SAGE: Sequential Agent Goal Execution Protocol for Multi-LLM Workflow Management

Muhammad Saim[1]

[1] National University of Computer & Emerging Sciences (FAST NUCES) Karachi

Abstract **:**
SAGE (Sequential Agent Goal Execution Protocol) is a modular protocol designed to orchestrate complex, multi-step AI workflows by decomposing user prompts into validated, goal-driven sub-tasks. Each sub-task is dynamically routed to the most suitable language model, executed, evaluated, and aggregated into a coherent final response. While the current demonstration leverages local language models via Ollama, SAGE is architected as an extensible and flexible framework, abstracting the workflow logic from any specific model provider or deployment environment.

Index Terms: Multi-LLM Workflow Orchestration, Workflow Automation, Prompt Decomposition, Model Routing.

1 Introduction

The increasing complexity of AI tasks often requires the coordination of multiple language models (LLMs), each with unique strengths. SAGE addresses this need by providing a protocol that decomposes user prompts, assigns sub-tasks to appropriate models, and ensures each step meets a defined success threshold before aggregation Saim, 2025. The protocol is designed for extensibility, reliability, and transparency, making it suitable for a wide range of AI orchestration scenarios.

2 System Architecture

SAGE is developed with the following core components:

1. **Decomposer Agent**: Breaks down the main user prompt into logical sub-prompts (sub-tasks) using rule-based heuristics.
2. **Router Agent**: Selects the best available model for each sub-task, using a meta-router LLM and fallback strategies.
3. **Execution Manager**: Executes sub-prompts sequentially using the assigned model, maintaining context between steps.
4. **Evaluator**: Judges the output of each sub-task against its expected goal using an LLM-based evaluation protocol, returning a success flag and similarity score.
5. **Retry/Reassign Handler**: Manages retries or reassignments if a sub-task does not meet the success threshold.
6. **Aggregator**: Combines all sub-task results into a final, coherent response.

The protocol is formally specified in `SAGE.spec.yaml` and implemented in the `src/sage/` directory.

3 Model Support and Demonstration

For demonstration purposes, the current implementation supports only local LLMs running via `Ollama`, specifically:

- `gemma3:4b`
- `deepseek-r1:1.5b`
- `qwen3:1.7b`

These models are configured in `config/settings.yaml` and are invoked through the `Ollama` API. The protocol, however, is designed to be model-agnostic and can be extended to support cloud-based or other local models with minimal changes.

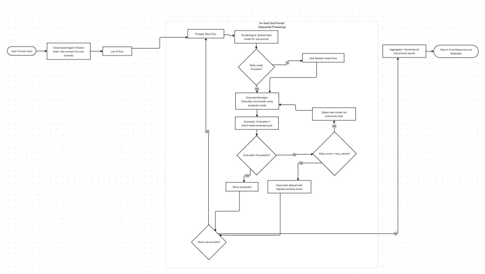
For Each Sub-Prompt (Sequential Processing)
User Prompt Input
DecomposerAgent: Breaks down user prompt into sub-prompts
List of Sub-
Process Next Sub-
RouterAgent: Selects best model for sub-prompt
Meta-router Success?
No
Use fallback model from
Yes
ExecutionManager: Executes sub-prompt using assigned model
Select new model not previously tried
Evaluator: Evaluates if result meets expected goal
Yes
Evaluation Successful?
No
Retry count < max_retries?
No
Yes
Store successful
Store best attempt with highest similarity score
Yes
More sub-prompts?
No
Aggregator: Combines all sub-prompt results
Return Final Response and Metadata

Figure 1. The SAGE Workflow Flowchart.

4 Workflow

The SAGE workflow proceeds as follows:

1. **Prompt Decomposition**: The user prompt is split into sub-prompts, each assigned a task type and expected goal.
2. **Model Assignment**: Each sub-prompt is routed to the most suitable model based on task type and meta-router LLM output.
3. **Sequential Execution**: Sub-prompts are executed in order, with context chaining as needed.
4. **Evaluation**: Each result is evaluated for correctness and sufficiency using an LLM-based evaluator.
5. **Retry/Reassignment**: If a sub-task fails to meet the similarity threshold, it is retried or reassigned to another model, up to a configurable maximum.
6. **Aggregation**: Successful results are aggregated into a final response, with metadata on execution and success rates.

All runs are logged to `sage_protocol.log` for audit and debugging.

5 Extensibility and Flexibility

SAGE is designed as an abstract protocol, not tied to any specific model provider or workflow. Key extensibility points include:

- Adding new agent types (e.g., for planning, validation, or post-processing)
- Integrating additional LLM providers (cloud or local)
- Customizing decomposition, routing, or evaluation logic
- Plugging in custom similarity metrics or feedback mechanisms

Configuration is managed via a YAML file, allowing easy adaptation to new models or strategies.

6 Limitations

- **Model Support**: The current version supports only local Ollama models for demonstration. Cloud LLMs (e.g., GPT, Claude) are not yet integrated.
- **Rule-Based Decomposition**: Prompt decomposition is currently rule-based and may require further development for more complex prompts.
- **Evaluation**: The evaluator relies on LLM-based judgment, which may be subject to the limitations of the underlying model.

7 Conclusion

SAGE establishes a robust and modular protocol for orchestrating multi-agent, multi-model AI workflows. By abstracting the decomposition, routing, execution, evaluation, and aggregation of sub-tasks, SAGE enables flexible integration of diverse language models and agent strategies. Its extensible architecture allows for seamless adaptation to evolving AI technologies and workflow requirements, making it a strong foundation for reliable and transparent AI task management across a wide range of applications.

References

- **Project Repository**: `https://github.com/saim-x/SAGE`