Samin Mahdipour Aghabagher – Samin. Mahdipour999@gmail.com

Architecture Overview: Lean 4 Theorem Prover System

1. Project Overview

This project implements an automated Lean 4 theorem prover powered by OpenAI API services. The system generates Lean code and formal proofs based on natural language descriptions of theorems. It leverages advanced language models for reasoning and proof generation, enabling automated theorem solving.

Key features include:

- A multi-agent architecture with distinct planning, generation, and verification phases.
- Dynamic feedback loops to handle errors and iteratively refine solutions.
- Fully generative pipeline without hard-coded solutions.

2. Agent Architecture

Agent	Role	Model	API Provider
Planning Agent	Decomposes problems into step-by-step strategies	gpt-4o	OpenAI API
Generation Agent	Produces Lean 4 code and formal proofs	gpt-4o	OpenAI API
Verification Agent	Executes Lean code and triggers feedback loops	Local	-

3. Workflow Overview

Step 1: Task Loading

- Reads problem description (`description.txt`) and Lean template (`task.lean`) from `tasks/task_id_*` directories.

Step 2: High-Level Planning

- The Planning Agent receives the problem description.
- It generates a plan detailing:
  - A concise problem summary.
  - The logical structure of the function.
  - Suggested Lean tactics for proof construction.

Step 3: Code and Proof Generation

- The Generation Agent utilizes `gpt-4o` to produce:
  - Lean code implementation (`{{code}}` section).
  - Formal proof of correctness (`{{proof}}` section).
- The agent operates based on the problem description, the Lean template, and the generated plan.

Step 4: Verification and Feedback Loop

- The system injects the generated code and proof into the Lean template.
- Executes the Lean code using the `execute_lean_code` function.
- If errors are detected, a feedback loop is triggered:
  - The agent revises the code and proof based on the Lean error output.
  - Up to three correction cycles are allowed per task.

Step 5: Final Output

- Returns a dictionary with the generated `code` and `proof`:

```
{
  "code": "...",
  "proof": "..."
}
```

4. Design Choices

- **Unified Model Usage:** Both planning and generation agents use `gpt-4o` for consistency.
- **Feedback-Driven Refinement:** The system iteratively corrects solutions using Lean compiler feedback.
- **No Hardcoded Solutions:** Outputs are fully generated based on task inputs and templates.
- **Extensible RAG Capability:** The architecture allows optional integration with a retrieval-augmented generation (RAG) system for enhanced reasoning.

5. Conclusion

This project demonstrates the integration of advanced language models with formal methods, enabling automated theorem proving in Lean 4. The architecture provides a scalable foundation for further enhancements, such as dataset expansion, improved prompt engineering, and advanced feedback mechanisms.