PromptGen: Dataset Compiler for LLM Verification

Divyansh Rajesh Jain
Department of Computer Science
University of California, Davis
drajeshjain@ucdavis.edu

Varsha Sivaprakash
Department of Computer Science
University of California, Davis
varsivaprakash@ucdavis.edu

Abstract

Compiling a large dataset of domain-specific prompts and correct answers. Auto-generating datasets for LLM verification is a non-trivial task that PromptGen will tackle. Using constraint based programming, PromptGen will auto generate a large set of prompts along with corresponding answers to verify and finetune/repair an LLM.

1 Introduction

Despite high performance accuracies in modern LLMs, these models are still prone to unpredictably producing hallucinated outputs. Often, the problem is in verifying whether an LLM is good at a certain task. Another way of thinking about this problem is to think about generating a corpus of text that is good enough to fine-tune an LLM for a certain downstream task. We aim to chip away at this problem by proposing a method that will allow domain experts to codify constraints that exist in their domain, which we can then use to generate a dataset to fine-tune/repair an LLM or use it to verify the effectiveness of an already fine-tuned LLM.

1.1 Problem Being Addressed

Our goal is to create a domain-specific compiler that can dynamically generate huge amounts of prompts along with their answers, which can then be used to either fine-tune or verify LLMs. Most likely, we will choose a sports-related domain to detect and repair as many possible hallucinations/inaccuracies in this domain.

1.2 Key Insights

For the chosen domain, our key insight is aiming to encode constraints using Prolog, which is a logic-based programming language that allows the user to specify relationships in a declarative way. Our system will take these Prolog predicates, domain-specific data which exists in an online database, and natural-language templates per predicate to be able to generate a large variety of prompts and answers which can then be used to verify LLMs or even fine-tune them.

1.3 Evaluation and Expected Outcomes

We will utilize the datasets generated by PromptGen to fine tuning or repair a small transformer architecture, such as the nanoGPT project by Andrej Karpathy, to see if we can fine-tune this LLM better than if we hand-wrote a set of prompts for the fine-tuning. We will evaluate both of these models on a preselected set of questions about the domain, and empirically reason about the effectiveness of our approach with the current most commonly used approach in sourcing datasets. Our expectation is that the fine tuned model will be able to generalize for the same prompts phrased differently, allowing higher accuracy.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

2 Related Work

In the following section we present a similar work utilizing constraint based LLM verification, along with its limitations and challenges.

2.1 End User LLM Verification

Similar works in the past include VeriPlan, a self-verification LLM system that formally verifies and corrects the outputs of LLM models for planning tasks based on user prompts. The key steps of VeriPlan include translating the rules, changing flexibility and checking the model outputs. It utilizes a mapping agent which is LLM-based to extract the constraints from a user's prompt. This includes temporal constraints that are time-based constraints on the prompts so that they make logical sense. The flexibility sliders allow users to choose the strictness of the constraints so the constraints have weights while rejecting the model outputs. If the model is rejected, the requirement violations are fed back into the LLM over several iterations until the LLM outputs according to the constraints. If the LLM doesn't output correctly even after iterations, the user has the option to change constraints and flexibilities.

A main limitation of the VeriPlan system is that it is constrained to a small set of prompts for scheduling events based on temporal constraints. Our hope is to include a more diverse range of prompts within our chosen domain. However, this would also mean that our answers won't be constrained to the facts from the user's prompt. To overcome this, we'll also include a database to get answers from using constraints. The VeriPlan system needed the user on the loop to adjust constraints or change their flexibilities based on results despite only a small class of constraints. We anticipate that our main challenge will be ensuring that our constraints are able to capture the scope of our questions and accurately unify prompts with answers in the database. For feasibility, we are restricting ourselves to a small domain with a concise database.

References

[1] Lee, C. Porfirio, D. Wang, X. J. Zhao, K. C. & Mutlu, B. (2025) VeriPlan: Integrating Formal Verification and LLMs into End-User Planning.