SLAM: Small Language Agentic Machine

Aravind SS[a,*], Kokane Manoj Bhausaheb[a,], Parth Bhatia[a,***], Sachin Bansal[a,****], Srividhya L[a,*****] and Swapnil Trivedi[a,******]**

[a]Indian Institute of Science, Bangalore, India

Abstract. *SLAM* (Small Language Agentic Machine) is a lightweight, local-first assistant that is designed to perform practical tasks through modular tool execution and coordinated language-model reasoning. SLAM adopts a two-stage architecture: an encoder-decoder T5 model first rewrites user inputs into concise, instruction-like prompts; a decoder-only SLM then interprets these prompts, decides whether a tool call is needed, and generates the final reply. A lean in-process controller validates tool JSON, executes a sandboxed registry (like calculator, OCR, summariser, formatter, . . .), and injects each result back into the generator. The entire stack is self-contained, requires no cloud APIs, and can be adapted to new domains via adapter-based fine-tuning of the rewriting model. By decoupling prompt interpretation, dialogue generation, and task execution, SLAM achieves agent-like behaviour while remaining efficient, extensible, and deployable on lightweight hardware.

1 Introduction

Large language models deliver impressive tool-augmented reasoning, but their large size, high latency, and privacy concerns make them challenging to deploy. Ironically, they often falter at basic tasks like arithmetic or factual retrieval, where smaller, specialized models perform exceptionally well. In this paper, we demonstrate that small language models can leverage external tools through straightforward APIs to combine the strengths of both approaches. SLAM explores a related question: *To what extent can we rely on compact, efficient models when integrated with a disciplined controller and well-defined tools?* We push a pair of sub-4B language models if we add a disciplined controller and a few well-scoped tools?

Why T5? A fine-tuned T5 layer standardises informal queries, strips chit-chat, and yields machine-friendly instructions—significantly reducing the chances of hallucination and malformed JSON by the user's query.

Our key contributions are:

- A fully local pipeline that pairs a T5 rewriter with a Phi-4-Mini generator under a minimal JSON controller.
- A tool registry with strict schemas and a safety sandbox, covering calculator, Python-shell, OCR and more.
- Detailed recipes for synthetic data generation, adapter-based T5 training, and prompt-engineering heuristics.

* Equal contribution.
** Equal contribution.
*** Equal contribution.
**** Equal contribution.
***** Equal contribution.
****** Equal contribution.

2 System Architecture

Figure 1 sketches the end-to-end flow. A lightweight UI feeds the T5 rewriter; the *PromptHandler* wraps the rewritten text with based on a tool schema; the SLM streams tokens via `llama.cpp`; the controller spots tool schema, validates it, executes the corresponding tool, and reinjects the result back into the SLM to resume the natural flow of conversation.

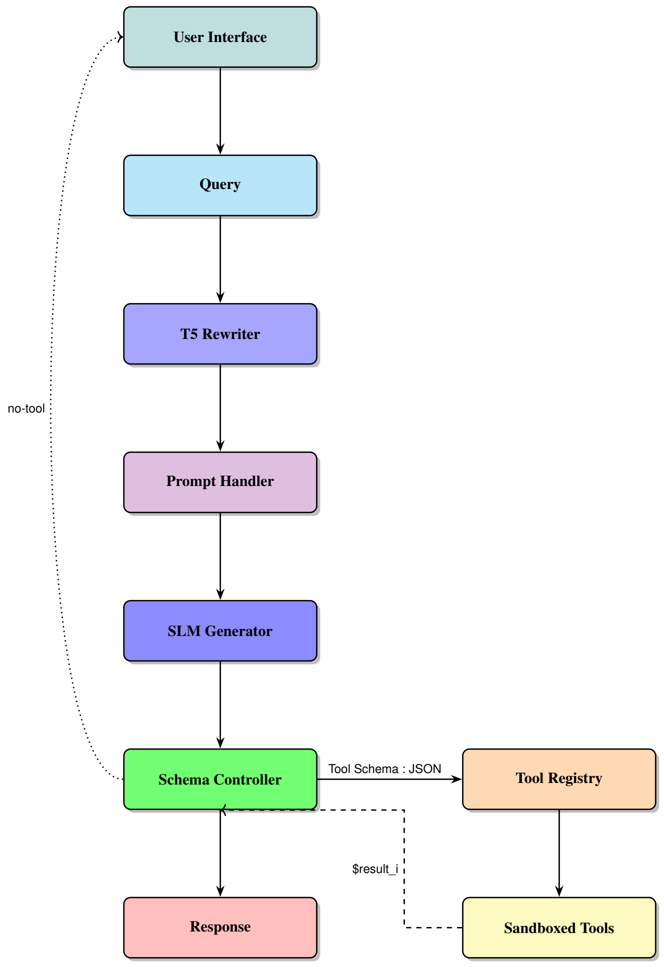


Figure 1. SLAM pipeline overview.

3 Synthetic Data Generation

We realized that out of the box T5 would not be able to perform as expected. To address this, we created a synthetic dataset that covered a wide range of tool and its use cases.

1. Drafted $\sim$60,000 query-response pairs covering tools like Calculator, OCR and JSON formatter.
2. The dataset was created using a hybrid approach that combined advanced prompt engineering techniques for large language models, including **GPT-4o, DeepSeek R1, & Claude Sonnet 4**, with synthetic data generation powered by paraphrasing models such as **Pegasus 568M**. In formats like – Text-Text, Text-Numbers & Text-XML.
3. Lastly, we performed a train-test-validation split of 80% for training, 10% for validation, and 10% for testing.

4 Adapter-based T5 Fine Tuning

We attach LoRA adapters to T5-Base and train five epochs with AdamW (η=5×10^{-5}, batch 8). Figure 2 plots the actual training and validation loss.

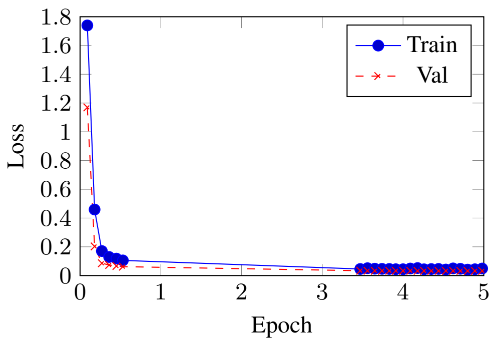


Figure 2. Loss curves for adapter-based T5 training.

The gradient analysis shows that the LoRA adapter weights have extremely small mean and standard deviation values, often close to zero. This suggests that the updates to the adapter layers during training are minimal, which may be expected early in training or due to conservative hyperparameters like low learning rate or LoRA rank. While not inherently bad, consistently small gradients across epochs could mean ineffective learning.

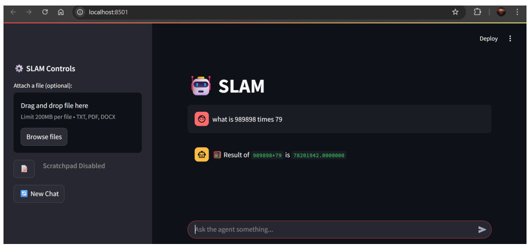
5 Prompt-Engineering Heuristics

The SLM occasionally ignored JSON fences and went on a runaway generation loop, producing an infinite stream of tokens. To mitigate this, we implemented a few heuristics:

- **Tool Schema** : We defined a strict JSON schema for each tool, ensuring that the SLM generates well-formed JSON objects. This schema includes required fields, types, and constraints for each tool's input and output. For example, the calculator tool requires an ‘expression‘ field of type string, while the weather tool requires a ‘city‘ field of type string.
- **Tool Blocks** : We encapsulated each tool call within a JSON block, ensuring that the SLM generates a complete and valid JSON object for each tool call. This prevents the SLM from generating partial or malformed JSON objects, which could lead to runaway generations.
- **Tool Fences** : We used special tokens to mark the beginning and end of each tool block. This helps the SLM to recognize the boundaries of each tool call and prevents it from generating tokens outside the intended scope.
- **Prompt Engineering** : We carefully designed the prompts to guide the SLM's generation process. This includes providing clear instructions on how to use each tool, specifying the expected input and output formats, and using examples to illustrate the intended use of each tool. For example, we provided a prompt like "Use the calculator tool to evaluate the expression: 2 + 2" to guide the SLM in generating the correct JSON object for the calculator tool.

6 Evaluation

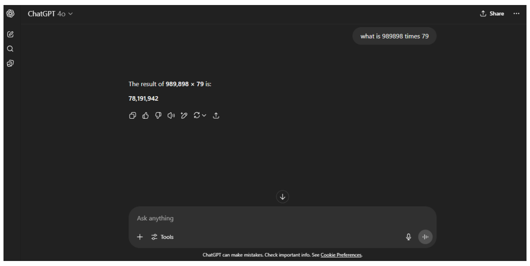
We manually evaluated SLAM on a diverse set of queries, including arithmetic, weather retrieval, and OCR tasks. The results showed that SLAM could handle most queries effectively, with a few exceptions where the T5 rewriter struggled to generate the correct simplified queries. One such example of SLAM performing better than GPT is mentioned below:



what is 989898 times 79

Result of `989898*79` is `78201942.0000000`

Figure 3. Comparison of SLAM with GPT-based tools.



ChatGPT 4o
Share
Ask anything
Tools
ChatGPT can make mistakes. Check important info. See Cookie Preferences.

what is 989898 times 79

The result of **989,898 × 79** is:

78,191,942

Figure 4. GPT-4o struggling with basic arithmetic tasks.

7 Tool Registry

Table 1. Core tools in SLAM.

Tool	Signature	Safety Guardrail
`calculator`	expr $\longrightarrow$ float	AST walk; no names
`python_shell`	code $\longrightarrow$ stdout	Empty globals; 3 s timeout
`get_weather`	city $\longrightarrow$ weather	NA
`ocr`	img $\longrightarrow$ text	Offline Tesseract
`json formatter`	key-value pair $\longrightarrow$ json	NA

We explore various tools to address different shortcomings of LLMs. The only constraints we impose are that (i) their inputs and outputs can be represented as texts, and (ii) we can obtain a few demonstrations of their intended use. Concretely, we explored a calculator, an OCR engine, a Python shell, a weather API, and a JSON formatter. Table 1 summarises the core tools in SLAM.

- **Calculator** : A simple calculator that can perform basic arithmetic operations. It can be used to solve mathematical problems that require calculations beyond the capabilities of the language model.
- **Python Shell** : A Python shell that can execute arbitrary Python code. It can be used to perform complex calculations, data processing, or any other task that can be expressed in Python.
- **Weather API** : A weather API that can provide current weather information for a given city. It can be used to answer questions about the weather in a specific location, such as "What is the weather in New York City today?".
- **OCR Engine** : An OCR engine that can extract text from images. It can be used to read text from images, such as scanned documents or photos of text.
- **JSON Formatter** : A tool that can format given key-value pairs into a JSON object. It can be used to convert structured data into a JSON format, which is commonly used for data interchange.

8 Related Work

ReAct [1] and Toolformer [2] showed that reasoning traces or synthetic demonstrations let LLMs master tool use. Gorilla [3] scales to thousands of APIs, while LangChain [4] provides orchestration for cloud backbones. Other lines explore tokenizer-aware prompt surgery, P-adapter fine-tuning, and program-of-thought decoding; none focus on a purely local, sub-4B stack with explicit tool schemas as the primary safety lever.

9 Conclusion

In this paper, we introduced SLAM, a lightweight and efficient framework that demonstrates how small language models can achieve agent-like behavior by leveraging external tools through a disciplined controller and well-defined schemas. By decoupling the responsibilities of prompt interpretation, dialogue generation, and task execution, SLAM achieves a modular and extensible architecture that is both privacy-preserving and deployable on resource-constrained hardware.

Our approach highlights the potential of compact models, such as T5 and Phi-4-Mini, when paired with a robust tool registry and a minimal JSON-based controller. The use of adapter-based fine-tuning for the T5 rewriter ensures that informal user queries are transformed into machine-friendly instructions, reducing errors and improving tool integration. Additionally, the safety guardrails implemented in the tool registry ensure secure and reliable execution of tasks, addressing common concerns with tool-augmented reasoning.

Through synthetic data generation and prompt-engineering heuristics, we demonstrated how SLAM can be adapted to diverse domains and tasks. The results show that even with sub-4B models, it is possible to achieve high accuracy and robustness in tool-augmented reasoning, challenging the notion that only large language models are capable of such feats.

SLAM's fully local pipeline, which avoids reliance on cloud APIs, makes it particularly suitable for privacy-sensitive applications and scenarios where internet connectivity is limited. By focusing on efficiency, extensibility, and safety, SLAM provides a compelling alternative to cloud-based solutions, paving the way for broader adoption of local-first AI systems.

In conclusion, SLAM represents a significant step forward in the development of lightweight, tool-augmented language models. It opens up new possibilities for deploying intelligent assistants in constrained environments while maintaining high levels of performance and reliability.

10 Future Work

Future work will focus on expanding the tool registry to include additional functionalities, such as a text summarizer, a document formatter, and a web search engine. We also aim to explore advanced tool schemas, including nested JSON structures, and enable more sophisticated interactions, such as chaining multiple tools to accomplish complex workflows. Furthermore, we plan to extend SLAM's capabilities to handle intricate tasks, including multi-turn dialogues and comprehensive task planning. We also aim to add multi language support via T5 rewriter fine-tuning, allowing SLAM to operate in various languages and domains. Finally, we will investigate the integration of SLAM with other AI systems, such as computer vision models, to create a more comprehensive and versatile AI assistant. We are also interested in adding a Text to Speech (TTS) module to SLAM, enabling it to convert text responses into spoken language. This would enhance the user experience by providing auditory feedback and making SLAM more accessible to users with visual impairments or those who prefer audio interactions.

11 Appendix

11.1 Contribution

Aravind SS:

- Led the development of the front-end interface for SLAM, ensuring a user-friendly and intuitive design.
- Contributed to the generation of synthetic datasets for training and evaluation.
- Played a key role in implementing and testing various tools integrated into SLAM.
- Assisted in the evaluation of SLAM's performance across diverse tasks.
- Actively participated in project management and coordination efforts.

Kokane Manoj Bhausaheb:

- Spearheaded the back-end development, focusing on the integration of the T5 rewriter and SLM framework.
- Conducted the training of the T5 model using adapter-based fine-tuning techniques.
- Contributed to the creation of synthetic datasets for tool-augmented reasoning.
- Played a significant role in implementing and refining the tool registry.
- Actively involved in project management and strategic planning.

Parth Bhatia:

- Designed and implemented the SLM-Agent framework, ensuring seamless interaction between components.
- Contributed to the generation of synthetic datasets and prompt-engineering heuristics.
- Played a key role in the development and testing of tools integrated into SLAM.
- Authored detailed documentation for the SLAM system, including technical and user-facing materials.
- Co-authored the research paper, focusing on system architecture and evaluation.

- Actively participated in project management and team coordination.

Sachin Bansal:

- Collaborated on the design and implementation of the SLM-Agent framework.
- Contributed to the generation of synthetic datasets and evaluation of SLAM's performance.
- Assisted in the development and testing of tools integrated into SLAM.
- Co-authored the research paper, focusing on evaluation and related work.
- Actively participated in project management and team coordination.

Srividhya L:

- Contributed to the generation of synthetic datasets for training and evaluation.
- Played a key role in implementing and testing various tools integrated into SLAM.
- Assisted in the evaluation of SLAM's performance across diverse tasks.
- Actively participated in project management and team coordination.

Swapnil Trivedi:

- Led the development of the front-end interface for SLAM, ensuring a seamless user experience.
- Contributed to the back-end development, focusing on tool integration and system optimization.
- Played a significant role in the generation of synthetic datasets and prompt-engineering heuristics.
- Authored detailed documentation for the SLAM system, including technical and user-facing materials.
- Co-authored the research paper, focusing on system design and future work.
- Actively participated in project management and strategic planning.

11.2 GitHub Repository

The SLAM project is hosted on GitHub, where you can find the complete source code, documentation, and instructions for running the system. The repository includes all components of SLAM, including the T5 rewriter, SLM generator, tool registry, and user interface.

https://github.com/aravindsreekumar077/SLAM-Small_Language_Agentic_Machine/

References

[1] Yao, S., Zhao, J., Weiss, P., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*.

[2] Schick, T., Dwivedi-Yu, J., Schütze, H., et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint arXiv:2302.04761*.

[3] Patil, S., Vashishth, S., et al. (2023). Gorilla: Large Language Models Connected with Massive APIs. In *Proceedings of the 40th International Conference on Machine Learning*.

[4] Chase, H. (2022). LangChain. https://github.com/langchain-ai/langchain.

Tools Used for Developing SLAM

- **T5-Base** : A compact encoder-decoder model for rewriting user queries into structured prompts.
- **Phi-4-Mini** : A lightweight decoder-only model for generating responses based on the rewritten prompts.
- **Llama.cpp** : A library for efficient inference of LLMs, used to stream tokens from the SLM generator.
- **Tesseract OCR** : An open-source OCR engine for extracting text from images.
- **Python** : Used for implementing the tool registry and executing Python code in a sandboxed environment.
- **GitHub** : For version control and collaboration on the SLAM project.
- **LaTeX** : For typesetting the paper and documenting the SLAM system.
- **TikZ** : For creating diagrams and visualizations of the SLAM architecture.
- **GPT-4o, DeepSeek R1, Claude Sonnet 4** : Used for synthetic data generation and prompt engineering.
- **Pegasus 568M** : A paraphrasing model used for generating diverse query-response pairs.
- **PyTorch** : For training the T5 rewriter with adapter-based fine-tuning.
- **NumPy, Pandas** : For data manipulation and analysis during synthetic data generation and evaluation.
- **Matplotlib, Seaborn** : For visualizing training loss curves and evaluation results.
- **Jupyter Notebook** : For interactive development and experimentation with the SLAM components.
- **Streamlit** : For building a lightweight user interface for SLAM.
- **Docker** : For containerizing the SLAM application, ensuring portability and ease of deployment.