PromptGen: Dataset Compiler for LLM Verification

Divyansh Rajesh Jain
University of California, Davis
Davis, CA 95616, USA
`drajeshjain@ucdavis.edu`

Varsha Sivaprakash
University of California, Davis
Davis, CA 95616, USA
`varsivaprakash@ucdavis.edu`

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks, yet they continue to exhibit unpredictable behavior and occasional factual inaccuracies. A persistent challenge in their deployment is the lack of systematic methods for evaluating model performance within specific domains. We introduce **PromptGen** , a framework that empowers domain experts to generate high-quality, structured evaluation datasets by encoding domain knowledge and constraints using logic programming. **PromptGen** leverages Prolog to represent domain-specific facts and relationships, augmented with natural language annotations that define how these logical structures should be verbalized. Through unification, the system derives all valid instantiations of a domain's logic and automatically converts them into well-formed English question-answer pairs. This approach provides a scalable, declarative alternative to manual prompt crafting, facilitating the evaluation, fine-tuning, and repair of LLMs in a domain-aware manner. **PromptGen** bridges the gap between symbolic reasoning and natural language generation, offering a principled method for aligning LLM evaluation with structured domain expertise. The code for this project can be found at our GitHub repository.

1 Introduction

Despite the impressive performance of modern large language models (LLMs), they remain prone to generating incorrect or inconsistent outputs in unpredictable ways. A core challenge lies in reliably evaluating whether an LLM performs well on a specific task or within a particular domain. To address this, we propose a method that allows domain experts to formally specify the constraints inherent to their domain, enabling the automated generation of evaluation datasets that can be used to evaluate, or fine-tune LLMs.

The central research question we explore is whether it is possible to design a system that enables users to define domain-specific constraints in a structured, logic-based format, such that the system can automatically generate well-formed English question-answer (QA) pairs. These QA pairs serve as targeted benchmarks for evaluating an LLM's capabilities in that domain.

The key insight behind our system, **PromptGen** , is to leverage Prolog as the underlying representation language for both domain constraints and associated natural language annotations. **PromptGen** uses Prolog's unification mechanism to exhaustively derive valid facts from the logic program and then applies user-provided annotations to convert these facts into fluent and grammatically correct QA pairs. This approach represents a significant departure from the standard practice of manually writing evaluation prompts. Rather than composing prompts by hand, users encode their domain knowledge through an enriched logic program. **PromptGen** then synthesizes diverse and high-coverage QA pairs directly from this structured representation, offering a principled and scalable alternative for generating evaluation datasets tailored to specific domains

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

2 Related Works

VeriPlan [1] is a self-verifying system that enables large language models (LLMs) to validate and correct their outputs for planning tasks by formally encoding constraints derived from user prompts. It employs an LLM-based mapping agent to extract logical and temporal constraints, incorporates a mechanism for adjusting constraint flexibility, and iteratively refines outputs based on constraint violations. If the generated output fails to satisfy the constraints, the violations are fed back to the model in successive rounds of refinement; users may also intervene by modifying constraint definitions or their strictness. However, VeriPlan is limited in scope, focusing primarily on a narrow class of temporal scheduling prompts and requiring human oversight to ensure constraint satisfaction. In contrast, PromptGen aims to generalize across a broader range of domains by allowing users to express domain constraints declaratively in logic, which are then automatically translated into well-formed English question-answer pairs. This eliminates the need for manual output verification, though a key challenge remains in ensuring semantic and grammatical correctness in the generated language—a challenge we address by initially constraining generation to a well-defined subset of English.

3 Approach

To address our research question, we developed a Prolog-based system that enables users to specify domain-specific constraints and generate a large number of question-answer (QA) pairs with correct answers. We implemented two versions of the system, each differing in the interface for specifying constraints, in order to explore trade-offs between expressiveness and usability.

System 1 emphasizes a minimal yet expressive interface, enabling users to generate QA pairs across a variety of fact-based domains with concise specifications. In contrast, System 2 offers a more verbose interface that supports richer natural language constructs, such as verb tense and more complex sentence structures.

3.1 Implementation Framework

Both systems share a common implementation framework, which consists of four major components:

1. Extensible Type Hierarchy The first step involves defining an extensible English-type hierarchy used to annotate variables in standard Prolog relations, which encode the logical structure of a domain. This hierarchy must support user-defined types to adapt to different domains. For instance, in System 1, the base types include `person`, `location`, `date`, `number`, and `object`. Users can extend these using the `type(Type1, Type2)` relation, which asserts that `Type1` is a subtype of `Type2`. For example, `type(food, object)` declares that `food` should be treated as a kind of `object`.

2. Argument Type Annotation The second step allows users to annotate the type of each argument in a given Prolog relation. In System 1, this is handled through the `arg/2` predicate, which takes the form `arg(RelationName, [Type1, Type2, ...])`. For example, if the relation `likes_food(X, Y)` expresses that person `X` likes food `Y`, it would be annotated as `arg(likes_food, [person, food])`, specifying the expected types for each argument.

3. English Meta-Template Mapping Next, we define a set of high-level English metatemplates that describe how Prolog relations are mapped to natural language QA formats. Each template is associated with a specific argument signature. In System 1, we provide three such templates:

- **Canonical:** Standard Wh-questions such as "What is your name?" Applicable to binary relations where the first argument is of any base type and the second argument is of type `object`.
- **Possessive:** Questions of the form "What is Nevada's capital?", also for binary relations where the first argument is of any base type and the second argument is of type `object`.
- **Canonical Negative:** Negative Wh-questions like "What states border California but not Nevada?", designed for ternary relations where all arguments are of type `object`.

Each metatemplate defines how arguments of specific relations should be rendered into grammatically correct question-answer pairs. The `predicate_bucket` relation `predicate_bucket(pred, metatemplate)` is used to annotate the metatemplate that each predicate in the logic program should be mapped to.

4. Natural Language Enrichment The final step supports additional annotations to enhance the fluency and grammaticality of the generated language. In System 1, this includes:

- **Verb annotations**, such as `verb(likes_food, ["like", "likes"])`, to indicate the correct singular and plural forms of the verb for the `likes_food` relation.
- **Modifier annotations**, which allow the inclusion of descriptive phrases, specified in singular and plural forms to enrich sentence structure.

3.2 End-to-End Flow

Together, these annotations form a complete specification that allows the system to automatically generate QA pairs. The system leverages Prolog's unification engine to enumerate all valid facts that satisfy the annotated relations. These are then translated into natural language using the metatemplates and annotations provided by the user.

An example of a complete annotation file for System 1 is shown below:

Listing 1: annotations.pl

```
generate_predicates([
    abbrev, simple_border, complex_logic_border, riverflow,
    statecapital, population, stateriverflow, statemajorcity
]).

type(myriver, object).
type(mystate, object).

arg(simple_border, [mystate, mystate]).
arg(riverflow, [myriver, mystate]).
arg(stateriverflow, [mystate, myriver]).
arg(statemajorcity, [mystate, object]).
arg(complex_logic_border, [mystate, mystate, mystate]).
arg(population, [mystate, object]).
arg(abbrev, [mystate, object]).
arg(statecapital, [mystate, object]).

verb(simple_border, ["border", "borders"]).
verb(riverflow, ["flow␣through", "flows␣through"]).
verb(stateriverflow, ["contain", "contains"]).
verb(statemajorcity, ["contain", "contains"]).
verb(complex_logic_border, ["border", "borders"]).
verb(population, ["population", "population"]).
verb(abbrev, ["abbreviation", "abbreviation"]).
verb(statecapital, ["capital", "capital"]).

modifier(simple_border, ["state", "states"]).
modifier(complex_logic_border, ["state", "states"]).
modifier(riverflow, ["state", "states"]).
modifier(stateriverflow, ["river", "rivers"]).
modifier(statemajorcity, ["major␣city", "major␣cities"]).

predicate_bucket(simple_border, cannonical).
predicate_bucket(riverflow, cannonical).
predicate_bucket(stateriverflow, cannonical).
predicate_bucket(statemajorcity, cannonical).
predicate_bucket(complex_logic_border, cannonical_negative).

predicate_bucket(population, possessive).
predicate_bucket(abbrev, possessive).
predicate_bucket(statecapital, possessive).
```

Using this framework, the system can systematically and scalably generate high-quality, grammatically correct QA pairs suitable for evaluating the domain-specific performance of large language models.

4 Experiment Evaluation

4.1 Experimental Setup

The primary objective of our experiments is to evaluate the degree to which our generated prompts resemble human-written prompts and to assess their effectiveness in verifying the behavior of a language model. To investigate this, we conducted a two-part evaluation framework.

As a baseline, we selected the `Geobase` dataset, which consists of U.S. geography facts represented in Prolog, alongside manually authored question–answer pairs. This dataset is well-suited to our goals, as it covers a complex, logic-based domain that aligns with the nature of our system. The human-written questions in `Geobase` serve as a reference point, enabling us to measure how accurately and naturally our generated prompts reflect the intended queries.

For the language model under evaluation, we chose LLaMA 2 (7B), a 7-billion-parameter model trained on 2 trillion tokens and fine-tuned using over 1 million human annotations. This model offers a practical balance between scalability and output quality, making it an appropriate choice for evaluating a large volume of prompts.

Part 1 of our evaluation compares the generated prompts to similar ones from the `Geobase` dataset, using BLEU and BERTScore to measure both syntactic and semantic similarity. To ensure a fair and meaningful comparison, we selected `Geobase` prompts across various categories that closely aligned in meaning with prompts from our dataset.

Part 2 focuses on evaluating the response quality of the LLaMA 2 model when prompted with our generated questions. The model's answers are compared against ground-truth responses from our dataset using accuracy, BLEU, and BERTScore. This evaluation is designed to measure how effectively our prompts elicit correct and semantically appropriate responses.

For the accuracy metrics, we performed a direct string match between the model's output and the ground-truth answer. A response was considered correct only if it matched exactly.

For the similarity metrics, we computed:

- BLEU scores to assess surface-level overlap between generated and reference answers.
- BERT precision, recall, and F1 to evaluate semantic similarity, capturing deeper meaning even when the wording differs.

This multi-metric approach allows us to assess both the literal correctness and the semantic adequacy of the LLM's responses, providing a more comprehensive picture of how well the generated prompts perform in practice.

In total, over 400 prompts in varying degrees of complexity, sampled from a pool of 30,000 generated prompts, were used to evaluate the language model. We ran these evaluations both with and without system prompts to see whether the system-level instructions we used to shape the model's responses introduced any unintended bias, particularly in cases where we needed outputs to align with strict string-matching criteria.

4.2 Experimental Results and Analysis

4.2.1 Part 1: Structural and Semantic Similarity Evaluation

In Table 1, we present the BLEU and BERTScore results comparing our generated prompts to those in the baseline dataset, `Geobase`. Prompt pairs used for this comparison are shown in Figure 1.

Table 1: BLEU and BERTScore results

Category	BLEU Score	BERT Precision	BERT Recall	BERT F1
Border Question	11.48	0.9394	0.9573	0.9482
Capital Question	9.42	0.9690	0.9682	0.9686
Population Question	8.75	0.9263	0.9268	0.9265
River in State Question	10.68	0.9529	0.9529	0.9529
Major Cities in State Question	13.54	0.9344	0.9433	0.9388

```
{
 "data": [
  {
   "category": "Border Question",
   "reference_question": "what states border montana?",
   "system_generated_question": "What states does montana border?"
  },
  {
   "category": "Capital Question",
   "reference_question": "what is the capital of washington?",
   "system_generated_question": "What is washington's capital?"
  },
  {
   "category": "Population Question",
   "reference_question": "how many citizens in alabama?",
   "system_generated_question": "What is alabama's population?"
  },
  {
   "category": "River in State Question",
   "reference_question": "what rivers are in utah?",
   "system_generated_question": "What rivers does utah contain?"
  },
  {
   "category": "Major Cities in State Question",
   "reference_question": "what are the major cities of texas?",
   "system_generated_question": "What major cities does texas contain?
      "
  }
 ]
}
```

Figure 1: Paired prompts from `bleu_experiment/data.json`

As shown in the figure, the relatively low BLEU scores suggest that our system's prompts often differ from human-written questions in surface structure and word choice. This is expected, as BLEU focuses on n-gram overlap and penalizes phrasing variation. In contrast, the high BERTScore indicates a strong degree of semantic similarity between the generated prompts and the baseline.

For example, the `Geobase` question *"How many citizens in Alabama?"* and the system-generated prompt *"What is Alabama's population?"* convey the same intent despite differing vocabulary. This illustrates the system's ability to produce meaning-preserving prompts even with varied phrasing.

Overall, these results suggest that while our system may not always replicate the syntactic style of human-authored prompts, it is effective at generating questions that retain the underlying semantics.

4.2.2 Part 2: Verifying LLM Performance

Tables 2 and 3 present evaluation results of the LLaMA 2 (7B) model when prompted without and with manually engineered system-level instructions, respectively.

Table 2: Llama Evaluation Results with **No** System Prompts

Category	**Q's Gen**	**Q's Asked**	**Correct**	**Passed**	**BLEU**	**BERT P.**	**BERT R.**	**BERT F1**
Abbreviation Knowledge	51	51	46	90.2%	93.10	0.99	1.00	0.99
Capital Knowledge	51	51	18	35.3%	51.39	0.95	0.96	0.96
Rivers Knowledge	46	46	0	0.0%	3.59	0.81	0.91	0.86
State Simple Border Questions	49	49	0	0.0%	3.25	0.83	0.92	0.87
States Complex Border and Location questions AND	543	100	0	0.0%	25.08	0.90	0.95	0.92
States Complex Border Questions AND BUT	23601	100	0	0.0%	12.84	0.84	0.91	0.87
States Complex Total population	19600	100	0	0.0%	4.77	0.82	0.89	0.85

Table 3: Llama Evaluation Results with System Prompts

Category	**Q's Gen**	**Q's Asked**	**Correct**	**Passed**	**BLEU**	**BERT P.**	**BERT R.**	**BERT F1**
State Abbreviation Knowledge	51	51	50	98.0%	98.39	1.00	1.00	1.00
State Capital Knowledge	51	51	48	94.1%	95.56	0.99	0.99	0.99
Rivers Knowledge	46	46	4	8.7%	54.09	0.96	0.98	0.97
State Simple Border Questions	49	49	3	6.1%	45.05	0.97	0.97	0.97
States Complex Border and Location questions AND	543	100	37	37.0%	74.21	0.97	0.98	0.98
States Complex Border Questions AND BUT	23601	100	2	2.0%	43.34	0.95	0.95	0.95
States Complex Total population	19600	100	0	0.0%	29.93	0.89	0.91	0.90

Figure 2: Inaccuracies ellicited by Generated Prompts

Question: Which state do Illinois, Ohio, and Virginia neighbor in common?

LLM Answer: Illinois, Ohio, and Virginia all border the state of Pennsylvania. Therefore, they share a common border with Pennsylvania.

Correct Answer: Illinois, Ohio, and Virginia neighbor Kentucky in common.

Figure 3: No System Prompt, Wrong Answer

Question: What is district of columbia's capital?

LLM Answer: *The District of Columbia (DC) does not have a capital...*

Correct Answer: District of Columbia's capital is Washington.

Figure 4: System Prompt, Right Answer

Question: What is district of columbia's capital?

LLM Answer: *Washington D.C.'s capital is Washington.*

Correct Answer: District of Columbia's capital is Washington.

Figure 5: High Similarity, Factual Error

Question: What states does California border?

LLM Answer: *California borders Arizona, Nevada, Oregon, and Mexico.*

Correct Answer: California borders Arizona, Nevada, and Oregon.

Scores: BLEU = 53.48; BERT Precision = 0.97, Recall = 0.99, F1 = 0.98

We observed a clear trend in both evaluations: accuracy, precision, recall, and F1 scores declined as the question complexity increased. Regardless of prompt condition, the model occasionally made factual errors, as illustrated in Figure 2. This supports the value of our prompts in identifying model weaknesses and guiding model verification.

Though system prompts generally improved accuracy, they sometimes introduced bias. For instance, in the category *Geography: U.S. State Abbreviation Knowledge*, all responses marked incorrect without system prompts were semantically correct but differed in phrasing. Consequently, the real accuracy in this category is effectively 100% without system prompts. In contrast, one actual factual error occurred with system prompts, yielding a real accuracy of 98.0%.

In the case shown in Figure 5, the LLM mistakenly included *Mexico* as a U.S. state, yet the similarity metrics remained high. This reveals a limitation of BLEU and BERTScore: they may signal high agreement even when responses contain factual inaccuracies.

Dynamic Information and Accuracy Questions involving dynamic information—such as population—proved especially difficult. This category scored 0% accuracy across both runs. Since population is time-sensitive and can differ by source, multiple plausible answers exist, making strict string matching unsuitable. This result emphasizes the importance of designing constraints that elicit verifiable, concrete answers when using string-based evaluation.

5 Conclusion

In this work, we presented a Prolog-based framework designed to generate large sets of domain-specific question–answer pairs by allowing users to specify precise constraints. By implementing two system versions with different interfaces, we explored the balance between expressiveness and ease of use in prompt generation.

Our evaluation showed that the prompts generated by our system, while structurally distinct from human-written examples, closely match their semantic content, as evidenced by high BERTScores despite lower BLEU scores. This indicates that our approach effectively captures the intended meaning of complex queries even if phrasing varies.

Testing the LLaMA 2 (7B) model with these prompts demonstrated the system's strength in detecting factual errors, especially as question complexity increased. We observed that system prompts generally improved accuracy but could sometimes introduce bias, underscoring the need for careful prompt design. Additionally, the limitations of current automated evaluation metrics, such as BERTScore and BLEU, became apparent, particularly in distinguishing factual correctness.

Overall, our approach offers a powerful tool for generating meaningful, constraint-driven prompts that can rigorously verify large language models in logic-based domains. Future work may focus on expanding the system to generate more logically complex prompts, along with stronger evaluation metrics.

References

[1] Christine P. Lee, David Porfirio, Xinyu Jessica Wang, Kevin Chenkai Zhao, and Bilge Mutlu. Veriplan: Integrating formal verification and llms into end-user planning. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, page 1–19. ACM, April 2025. doi: 10.1145/3706598.3714113. URL `http://dx.doi.org/10.1145/3706598.3714113`.