# 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 can be used 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 correct answers, given a set of constraints about a specific domain. The purpose of creating these datasets is to evaluate the LLM on its knowledge of the specific 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, and natural-language metadata per predicate to be able to generate a large variety of natural language prompts and answers which can then be used to verify LLMs or even fine-tune them.

### 1.3 Evaluation and Expected Outcomes

The generated datasets will be both quantitatively and empirically evaluated. We'll use quantitative scores such as the BLEU score to determine how similar the NLG prompts are to human written prompts. For empirical evaluation, we'll run both generated and human written prompts into an LLM such as Llama or NanoGPT. The purpose of this evaluation is to determine the effectiveness of the generated prompts by seeing how many of those prompts cause hallucinated outputs by the LLM compared to the handwritten prompts. Our expectation is the generated prompts will be closely related to the human written prompts in terms of natural language, and the generated prompts will be more likely to detect hallucinations in an LLM because of its large quantity.

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

## 1.4 Examples

Below is a small sample of the Prolog constraints and the Natural Language Constraint annotations provided by the user.

1: logic.pl

```
% Facts
person(alice).
person(bob).
person(claire).

food(pizza).
food(sushi).

like(alice, pizza).
like(bob, pizza).
like(bob, sushi).
like(claire, sushi).

% Rule
friend_of(A, B, F) :-
    like(A, F),
    like(B, F),
    A \= B.
```

2: annotations.txt

```
% --- BUILT-INS SAMPLE ---
type noun
type verb
type pronoun
type adverb
type adjective

% --- TYPE HIERARCHY ---
type food :: noun.object
type person :: noun.object
type like :: verb.object

% --- FACT METADATA ---
relation like(A: person, B: food)
  verb_phrase = "likes"
  question_template =
      binary_relation
  description = "A likes B"

% --- RULE METADATA ---
relation friend_of(A: person, B:
   person, F: food)
  question_template =
      shared_preference
  description = "A and B both
      like F"
```

The logic.pl file defines a set of facts and relations. Facts associate words with user defined data types (eg. person(Alice)), or can define a relation between two words (eg. like(alice, pizza)).

The annotations.txt file is where the user can define the type inheritance structure from the built in classes. It is also where the user can define the meta data associated with the facts and relations, which essentially provides information on some natural language constraints. For example, the user can specify some temporal constraints for relations (eg. verb phrase = "likes" tells the system that the relationship is of present simple tense). The user can also map these relations to some high level templates such as binary relation. In our backend, we'll have some database of different of English components, such as adjectives, verbs, nouns, pronouns, etc. The high level templates will map to certain permutations of these sets to create natural langauge prompts. Below is a set of sample questions and answers from the above constraints and natural language annotations:

3: prompts.txt

```
Q: What does Alice like?
A: Alice likes pizza.

Q: What does Bob like?
A: Bob likes pizza and sushi.

Q: What does Claire like?
A: Claire likes sushi.

Q: Who all likes pizza?
A: Alice and Bob like pizza.

Q: Who all likes sushi?
A: Claire and Bob like sushi.
```

```
Q: What do Alice and Bob both like?
A: Alice and Bob both like pizza.

Q: What do Bob and Claire both like?
A: Bob and Claire both like sushi.

Q: Does Bob like sushi?
A: Yes
```

## 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 with our set of 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. However, PromptGen will ensure that the generated natural language sets are accurate based on the constraints to avoid the user having to check them. We anticipate the natural language correctness guarantee to be the most challenging for PromptGen, so we'll start off with a subset of English for greater feasibility.

## 3 Experimental Setup

### 3.1 Research Questions the Experiment is Designed to Answer

The main experimental questions that we are trying to answer are if we can generate many natural language prompts and accurate answers with an enhanced logic program as input. We think this experimental question is significant because it allows us to explore a very different approach to LLM verification than has been taken in the past. In our approach, the evaluation dataset is NOT handwritten, but "declared" by a programmer through a logic program to encode constraints that exist in the data for a particular domain, and a few extra natural language annotation constraints to assist our system in generating question and answer pairs, which can be used to evaluate the local effectiveness of an LLM in that particular domain.

### 3.2 Datasets Used

The main dataset that we are using for the experiments is a database called Geobase. This contains both prolog facts about important geographic facts about the United States Geography, and some natural language prompts (Question and Answer Pairs). This is the right dataset for this experiment because it is a complex, logic-based domain, which are the exact parameters of our system. Additionally, it also contains Question Answer Pairs, which we can use to cross-check with the type of Question Answer Pairs that our system can generate. This will allow us to evaluate the effectiveness of our system by using the Question Answer Pairs in the dataset as the "ground truth" prompts, to measure the expressiveness of our system, which is the main goal of our experiments.

### 3.3 Evaluation Metrics

The evaluation metrics will be a two-stage evaluation metric.

The first stage of our evaluation metric will be a quantitative measure that measures the coverage difference between the prompts (Question Answer Pairs) that our system can generate vs the prompts that exist in the geobase system. We hope to show that our declarative system allows the programmer to express equally or more complex prompts than those in the geobase dataset, and also show that the generated prompts can cover the underlying domain better than the existing prompts in the geobase dataset. A specific quantitative metric we'll use is the BLEU score. This is a score between the range 0 and 1 that is indicative of how similar a machine generated natural language sentence is to that written by a human. We'll hand pick prompts from the geobase that are similar to the ones generated by the system and compute the BLEU score on them to quantify how similar the prompts are to human natural language.

The second stage of our evaluation metric will be an empirical measure to try to express the benefit of using our system vs existing solutions to LLM verification. We will measure the programmer time (the time it takes to write the code to feed into our system) vs the human time to handwrite prompts for the same domain, and the failure rate of the LLM of the programmed vs human evaluation sets (higher failure rate of the LLM is better, because it means the question was too complex for the LLM to answer) for a small and simple domain. Our domain of choice is Computer Science Undergraduate Advising.

### 3.4 Baselines

As mentioned before, the main baseline that we are comparing our system to is the prompts (Question Answer Pairs) that already exist in the geobase dataset. Our main evaluation criterion is coverage difference, which illustrates how much of the geobase prompts our system can recreate. This is to show that the results of our system is at least as good as human written evaluation sets (the main way to currently evaluate LLMs) and we hope to convince the reader through empirical evidence, that even if the best case result is to recreate the prompts the human can generate, that the declarative and programmatic style of our approach to LLM evaluation dataset generation is cheaper, faster, and easier to change when the underlying requirements of the domain change.

## 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.