GitHub Repository Link: https://github.com/shahnawazsyed/476_final_project

Introduction

This agent's goal is to solve complex reasoning problems across multiple domains by selecting the best inference-time reasoning strategy. It consists of the following core components:

- API interface: provided LLM that provides answers for a given prompt and system instruction, subject to temperature and max tokens parameters
- Agent logic: Decisioning logic that calls a particular strategy based on the domain of the question
- Strategies: Chain of Thought Prompting (helper), Self Consistency, Self Refinement, Assumption Explicit Reasoning. These strategies frequently employ each other in their execution (see below)

The main execution flow is in generate_answer_template.py. For each prompt, it calls run_agent() in agent.py to extract an answer. After iterating over all prompts, it outputs the answers to a JSON file.

Agent routing and API architecture

The agent's core function is to map the problem's domain to a specific reasoning strategy. The mapping is defined below in the reasoning strategies section. The conditional logic is implemented in the run_agent(prompt, domain) function in agent.py. The domain is determined by the result of calling get_domain(prompt) in strategies.py, which calls the LLM to determine the topic of the question from the options: Math, Common Sense, Future Prediction, Coding, and Planning.

All strategies rely on a single, standardized function to communicate with the underlying LLM. This is the call_model_chat_completions() function in api.py. The function handles system prompts, user prompts, temperature settings, and maximum token limits.

Domain-Specific Reasoning Strategies (all located in strategies.py)

Strategy 1: Self-Consistency - self_consistency(prompt, isMath, num_samples, verbose)

This strategy is best used for the "math" and "common_sense" domains. For the former, the prompt first undergoes a conversion step to ensure LaTeX is converted to plain text which is easier to read by the LLM (convertToPlainText(prompt) in strategies.py). Self consistency concurrently (using ThreadPoolExecutor) generates multiple (default = 7) independent CoT samples using chain_of_thought() with random temperatures. It then selects the final answer based on the majority vote.

Strategy 2: Self-Refine - self_refine(prompt, domain, temp, max_iter, verbose)

This strategy is best used for the "planning" and "coding" domains. After first calling the API for an initial answer to the prompt, it utilizes an iterative process of gathering feedback via another LLM call, using a refine_sys_prompt that is specialized for the domain, and then combining this with the initial answer and prompt to create a new system prompt for a revised attempt. This process continues until a sentiment score above 0.7 is judged by the LLM or the maximum iteration limit is reached.

Strategy 3: Assumption-Explicit Reasoning - assumption_explicit_reasoning(prompt, domain, temp)

This strategy is best used for the "future prediction" domain. It enhances the standard CoT by extracting all implicit assumptions from an initial CoT response and then running the prompt through the model again with these in mind. This forces the LLM to address assumptions which may be unrealistic.

Strategy 4: Chain of Thought (CoT) reasoning - chain_of_thought(prompt, temp)

CoT reasoning is used if the result of get_domain() fails (i.e. returns an empty string or an invalid domain). It is also used as a helper method by nearly all other strategies. It works by providing an initial instruction to the model to provide a full, complete solution to the prompt while also detailing its thought process, forcing the LLM to "show its work". The final answer is then extracted from the initial output, including the steps if the prompt is related to planning.

Helper method: extract_final_answer(ans, isMath)

This helper method uses regex parsing to extract the final answer from LLM output. It searches (chronologically) for "final answer", multiple choice options (e.g. A, B, C), and the final number in the answer if the domain of the prompt is "Math". This provides a deterministic answer extraction from reasoning outputs.