Final Project Report

CSE 476: Intro NLP

By: James Price

Repo: https://github.com/jrpii/CSE-476-Final-Project

Techniques: CoT, ReAct, Inference Scaling, Tool Integration, LLM-as-a-Judge

For our final project, we were tasked with transforming a regular LLM into a capable, general agent using inference time techniques. The AI was callable via API with the model unknown, effectively making it a black box. In order to transform it into a capable agent that could perform well on a question answering benchmark across a range of domains and difficulties, we would have to implement advanced techniques in class, focusing on prompting, reasoning, and tools. Below outlines my agent loop and the techniques it utilizes.

My agent is built in python and accessible via the GitHub link at the top of the paper. To setup, simply create a conda environment, install requirements, and call the main method with the desired configuration. More detailed instructions available in the README with a copy and paste command to run the agent loop on the test dataset.

For the agent workflow, the process starts with data processing, where the main script loads the JSON file containing entries with at least the “input” field. It then moves on to part of a smaller method that helps to balance resource usage down the line, domain and difficulty inference. Using the LLM-as-a-Judge paradigm, we prompt the AI to determine the domain and difficulty for each question from a predetermined list (most prompts found and retrieved in src/prompts.py, and the guess_domain and guess_complexity methods are in src/agent.py).

With domain and difficulty inferred, we can move on to our next set of techniques and initial question inference. The call_agent block of the agents.py script holds the logic for a two-pass CoT reasoning system. First, the reasoning prompt is fetched from prompts.py depending on the inferred (or provided) domain. Next, the max reasoning output tokens and temperature are scaled by predicted complexity (multipliers in lines 127-140 of main.py). The input and domain-specific prompts are then passed to the agent to perform CoT reasoning. With that reasoning done, the loop then fetches the

extraction prompt and passes the CoT reasoning output to the model again and asks it to extract a well-formatted final answer (and gives examples from the dev data depending on the domain to take advantage of few-shot learning capabilities), normalizing that output with a heuristic if needed.

Although this is enough for some questions, performance is still lacking with this approach. For harder questions, we add on a ReAct agent loop that focuses on the think, act, observe cycle. The max amount of iterations of this cycle is calculated from inferred difficulty (and domain for coding and math) in lines 180-190 of main.py. We then initialize the ReAct loop with the question, initial Chain-of-Through reasoning, and reuse the difficulty-scaled temperature. This looprun_react_loop method lives in agent.py and differs from the CoT by adding a message log and the tool call functionality for acting and observing. A react loop calls the CoT reasoning, then if no final answer is given, asks the model to select an action (prompts for tool usage and formatting in prompts.py) then adds that observation to the conversation history and repeats the loop. For tool calling, a simple assortment of tools was provided in the src/tools.py script. The add math calculations through the Python math library, coding verification through the Python compiler that returns error status and location, and a simple reflect tool that just asks the model to check its reasoning, assumptions, and answer (a lightweight way to take advantage of the idea behind self-verification and consistency). Finally, if the model fails to submit a final answer within the allotted iteration count, one final pass call is made to the Chain of Thought agent to force it to output a final answer using the full conversation history as input.

That concludes the agent functionalities and control flow. I learned a lot from this and the repo serves as a good baseline for how an LLM can become an agent. A lot of the heavy lifting comes from smart prompting and knowing how to take advantage of the emergent properties and tendencies that make LLMs so generally capable. Though the values like reasoning effort, temperature, and max ReAct iterations need some tweaking, the general workflow is very capable and I am impressed. There are weak areas, the current configuration is heavy and alot of the reasoning could be cut down on some problems by adding a proper search tool, but overall the agent is successful.