelization
- Implemented and compared Shared Memory and Message Passing parallelization techniques.

Experimental Setup

- Many private and government organizations have strict regulation on private and/or export-controlled information that cannot be publicly shared. In this experiment, we will target such scenario that LLM are locally deployed, and its privacy is maintained.

- Language Models Used:
    - Llama 3
    - Phi
    - Mistral
    - Gemma
    - Quen2
    - Mistral-Nemo
    - QWQ
- Language Mode Server: Ollama
    - Ollama is capable of handling concurrent queries - set the following environment variables
        - OLLAMA_NUM_PARALLEL
        - OLLAMA_MAX_LOADED_MODELS
- Testbed System Specification – please see Appendix A for more details
    - # of CPU utilized: 1 through 4
    - # of thread: 1 through 8
    - # of GPU utilized: 1

Summary of the Improvements from the Preliminary Results Presented

Based on instructor and peer feedback, the program has been significantly refined, with the following modifications:

- Project Rescoping:
    - Removed unnecessary efforts, such as hypercube communication, to focus on essentials.
- Parallelization Techniques:
    - Separated into two distinct methods for clearer analysis – Shared Memory Parallelization and Message Passing
- MPI Implementation:
    - Eliminated hypercube network communication.
- Updated Goals:
    - Revised hypothesis, problem statements, and objectives to target measurable speedup via dependency graph construction (Figure 3) and LLM latency evaluation.
- Thread Contention:
    - Reduced lock-controlled variables to improved performance.
- MapReduce Optimization:
    - Assigned a unique *prompt_id* to each task for full parallel processing.

- Used thread-local dictionaries for intermediate results, minimizing locking contention.
- Aggregated results via thread-safe reduction.

Result and Analysis

To highlight, optimizing the MapReduce concept implementation and removing lock control from a global variable successfully reduced latency for high task loads from Shared Memory method.

Reducing the frequency of global variable updates was effective in MPI method.

Task Dependency Graph Analysis

Figure 3 shows the task dependency graph constructed for this study.

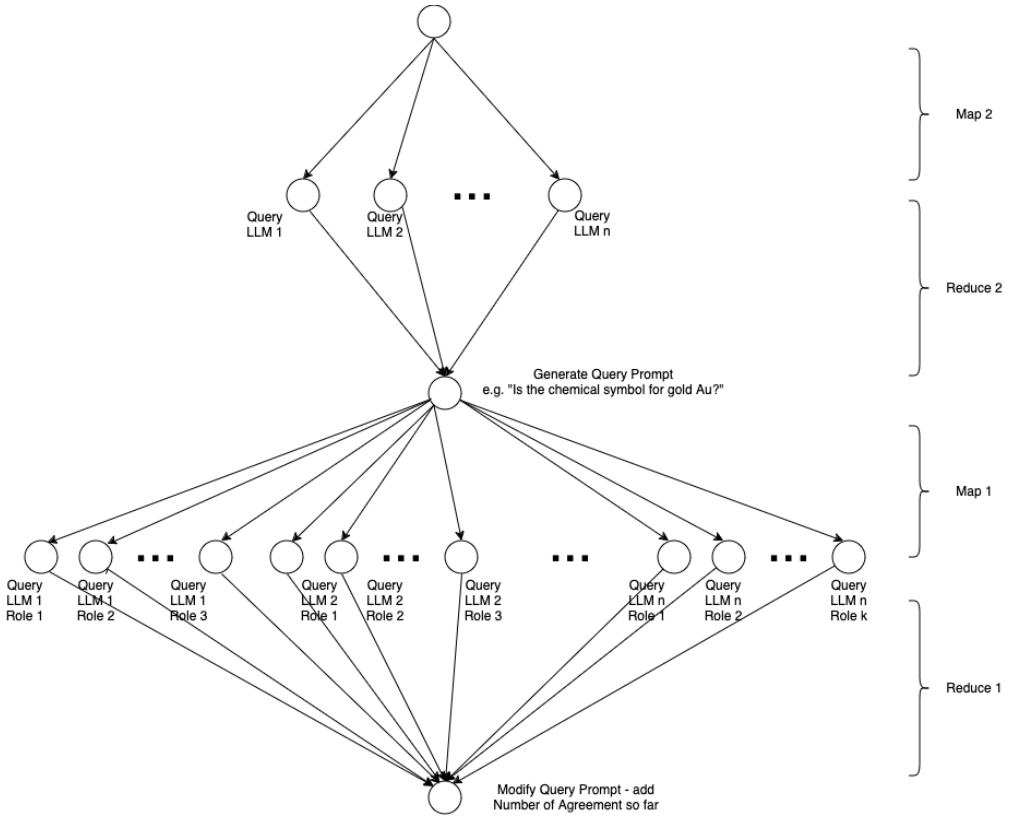

Figure 3 Task Dependency Analysis

Average time it takes to construct prompts: 1.5×10^{-7} seconds (essentially negligible in comparison to query time).

Query response time per LLM: Takes **12 seconds** (this dominates the time).

Max degree of concurrency: (# of LLMs) × (# of roles) (# of LLMs)×(# of roles).

Critical path length: **4 units** (limiting factor).

Theoretical Achievable Speedup:

With 3 LLMs and 5 roles,

$$\text{Max Speedup} = \frac{TotalWork}{CriticalPathLength + \frac{P}{\text{Concurrency}}}$$

$$= \frac{16}{4+\frac{12}{15}} \approx 3.33$$

However, this does not take into account the limitation imposed by the LLM server and lock contentions associated with necessary shared variables.

Practical Achievable Speedup: Given the LLM response latencies and lock contentions, 15% to 25% is more likely

Speedup/Acceleration Achieved

Serial vs. Shared Memory (4 threads): **14%**

MPI (4 processes): **18%**

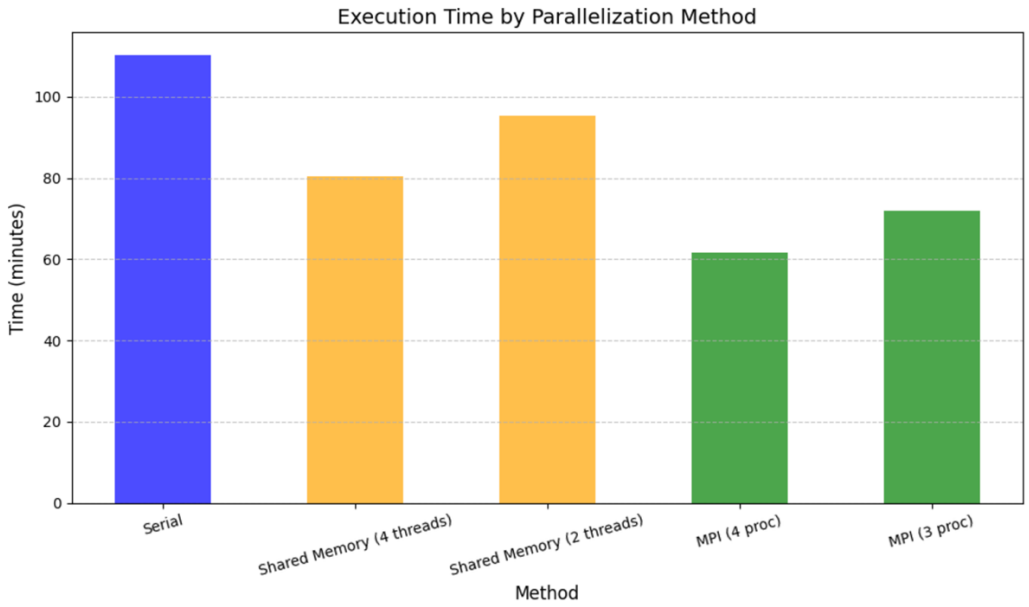

Figure 4 Speedup/Acceleration

Number of Processes Comparison

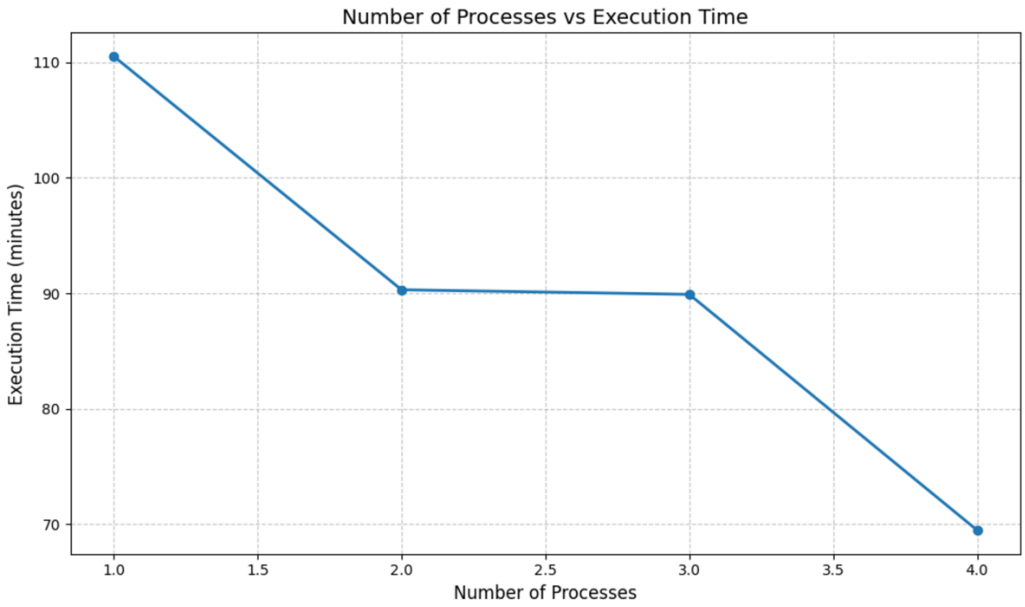

Figure 5 Number of Cores Comparison

Number of Threads Comparison

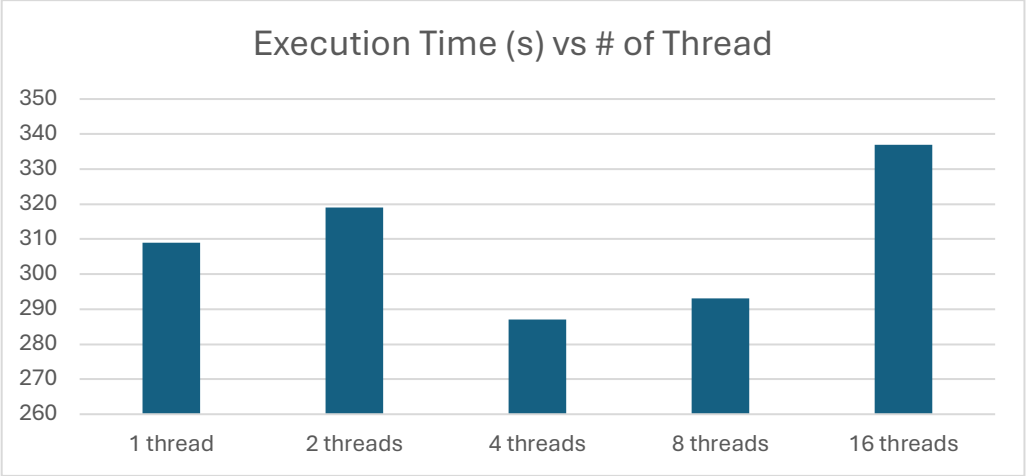

Figure 6 Number of Threads Comparison

Conclusion

- For this project, MPI parallelization method outperformed Shared Memory method.

- In shared memory programming, excessive use of global variables requiring locks significantly hinders performance gains. Shared counters, such as *global_agree_counter*, limit speedup due to thread contention.
- The server's response time is unpredictable, leading to non-parallelizable components dominating the program with the current design and language model server (Ollama).
- To reduce contention, moving shared variable writes outside thread functions and aggregating results externally has improved speed.
- This project highlighted architectural design flaws and provided valuable insights for future program improvements.
- Redesigning the third-layer verification architecture is recommended to address its dependency on two global variables for reducing lock contention.

GitHub Repository

https://github.com/ksawada0/chronoarbitor

Reference

Prasanna, Victor 2024. "EE/CSCI 451: Parallel and Distributed Computation Lecture #13." Department of Computer Science, University of Southern California, October 15. https://piazza.com/class_profile/get_resource/lzucri269tu70c/m2f67i9s3zs2ec.

Prasanna, Victor 2024. "EE/CSCI 451: Parallel and Distributed Computation Lecture #14." Department of Computer Science, University of Southern California, October 17. https://piazza.com/class_profile/get_resource/lzucri269tu70c/m2f67i9s3zs2ec.

Wickramasinghe, Sachini 2024. "EE/CSCI 451: Introduction to Parallel and Distributed Computation Discussion #6." University of Southern California, October 8. https://piazza.com/class_profile/get_resource/lzucri269tu70c/m2f67i9s3zs2ec.

Liang, Percy, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, et al. “Holistic Evaluation of Language Models.” arXiv, October 1, 2023. https://doi.org/10.48550/arXiv.2211.09110.

Wu, Qingyun, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, et al. “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.” arXiv, October 3, 2023. http://arxiv.org/abs/2308.08155.

David Culler, Richard IQ, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken

“LogP: Towards a Realistic Model of Parallel Computation,” n.d., Computer Science Division, University of California Berkeley

Barbic, Jernej. “Multi-Core Architectures,” 15-213, 2007, May 3rd https://www.cs.cmu.edu/~fp/courses/15213-s07/lectures/27-multicore.pdf

“PJM - System Operations.” Accessed October 9, 2024. https://www.pjm.com/markets-and-operations/ops-analysis.

Barbic, Jernej. “Multi-Core Architectures,” n.d.

“Stanford Large Network Dataset Collection.” Accessed October 9, 2024. http://snap.stanford.edu/data/index.html.

Weng, Yixuan, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. “Large Language Models Are Better Reasoners with Self-Verification.” arXiv, October 19, 2023. http://arxiv.org/abs/2212.09561.

Shen, Shannon Zejiang, Hunter Lang, Bailin Wang, Yoon Kim, and David Sontag. “Learning to Decode Collaboratively with Multiple Language Models.” arXiv, August 27, 2024. http://arxiv.org/abs/2403.03870.

Tihanyi, Norbert, Mohamed Amine Ferrag, Ridhi Jain, and Merouane Debbah. “CyberMetric: A Benchmark Dataset for Evaluating Large Language Models Knowledge in Cybersecurity.” arXiv, February 12, 2024. http://arxiv.org/abs/2402.07688.

Huang, Jun, Jiawei Zhang, Qi Wang, Weihong Han, and Yanchun Zhang. “Exploring Advanced Methodologies in Security Evaluation for Large Language Models,” n.d.

Smith, Eric Michael, Melissa Hall, Melanie Kambadur, Eleonora Presani, and Adina Williams. "'I'm Sorry to Hear That': Finding New Biases in Language Models with a Holistic Descriptor Dataset." arXiv.org, May 18, 2022. https://arxiv.org/abs/2205.09209v2.

Solat, Siamak, and Farid Naït-Abdesselam. "Parallel Committees: High-Performance, Scalable, Secure and Fault-Tolerant Data Replication Using a Novel Sharding Technique." In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, 546–53. Kuwait, Kuwait: IEEE, 2023. https://doi.org/10.1109/BCCA58897.2023.10338937.

Zheng, Xi, Aloysius K. Mok, Ruzica Piskac, Yong Jae Lee, Bhaskar Krishnamachari, Dakai Zhu, Oleg Sokolsky, and Insup Lee. "Testing Learning-Enabled Cyber-Physical Systems with Large-Language Models: A Formal Approach." arXiv, January 15, 2024. https://doi.org/10.48550/arXiv.2311.07377.

"Universal and Transferable Attacks on Aligned Language Models." Accessed April 10, 2024. http://llm-attacks.org/.

UpTrain AI. "Decoding Perplexity and Its Significance in LLMs," December 4, 2023. https://blog.uptrain.ai/decoding-perplexity-and-its-significance-in-llms/.

Plimpton, Steve. "Fast Parallel Algorithms for Short-Range Molecular Dynamics." *Journal of Computational Physics* 117, no. 1 (March 1, 1995): 1–19. https://doi.org/10.1006/jcph.1995.1039.

Appendix

Improvement 1: Handling Lock Controlled Variables

Original implementation

```
def worker_task(task):
    """Thread worker to process a single task."""
    global global_agree_counter, global_decision_count

    model, prompt, layer, role = task
    start_time = time.time()

    # Dynamically generate the prompt for Layer 3
    if layer == 3:
        prompt = ollama_helper.generate_prompt(prompt, layer=3, role=role, agree_count=g

    response = ollama_helper.query_ollama(model, prompt)
    if response:
        result = ollama_helper.process_response(response)
        end_time = time.time()
        if IS_MEASURE_QUERY_RESPONSE:
            print(f"Task: {model}-{role or 'Generalist'} | Layer: {layer} | Time: {end_t

        with decision_count_lock:
            global_decision_count += 1
        if result.decision == "TRUE":
            with agree_counter_lock:
                global_agree_counter += 1
```

Figure 7 Improvements 1 a

Improved Program

Each thread returns its local computational result:

```
def worker_task(task):
    """Thread worker to process a single task."""
    global global_agree_counter, global_decision_count

    model, prompt, layer, role = task
    start_time = time.time()

    # Dynamically generate the prompt for Layer 3
    if layer == 3:
        prompt = ollama_helper.generate_prompt(prompt, layer=3, role=role, agree_count=0

    try:
        response = ollama_helper.query_ollama(model, prompt)
        if response:
            result = ollama_helper.process_response(response)

            end_time = time.time()
            if IS_MEASURE_QUERY_RESPONSE_TIME:
                print(f"Task: {model}-{role or 'Generalist'} | Layer: {layer} | Time: {e

            if IS_LOG_RESPONSE:
                log.info(f"Decision: {result.decision}")
                log.info(f"Justification: {result.justification}")

            return 1 if result.decision == 'TRUE' else 0
    except Exception as e:
        print(f"Erorr processing  task {task}: {e}")
```

Figure 8 Improvement 1 b

The main function aggregates (reduces) the result

```python
    # Use ThreadPoolExecutor for parallelism
    with ThreadPoolExecutor(max_workers=N_WORKERS) as executor:
        results = executor.map(worker_task, tasks)
        with agree_counter_lock:
            global_agree_counter = sum(results)
```

Figure 9 Improvement 1 c code snippet

Improvement 2: Tagging to Flatten the Tasks

All tasks can be executed by any idle processes regardless of which generative AI response it's verifying

```python
# Create tasks
tasks = []
for prompt_id, prompt in enumerate(prompts):
    # Layer 2: Query each model as Generalist
    for model in models:
        tasks.append((prompt_id, model, ollama_helper.generate_prompt(prompt, layer=2), 2, DEFAULT_ROLE))

    # Layer 3: Query each model with role-specific prompts
    for model in models:
        for role in roles:
            tasks.append((prompt_id, model, prompt, 3, role))
```

Figure 10 Improvement 2 code snippet

```
Task: gemma-Philosophy Professor | Layer: 3 | Time: 0.86 seconds
Task: llama3-Generalist | Layer: 2 | Time: 10.06 seconds
Task: phi-Generalist | Layer: 2 | Time: 6.96 seconds
Task: gemma-Generalist | Layer: 2 | Time: 11.47 seconds
Task: llama3-Engineer | Layer: 3 | Time: 10.80 seconds
Task: llama3-Philosophy Professor | Layer: 3 | Time: 2.20 seconds
Task: phi-Engineer | Layer: 3 | Time: 13.08 seconds
Task: phi-Philosophy Professor | Layer: 3 | Time: 4.35 seconds
Task: gemma-Engineer | Layer: 3 | Time: 11.27 seconds
Task: gemma-Philosophy Professor | Layer: 3 | Time: 1.09 seconds
Task: llama3-Generalist | Layer: 2 | Time: 10.65 seconds
Task: phi-Generalist | Layer: 2 | Time: 9.87 seconds
Task: gemma-Generalist | Layer: 2 | Time: 12.14 seconds
Task: llama3-Engineer | Layer: 3 | Time: 10.25 seconds
Task: llama3-Philosophy Professor | Layer: 3 | Time: 1.43 seconds
Task: phi-Engineer | Layer: 3 | Time: 6.34 seconds
Task: phi-Philosophy Professor | Layer: 3 | Time: 1.25 seconds
Task: gemma-Engineer | Layer: 3 | Time: 12.14 seconds
Task: gemma-Philosophy Professor | Layer: 3 | Time: 0.99 seconds
```

Figure 11 Language Model-Role Performance Measurements