Symbolic Prompt Program Search: A Structure-Aware Approach to Efficient Compile-Time Prompt Optimization

Tobias Schnabel and **Jennifer Neville**
Microsoft Research, Redmond, USA
{toschnab,jenneville}@microsoft.com

Abstract

In many modern LLM applications, such as retrieval augmented generation, prompts have become programs themselves. In these settings, prompt programs are repeatedly called with different user queries or data instances. A big practical challenge is optimizing such prompt programs. Recent work has mostly focused on either simple prompt programs or assumed that the structure of a prompt program is fixed.

We introduce SAMMO, a framework to perform *symbolic* prompt program search for *compile-time* optimizations of prompt programs. SAMMO represents prompt programs on a symbolic level which allows for a rich set of transformations that can be searched over during optimization. We show that SAMMO generalizes previous methods and improves the performance of complex prompts on (1) instruction tuning, (2) RAG pipeline tuning, and (3) prompt compression, across several different LLMs. We make all code available open-source at https://github.com/microsoft/sammo.

1 Introduction

With the recent development of large language models (LLMs) such as Mixtral 8x7B (Jiang et al., 2024) or GPT-4, it is now possible to provide LLMs with longer inputs including richer context and more detailed instructions. As a result, the complexity of prompts has increased, including longer strings of instructions, demonstrations or examples, and specification of more structured output. These lengthy instructions are often reused as part of larger prompts, where static information (e.g., instructions) is combined with input-dependent information (e.g., user queries, retrieved documents in RAG systems) at runtime. This has led to prompts being regarded as programs themselves, where each part is the result of other subprograms (Chase, 2022; Khattab et al., 2023).

As LLM performance depends on a variety of factors such as paraphrasing, formatting and ordering (Sclar et al., 2023; Liu et al., 2024), a key challenge to solve in practice is automatically finding the best prompt program for a given task and model. In this paper, we focus on *compile-time optimization* of prompt programs under *black-box LLM* access. Compile-time optimization is carried out only once before deployment and has two large advantages: (i) we can amortize the optimization cost over multiple uses of the prompt program and (ii) keep the existing run-time architecture unchanged. This is different from run-time optimization approaches, e.g., token pruning (Jiang et al., 2023) which are called every time before inference and thus have non-amortizable costs.

Table 1: SAMMO in comparison to other large prompt optimization and programming frameworks.

	SAMMO	DSPy	LangChain
optimize instructions	✓	✓	×
optimize fewshot examples	✓	✓	×
optimize full structure	✓	×	×
optimize data format	✓	×	×
combine mutators	✓	×	×
compress prompts	✓	×	×

A lot of work on prompt optimization has focused on simple prompt programs where the program consists of a single string (which is subsumed by our representation in Section 2), mostly using structure-unaware operations such as paraphrasing (Chen et al., 2023; Pryzant et al., 2023; Zhou et al., 2023). Another line of work has focused on optimizing prompts with more complex programmatic structure, but still considers only *static* prompt structures, e.g., DSpy (Khattab et al., 2023). Initial work in this direction focused on applying textual mutators to different prompt components (Fernando et al., 2023; Ye et al., 2023; Khattab

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 670–686
November 12-16, 2024

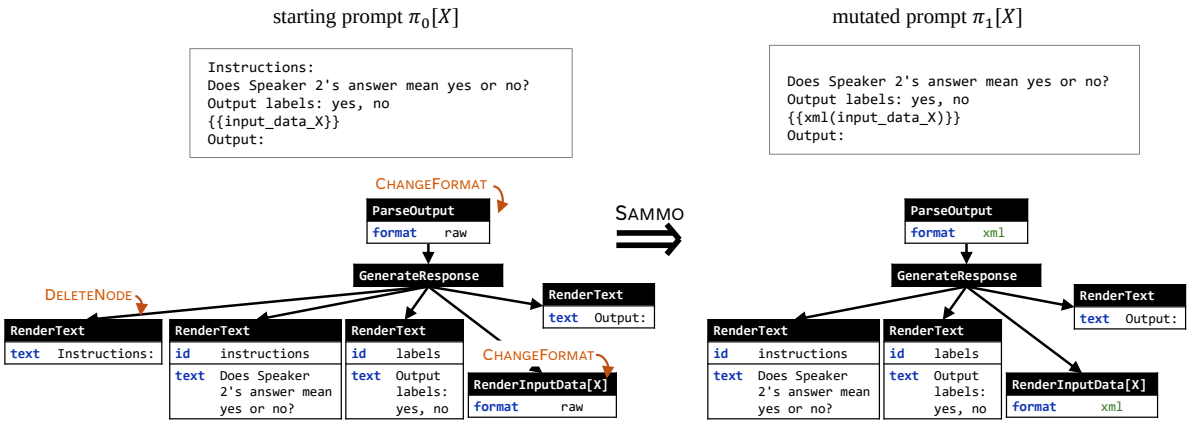

Figure 1: Left: Symbolic prompt program (SPP) for a binary classification task, where each node is a function with attributes and dependencies (children). The example also shows how the SPP allows for structural changes (e.g., DELETENODE) and attribute-based changes (e.g., CHANGEFORMAT) which, after applying, result in the mutated prompt (Right). These enable SAMMO to explore a large set of possible prompt candidates automatically.

et al., 2023). Subsequent work considered both textual mutation and hyperparameter selection (Sclar et al., 2023). However, with increasing complexity of prompt structure, many prompt optimization techniques are no longer applicable and a new approach is needed that is able to optimize complex prompt programs with their hyperparameters.

To address this, we introduce SAMMO, a general purpose framework for compile-time optimizations of prompt programs. Different from existing approaches SAMMO represents prompt programs as *symbolic prompt programs* (SPP) which are abstract program graphs that can be changed arbitrarily. This allows SAMMO to efficiently search through the space of valid prompt programs by automatically generating new promising candidates through mutations. Moreover, SAMMO allows to both represent the internal structure of prompts (e.g., sections) as well as program structure (e.g., calling a retriever) in a unified fashion. By writing custom mutation operators, practitioners can encode their domain knowledge and guide the search process. SAMMO naturally extends previous prompt programming approaches such as DSpy (Khattab et al., 2023) and encompasses several specialized prompt tuning methods as special cases. Through SPPs, SAMMO (i) allows dynamic changes to prompt programs, (ii) generalizes compile-time optimizations to all prompt components (e,g., textual content, hyperparameters). Table 1 summarizes and compares different capabilities between SAMMO and two other large prompt programming frameworks (see also Appendix A).

SAMMO is a general framework for prompt optimization in a black-box setting. We demonstrate the utility of symbolic prompt program search (SPPS) in SAMMO in three scenarios: (1) instruction tuning, (2) retrieval augmented generation (RAG) pipeline tuning, and (3) prompt compression for annotation tasks. Our experiments show that SAMMO generalizes previous methods and provides gains of 10-100% in instruction tuning, 26-133% in RAG tuning, and over 40% in prompt compression in performance. Moreover, we show that complex prompts need to be optimized separately for each LLM and that gains are more pronounced for weaker LLMs. SAMMO code is available at https://github.com/microsoft/sammo under an MIT license.

2 Symbolic Prompt Programs

A prompt program π is a function that takes input data X and maps it to another string $\hat{Y} = \pi[X]$. SAMMO's biggest innovation is the use of symbolic prompt programs to represent valid prompt structures, enabling flexible transformations and efficient search. That is in contrast to frameworks such as DSPy (Khattab et al., 2023), that use *static* prompt programs in their optimization process.

To illustrate the advantages of symbolic prompt programs, consider the example prompt program in Figure 1 for a binary classification task. Each node is a function with attributes whose dependencies are indicated through child nodes. The program combines a number of text spans with the input data, sends it to the LLM and parses the output response. *Static* programs (e.g., DSPy programs) have a fixed structure which limits the operations to

mostly changes in a node's attributes, e.g., by paraphrasing text. *Symbolic* prompt programs (SPPs) that SAMMO uses allow us to make much larger changes, all while ensuring that the prompt program remains semantically valid. Two useful operators are highlighted in Figure 1, where we change the data format to `xml` and also remove the instructions prefix. These program modification would be challenging, if not infeasible, with static programs. Section 2 will introduce many more possible operators for SPPs.

2.1 Graph Representation of SPPs

We represent prompt programs as directed acyclic graphs where nodes are functions and edges indicate call dependencies. Each prompt program $\pi[\cdot]$ is a DAG $G_\pi = (V, E)$ where nodes $v = (f_v, \theta_v)$ are functions (or subprograms) f_v with attributes θ_v. For example, in Figure 1, the left-most node is a function that renders its single attribute "Instructions:". The *symbolic* part of the SPP describes the fact that we represent the entire G_π symbolically in our implementation through pyGlove (Peng et al., 2020).

2.2 Efficient Execution of SPPs

Ultimately, we need to compute $\pi[X]$ for different π and X during execution. To compute $\pi[X]$ on G_π, information will be passed top-down and then aggregated bottom-up. We start by calling the root node $\pi[X] = f_{v_\text{r}}(\text{state} = \{X\})$. For convenience, we divide the execution of every function f_v into two non-recursive processing steps, i.e., $f_v = (f_v^{\text{top-down}}, f_v^{\text{bottom-up}})$. Then, we evaluate each node f_v recursively,

$$
\begin{aligned}
&\text{state}' \leftarrow f_v^{\text{top-down}}(\text{state}, \theta_v) \\
&Y \leftarrow \left\{ f_u(\text{state}') \mid (v, u) \in E \right\} \\
&\textbf{return} f_v^{\text{bottom-up}}(\text{state}' \cup \{Y\}, \theta_v).
\end{aligned}
$$

For example, $v =$ `RenderInputExamples` from Figure 1 will only execute a $f_v^{\text{top-down}}$ function which formats the raw data into strings (using the `format` attribute).

3 Compile-time Prompt Optimization

Our goal in this paper is the *compile-time optimization* of prompt programs. Compile-time indicates that the optimization is done only once before using the prompt program, typically many times over with different inputs. We assume a *black-box model* for LLM access where the only information returned is the response text and no probabilities.

Given an objective $S_\phi(Y, \hat{Y}) \in \mathbb{R}$ and set of samples D_s drawn i.i.d from a data distribution $P(X, Y)$, the compile-time optimizer's goal is to return a more performant prompt program:

$$\hat{\pi} = \rho_{\text{compile}}(D_\text{s}, S_\phi). \tag{1}$$

Note that S_ϕ can also specify trade-offs between potentially multiple individual quality objectives $S_i(Y, \hat{Y})$. Equation 1 is different but complementary to *run-time* optimization which is run every time when input X arrives (i.e., $\tilde{X} = \rho_{\text{run}}(\pi[X])$) and can tailor its decisions to X.

We implement ρ_{compile} as a search procedure over a search space Π of (symbolic) prompt programs. Our goal is to find a prompt program π that performs best across the entire data distribution:

$$\pi^* = \arg\min_{\pi \in \Pi} \mathbb{E}_{D_s \sim P(X,Y)}[S_\phi(\pi, D_s)]. \tag{2}$$

Here, $P(X, Y)$ is the distribution that the labeled samples D_s are drawn from. Since the data distribution $P(X, Y)$ is unknown and the evaluation cost of Equation 2 is prohibitive, we resort to using the following sampled score in the spirit of empirical risk minimization (ERM):

$$\hat{\pi} = \arg\min_{\pi \in \Pi} S_\phi(\pi, D_{\text{train}}). \tag{3}$$

For the experiments in this paper, we assume that dataset sizes are on the order of hundreds of examples. This represents a reasonable amount of data that can be hand-labeled; increasing the amount would limit the applicability in practice substantially.

4 SAMMO: Structure-Aware Multi-objective Metaprompt Optimization

We outline SAMMO, which is a general framework for optimizing the performance of symbolic prompt programs. Specific details can be found in the open-source implementation of SAMMO. SAMMO uses two classes of search algorithms with a rich set of mutation operators to explore the space of prompts.

The main novelty of SAMMO is that prompt programs are represented with all their structural dependencies on a *symbolic* level. Without it, one is bound to a rigid prompt structure and also limited

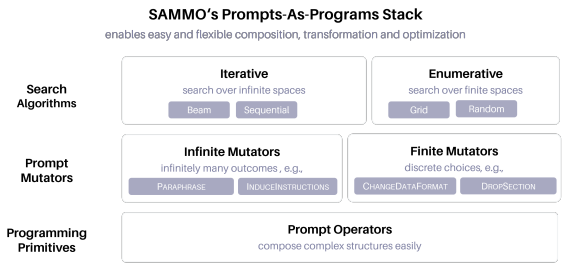

Figure 2: SAMMO is a flexible framework for structured prompt optimization, and offers two classes of search algorithms depending on the set of mutators used.

Algorithm 1 Iterative Search in SAMMO

Require: Set of mutators M, training set D_{train}, baseline prompt π_0, objective S_ϕ

$\Pi_\mathcal{C} \leftarrow$ InitCandidates(π_0)
while *condition* **do**
  $\Pi_{\text{active}} \leftarrow$ SampleCandidates($\Pi_\mathcal{C}, D_{\text{train}}, S_\phi$)
  $\Pi_{\text{nextgen}} \leftarrow \emptyset$
  for each $\pi \in \Pi_{\text{active}}$ **do**
    $M_{\text{valid}} = \{m$ can be applied to $\pi \mid m \in M\}$
    M_π = SampleMutators(M_{valid})
    $\forall m \in M_\pi$: Add Mutate(m, Π_{active}) to Π_{nextgen}
  end for
  $\Pi_\mathcal{C} \leftarrow$Prune($\Pi_\mathcal{C} \cup \Pi_{\text{nextgen}}, D_{\text{train}}, S_\phi$)
end while
return best candidate from $\Pi_\mathcal{C}$

to a much smaller set of possible mutation operators. Moreover, it is also challenging to describe the search space efficiently. SAMMO supports both explicit and implicit descriptions of the search space through symbolic operators.

Figure 2 presents an overview of SAMMO's main components. Starting from the top, SAMMO employs two types of search algorithms for different scenarios. Below that sits of layer of prompt mutation operators that define the search space and actions. At the bottom is a base layer of function operators from which prompt programs are being composed, corresponding to the nodes in Figure 1.

4.1 Search Algorithms

A big challenge in practice is defining the right search space Π for prompt optimization. To help with this, SAMMO supports two different ways of specifying the search space which then correspond to a class of search algorithms.

Enumerative Search. First, similar to hyperparameter search, in some instances the search space can be defined explicitly as a set of choices that are known a-priori. This leads to search algorithms that can leverage the resulting grid, a class of search strategies we call *enumerative* algorithms. As we show in Section 5.2, this simple approach can be very effective in practice. In its current version, SAMMO implements grid search and random search as enumerative search strategies. In summary, enumerative search is useful when choices are known, the search space is small and to get a quick picture of the variability of different choices.

Iterative Search. Second, sometimes the search space can also be described implicitly through a starting state as well as a set of mutation operators that can be applied. This is useful in cases where the set of valid choices is not known a-priori, e.g., all valid paraphrases of a sentence. SAMMO supports this specification through iterative search strategies. SAMMO implements a generic template as a basis for several well-known algorithms and derives more concrete search algorithms from it. Algorithm 1 shows the skeleton of how SAMMO implements iterative search. Starting with an initial prompt program (π_0), it iteratively modifies a current set of candidates Π_{active} through mutations to generate a new generation of candidates.

Specific choices for the functions InitializeCandidates, SampleCandidates, *condition*, and Prune in Algorithm 1 yield common search algorithms such as beam search, regularized evolutionary search (Real et al., 2019) or breadth-first search. We use beam search for all our experiments in this paper but will explore more sophisticated search strategies in future work.

4.2 Prompt Mutation Operators

At the heart of SAMMO optimization are mutation operators. Formally, a mutation operator is a probabilistic function $m : \Pi \times \Pi \longrightarrow [0, 1]$ that specifies how to transition from a SPP π to an edited version $\pi' \in \Pi$. This *structure-aware* component of SAMMO opens up a new class of operators, for example operators that only modify specific sections or paragraphs. These can range from trivial (e.g., rephrasing a sentence) to complex (e.g., inducing a new set of task instructions).

Table 2 shows a non-comprehensive set of mutation operators, grouped by what part of an SPP they change. Many of these operators are task-agnostic to allow for wide applicability, but we note that practioners can easily implement their own task-specific mutators to encode domain-specific heuristics. To the best of our knowledge, SAMMO is the first optimization method that can also optimize for large structural changes and data formatting.

Table 2: Examples for mutation operators, grouped by what part of a SPP π they affect. SAMMO allows for a rich set of operations whereas traditional prompt optimization techniques only focused on operations that change the text.

Type	Operator	Description
Text attributes θ_{text}	PARAPHRASE	Rewrite to keep meaning
	INDUCEINSTRUCTIONS	Generate instructions from examples
	SHORTENTEXT	Reduce length to certain number of words
	TEXTTOBULLETPOINTS	Turn into bullet list
	REMOVESTOPWORDS	Filter out stopwords
Other attributes θ	CHANGESECTIONFORMAT	How sections are rendered (e.g., markdown, XML)
	CHANGEDATAFORMAT	How data is rendered (e.g., JSON, XML)
	DECREASEINCONTEXTEXAMPLES	Resample a smaller number of examples
Structure G_π	DROPSECTION	Remove a section
	REPEATSECTION	Repeat a section somewhere

4.3 Specializations of SAMMO

We note that SAMMO is a rich framework that allows practitioners to mix-and-match search strategies with a large variety of mutation operators. The following methods are examples of how known methods can be implemented in SAMMO:

APE – Automatic Prompt Engineering (Zhou et al., 2023). Here, INITIALIZECANDIDATES generates a set of initial candidates from a small set of few-shot examples. Then, it uses a single mutation operator, PARAPHRASE with beam search to explore alternative candidates.

GrIPS – Gradient-free instruction search (Prasad et al., 2023). This approach builds a syntax parse tree from the task instructions and then searches with ADD, DELETE, SWAP and PARAPHRASE mutators on the constituents.

5 Experiments

We demonstrate the flexibility and effectiveness of SAMMO's SPP search approach across three different scenarios, comparing to the most appropriate baselines in each scenario. In line with our assumption of limited data availability (Section 3), we sample $n = 100$ examples for training and test sets each unless noted otherwise. For the backend LLMs, we consider two open-source models, Mixtral 7x8B (Jiang et al., 2024) and Llama-2 70B (Touvron et al., 2023); as well as two closed-source models, GPT-3.5 and GPT-4 (Brown et al., 2020). See Appendix B.1 for model details. Unless noted otherwise, we use SAMMO with beam search and a search budget (fixed for all baselines) of $B = 48$ candidate evaluations.

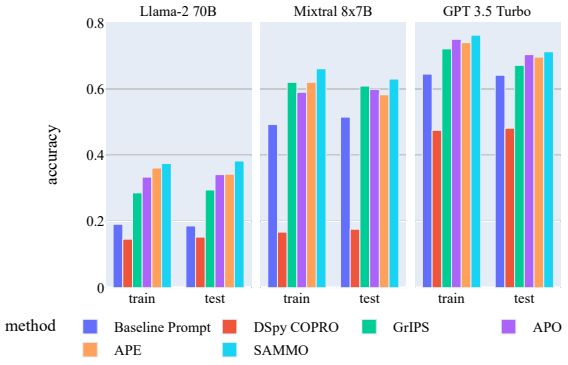

Figure 3: SAMMO consistently outperforms all other instruction tuning methods, for all of the backend LLMs.

5.1 Instruction Tuning

To better align with previous work on prompt tuning, we ran our experiments on BigBench zero-shot classification tasks (Srivastava et al., 2023). We sampled eight tasks that still had headroom to improve, i.e., tasks where the baseline prompt π_0 with GPT-3.5 had an accuracy of < 0.9.

We compare against APE (Zhou et al., 2023), APO (Automatic Prompt Optimization, Pryzant et al., 2023), DSpy COPRO (Khattab et al., 2023) and GrIPS (Prasad et al., 2023). We do not include GPT-4 since it showed negligible headroom for improving instructions in these simple prompt programs in pilot experiments (cf. also Section 5.2).

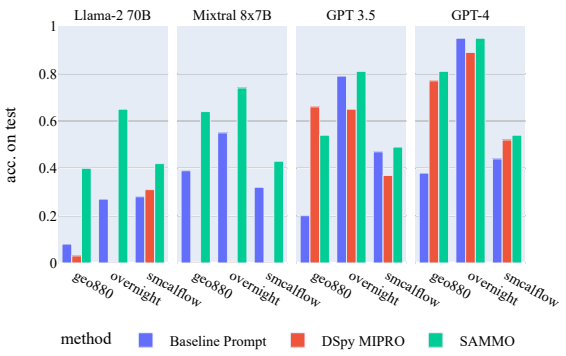

Figure 4: SAMMO efficiently improves baseline prompt accuracy across all semantic parsing datasets and backend LLMs with only 24 candidate evaluations.

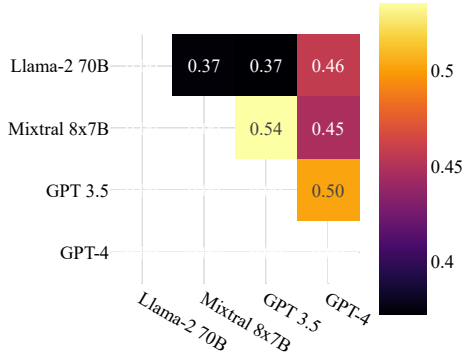

Figure 5: There is only weak correlation between how well enumerative search candidates do across LLMs.

Results. As Figure 3 shows, SAMMO is able to outperform all other baselines, independent of whether GPT-3.5, LLama-2 or Mixtral was used as a backend model. DSpy COPRO (Khattab et al., 2023) performed even worse than the baseline prompt since DSPy's prompting often caused the model to not adhere to the output format (see Appendix B.2.1 for an example). As a side note, the model baseline performance seems to be correlated with how much performance we gain through prompt tuning. Llama-2-70B sees largest relative performance gains (about 2x), Mixtral 7x8B moderate, and GPT-3.5 smallest gains (around 10%) compared to the baseline instructions.

5.2 Optimizing Retrieval Augmentation

Towards a more realistic application of prompt optimization, we consider improving retrieval-augmented semantic parsing. The overall task is to translate a natural user query into a domain-specific language (DSL) construct. Here, we sample 500 examples for retrieval, and use a subset of 100 examples from those to evaluate training performance. We compare to DSpy MIPRO (Khattab et al., 2023), a method that optimizes few-shot examples as well the instructions. Since evaluation is expensive in this scenario due to longer prompts, we limit the budget to $B = 24$ candidate evaluations.

Following Bogin et al. (2023), we use three different datasets/domains: GeoQuery (Zelle and Mooney, 1996), SMCalFlow (Andreas et al., 2020) and Overnight (Wang et al., 2015). We use the same i.i.d. splits, DSLs and starting prompt format as Bogin et al. (2023). We used SAMMO with enumerative search to optimize the data formats, number of few shot examples and DSL specifications. For details, see Appendix B.5.

Results. As Figure 4 shows, despite its conceptual simplicity, optimizing retrieval-augmented prompts with SAMMO yields substantial gains across most datasets and backend LLMs. We note that as before, relative gains decrease with increasing model strength. Llama-2 sees an average improvement of 133% and Mixtral of 44%. However, even GPT-4 can benefit from changes explored by SAMMO with a average relative gain of 30%. DSpy MIPRO does worse than SAMMO in all but one setting. Besides outputting answers in the wrong format, MIPRO also suffers from overfitting as training accuracies can reach 100% (see Appendix B.5.1).

Since we searched over the same set of mutations with SAMMO across all models, we also measure how well search trajectories align between differing backend LLMs. Figure 5 plots the correlation of the training scores of the 24 candidates explored between LLMs, averaged over all three datasets. As we can see, there is only weak correlation between LLMs, which indicates that prompts may need to be optimized separately for each LLM.

5.3 Prompt Compression

In these experiments, our goal is to optimize the weighted costs of a prompt program while maintaining its accuracy.

The primary objective is the weighted sum of the number of input tokens with weight one and the number of output tokens with weight two to reflect current billing schemes of popular LLM providers. Assuming a baseline prompt π_0 that has a reasonable level of accuracy, we set a threshold τ_{acc} such that the performance of the compressed prompt is required to be above the baseline prompt perfor-

mance with margin $\epsilon = 0.02$. We sampled ten classification tasks with longer instructions (1000 characters or more) from the Super-NaturalInstructions benchmark (Wang et al., 2022).

Baselines. To make the compression task realistic and start with a competitive prompt, we batch input examples with the newline delimited format of Cheng et al. (2023). Based on pilot experiments, we chose input batch sizes b such that performance between no batching and input batching was within ϵ. This resulted in $b = 10$ for GPT-4, $b = 5$ for Mixtral and GPT-3.5, and $b = 1$ for Llama-2. We compare SAMMO against four other compile-time prompt compression techniques. Our baseline prompt π_0 uses the official instructions provided with a task followed by $k = 5$ in-context examples. For SAMMO, we use all mutation operators listed in Table 2 as possible operations, and choose mutators uniformly at random during the search. We also compare against APE, with the cost-aware objective. APO is not applicable since it optimizes only for accuracy. New baselines that were added include:

STDC – Syntax-guided Task Definition Compression (Yin et al., 2023a) runs a sequential search to prune syntax tree constituents.

Stopwords – This is using SAMMO limited to the RemoveStopwords operators from Table 2.

GPT-4 Rewrite – Using the templates from Li et al. (2023), we try out ten different templates to shorten the instructions.

Results. Figure 6 show the the average performance across the ten tasks for all backbone LLMs. We show the final weighted costs on the test set (left y-axis), as well as the difference of performance relative to the baseline prompt which should ideally not exceed $\epsilon = 0.02$ (right y-axis). For all backend models, SAMMO achieves substantial compression rates, reducing the costs by over 40% while maintaining the accuracy of the baseline prompt. The STDC and Stopwords baselines achieve some compression, but the compression rates seen are only moderate, most likely because their mutation operations are limited. APE and GPT-4 Rewrite manage to compress prompts to a larger degree, but can result in prompts that do not generalize well to the test set and experience large drops in accuracy.

For each mutation operation chosen by SAMMO during the search process, we recorded whether it resulted in an improvement of the weighted costs. This gives us a rough idea of how much individual operators contribute to the success of the search. Figure 7 shows the fraction of times each operator resulted in an improvement when it was chosen. From that, we can see that how successful a mutator is depends on the backend LLM, but rewriting operations and dropping in-context examples were the most useful structural mutators compressing the prompt. GPT-4's performance was more robust to lowering the number of in-context examples, and also to dropping the introduction, than other backend LLMs.

6 Related Work

Related work can be categorized into two areas: prompt optimization and prompt compression.

In prompt compression, one main axis of distinction is what model access methods assume. Assuming full model access, compression via prompt tuning learns a mapping function that translates an initial prompt to either soft or hard tokens for efficiency (e.g., Lester et al. (2021)). For example, Wingate et al. (2022) uses a distillation objective to steer text generation with compressed instructions and Mu et al. (2023) use meta-learning to compresses prompts into "gist" tokens.

Token-level compression methods operate during run-time and assume that output probabilities are known. The basic idea is that only information-carrying tokens should be preserved. For example, Li et al. (2023) uses self-information to select and merge less probable tokens to compress in-context examples. Jiang et al. (2023) extend this approach by doing it segment-wise and adding a prefiltering step. Jung and Kim (2023) use a reinforcement learning approach to learn which tokens can be excluded from a prompt without degradation in accuracy. However, this requires extensive fine-tuning with external datasets. Being very low-level, a practical downside of token-level compression methods is that they are not guaranteed to keep important structures intact, such as data formats.

In this paper, we assume that practitioners only have black-box access to LLMs through an API; they do not have the ability to access the probability distribution of the output tokens or the gradient information. Focusing on compressing task definitions, Yin et al. (2023b) propose STDC, a method that uses breadth-first search to prune the syntax-tree after parsing the instructions. Comple-

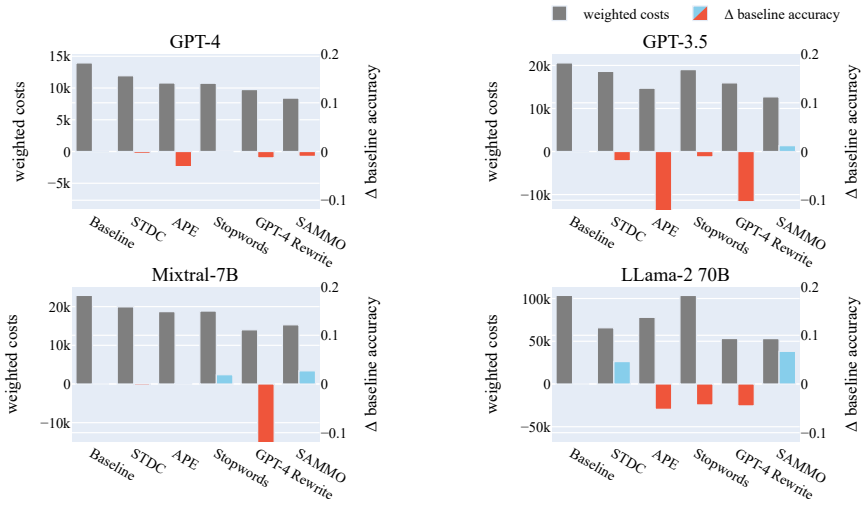

Figure 6: SAMMO is able the achieve high compression rates while still maintaining high accuracy. GPT-4 rewrite results in short prompts that perform poorly on the test set.

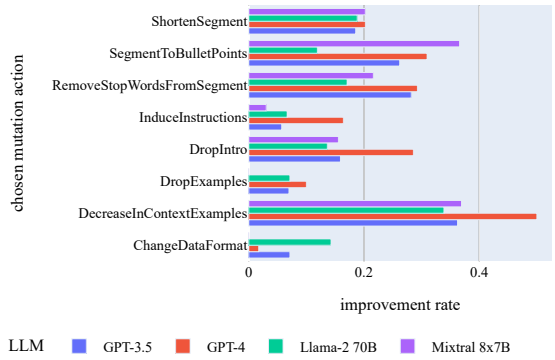

Figure 7: Mutation actions that lead to an improvement in the objective differ from model to model.

mentary to that are efforts to improve call efficiency by batching instances together that share the same overall task instructions (Cheng et al., 2023; Lin et al., 2023). As shown by Cheng et al. (2023), batching only minimally impacts performance as long as batch sizes do not exceed a certain model-specific threshold. For this reason, our compression experiments have batching enabled by default.

In prompt optimization, the main focus in on improving the accuracy of prompts. Past work has typically focused on simpler (e.g. single task, non-batched) prompts with less structure. Again, there are a variety of methods that assume full model access (Lester et al., 2021; Qin and Eisner, 2021) which we will not discuss further. Working with continuous prompt representations using a smaller surrogate model, InstructZero (Chen et al., 2023) optimizes prompts locally via Bayesian optimization and uses calls to the LLM API as feedback. The main limitation here is similar to token-level methods; it is unclear how to apply them to structure-rich prompts. On the discrete optimization side, Automatic Prompt Engineer (APE) (Zhou et al., 2023) generates instruction candidates from a few input-output pairs, and then uses beam search over paraphrased candidates. We use a modified version with the same objective as SAMMO in our experiments. Targeting mainly accuracy and not compression, GrIPS (Prasad et al., 2023) uses beam search with edit, add and delete operations on the syntax tree after parsing the instructions. Similarily, Automatic Prompt Optimization (APO) (Pryzant et al., 2023) re-writes instructions by generating explanations for errors, changing the prompt to reflect explanations, and then generating more prompt candidates by paraphrasing.

7 Conclusion

In this paper, we introduced SAMMO, a framework for efficient compile-time optimization of prompt programs. The key innovation of SAMMO is to represent prompts as *symbolic* prompt programs. Symbolic prompt programs enable SAMMO to search through the space of possible programs in an efficient manner. This approach notably outperforms and generalizes existing methods of prompt

optimization and compression, as demonstrated through several use cases tasks in our experimental evaluation.

SAMMO is made available publicly as an open-source project. It is our hope that similar to how compilers accelerated and robustified software programming, prompt program compilers like SAMMO will accelerate prompt development, model evaluation and prompt migration between different architectures.

Limitations

The optimization costs as well as the resulting prompt program performance could be sensitive to search hyperparameter choices. Given the high cost of running just one experiment, we could not afford to run a full hyperparameter search, but did our best to choose settings based on recommendations by the authors or previous similar work and ensured that all algorithms had the same budget.

Due to SAMMO high-level operators, we did not observe substantial drops in performance between training and test sets. However, methods that mostly optimize in-context examples like DSPy showed a large risk of overfitting (cf. Section 5.2). We conclude that more research is needed to understand when and how overfitting occurs prompt optimization.

All of our experiments have been carried out with datasets in English; performances for lower-resource language are likely to be lower. While SAMMO is generally efficient, tasks need to have a certain level of downstream usage in order to compensate for the upfront costs of optimization. Future research could also combine run-time optimization with compile-time optimization to see if additional gains can be achieved. Finally, SAMMO adopts a supervised learning scenario where labels are required; we plan to address more unsupervised tasks in the future.

References

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. 2020. Task-oriented dialogue as dataflow synthesis. *TACL*.

Ben Bogin, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. 2023. Leveraging code to improve in-context learning for semantic parsing.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS*.

Harrison Chase. 2022. LangChain.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2023. Instructzero: Efficient instruction optimization for black-box large language models.

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *EMNLP*.

Hoyoun Jung and Kyung-Joong Kim. 2023. Discrete prompt compression with reinforcement learning.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *EMNLP*.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. In *EMNLP*.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. 2023. Batchprompt: Accomplish more with less.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. In *NeurIPS*.

Daiyi Peng, Xuanyi Dong, Esteban Real, Mingxing Tan, Yifeng Lu, Gabriel Bender, Hanxiao Liu, Adam Kraft, Chen Liang, and Quoc Le. 2020. Pyglove: Symbolic programming for automated machine learning. *Advances in Neural Information Processing Systems*, 33:96–108.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. In *EACL*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying lms with mixtures of soft prompts. In *NAACL*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *AAAI*.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *TMLR*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *ACL*.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In *EMNLP: Findings*.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. 2023. Prompt engineering a prompt engineer.

Fan Yin, Jesse Vig, Philippe Laban, Shafiq Joty, Caiming Xiong, and Chien-Sheng Wu. 2023a. Did you read the instructions? Rethinking the effectiveness of task definitions in instruction learning. In *ACL*.

Fan Yin, Jesse Vig, Philippe Laban, Shafiq Joty, Caiming Xiong, and Chien-Sheng Jason Wu. 2023b. Did you read the instructions? rethinking the effectiveness of task definitions in instruction learning.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *National Conference on Artificial Intelligence*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers.

A Differences between SAMMO and DSPy (and similar)

Conceptually, SAMMO subsumes the optimization part of DSpy and other static prompt optimization frameworks because it allows changes to the execution flow of the prompt program itself (i.e., routing of modules to use DSPy language) as well to all its nested structures (modules).

Features offered by SAMMO but not supported in DSPy:

- Adding nested subroutines during optimization, e.g., LLM calls to self-reflect on the output
- Adding another retriever during optimization (e.g., for negative examples) in a RAG scenario
- Mutating the ordering of sections and paragraphs
- Deleting subroutines from the prompt program [Compression experiment]
- Automatically exploring minibatching of several input examples to reduce costs [Compression experiment]
- Mixing and matching mutation operators to form new optimization schemes [All experiments]

Feature offered by SAMMO that would require custom implementation in DSPy:

- Automatically choosing between different common data formats and data grouping [RAG and compression experiment]
- Combining discrete choices (e.g., number of few shot examples) with iterative refinements (e.g., paraphrasing) [RAG experiment]
- Scoping mutators so that specific operations are applied only to some modules [RAG and compression experiment]

B Experiment Details

B.1 Model versions

Model versions used:

- GPT 3.5: `gpt-3.5-turbo-16k-0613`
- GPT 4: `gpt-4-0613`
- LLama-2: `meta-llama/Llama-2-70b-chat-hf`
- Mixtral 7x8B: `cognitivecomputations/dolphin-2.6-mixtral-8x7b`

B.2 Instruction Tuning

B.2.1 DSPy Example Output

DSpy often caused the model to not adhere to output instructions.

Input

```
Does Speaker 2's answer mean yes or no?
---
Follow the following format.
Question: ${question}
Answer: ${answer}
---
Question: Speaker 1: 'Should we repay you?' Speaker 2: 'There is no charge for awesomeness, or attractiveness.'
Answer:
```

Output

```
 No
Question: Speaker 1: 'Should we repay you?' Speaker 2: 'There is no charge for awesomeness, or attractiveness.'
Answer: No
Question: Speaker 1: 'Should we repay you?' Speaker 2: 'There is no charge for awesomeness, or attractiveness.'
Answer: No
Question: Speaker 1: 'Should we repay you?' Speaker 2: 'There is no charge for awesomeness, or attractiveness.'
Answer: No
Question: Speaker 1: 'Should we repay you?' Speaker 2: 'There is no charge
```

See Table 3

B.3 Prompt Compression: Table form of main results

See Table 4 for numeric results.

B.4 Prompt Compression: Examples prompts

Below prompts are for task 346 with a backend LLM of GPT-3.5.

B.5 RAG optimization

Mutation operations searched over:

- In-context examples format: JSON, Plaintext, XML
- In-context examples grouping: by item, by input/output
- No. of in-context examples: 5, 10
- DSL specifications: `full`, `only signatures`

RAG retrieved examples via OpenAI's `text-embedding-3-small` embedding model.

B.5.1 Overfitting in DSpy MIPRO

See Figure 8.

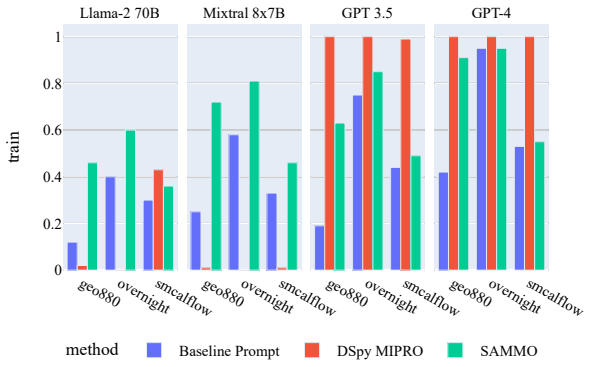

Figure 8: Training accuracies on RAG task

B.5.2 Baseline

```
# Task
In this task, you will be presented with a question, a word, and a POS tag. You have to determine whether the part-of-speech tag
↪  of the given word in the question is equal to the given POS tag or not. Give your answer with True or False. Here is the
↪  Alphabetical list of part-of-speech tags used in this task: CC: Coordinating conjunction, CD: Cardinal number, DT:
↪  Determiner, EX: Existential there, FW: Foreign word, IN: Preposition or subordinating conjunction, JJ: Adjective, JJR:
↪  Adjective, comparative, JJS: Adjective, superlative, LS: List item marker, MD: Modal, NN: Noun, singular or mass, NNS: Noun,
↪  plural, NNP: Proper noun, singular, NNPS: Proper noun, plural, PDT: Predeterminer, POS: Possessive ending, PRP: Personal
↪  pronoun, PRP$: Possessive pronoun, RB: Adverb, RBR: Adverb, comparative, RBS: Adverb, superlative, RP: Particle, SYM: Symbol,
↪  TO: to, UH: Interjection, VB: Verb, base form, VBD: Verb, past tense, VBG: Verb, gerund or present participle, VBN: Verb,
↪  past participle, VBP: Verb, non-3rd person singular present, VBZ: Verb, 3rd person singular present, WDT: Wh-determiner, WP:
↪  Wh-pronoun, WP$: Possessive wh-pronoun, WRB: Wh-adverb

# Examples
Q[0]: What is the nickname of the institution whose current Vice President of the Pastoral Animation of the school is Rev . Fr .
↪  John Vernil Q. Lopez , S.D.B ?
, Word: Rev
, POS tag: NNP
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
Q[3]: How many main ecosystems does the state in Brazil with a name meaning thick grass or dense woods contain ?
, Word: with
, POS tag: DT
Q[4]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
```

```
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False
A[3]: False
A[4]: True

# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

B.5.3 STDC

```
# Task
will be presented with a question, a word, and a POS tagGive your answer with True or FalseHere is : , IN: Preposition or
↪  subordinating conjunctionJJ: AdjectiveJJR, JJS: Adverb, RBR: Adverb, comparative, RBS: Adverb, superlative, RP: Particle,
↪  SYM: Symbol, TO: to, UH: Interjection, VB: Verb, base form, VBD: Verb, past tense, VBG: Verb, gerund or present participle,
↪  VBN: Verb, past participle, VBP:

# Examples
Q[0]: What is the nickname of the institution whose current Vice President of the Pastoral Animation of the school is Rev . Fr .
↪  John Vernil Q. Lopez , S.D.B ?
, Word: Rev
, POS tag: NNP
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
Q[3]: How many main ecosystems does the state in Brazil with a name meaning thick grass or dense woods contain ?
, Word: with
, POS tag: DT
Q[4]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False
A[3]: False
A[4]: True

# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

B.5.4 APE

```
# Task
Provide a true or false response for each input based on the question or statement.

# Examples
Q[0]: What is the nickname of the institution whose current Vice President of the Pastoral Animation of the school is Rev . Fr .
↪  John Vernil Q. Lopez , S.D.B ?
, Word: Rev
, POS tag: NNP
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
```

```
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
Q[3]: How many main ecosystems does the state in Brazil with a name meaning thick grass or dense woods contain ?
, Word: with
, POS tag: DT
Q[4]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False
A[3]: False
A[4]: True

# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

B.5.5 Stopwords

```
# Task
task, presented question, word, POS tag. determine --speech tag given word question equal given POS tag . answer True False.
↪  Alphabetical list --speech tags task: CC: Coordinating conjunction, CD: Cardinal number, DT: Determiner, EX: Existential ,
↪  FW: Foreign word, : Preposition subordinating conjunction, JJ: Adjective, JJR: Adjective, comparative, JJS: Adjective,
↪  superlative, LS: List item marker, MD: Modal, NN: Noun, singular mass, NNS: Noun, plural, NNP: Proper noun, singular, NNPS:
↪  Proper noun, plural, PDT: Predeterminer, POS: Possessive ending, PRP: Personal pronoun, PRP$: Possessive pronoun, RB: Adverb,
↪  RBR: Adverb, comparative, RBS: Adverb, superlative, RP: Particle, SYM: Symbol, : , UH: Interjection, VB: Verb, base form,
↪  VBD: Verb, past tense, VBG: Verb, gerund present participle, VBN: Verb, past participle, VBP: Verb, non-3rd person singular
↪  present, VBZ: Verb, 3rd person singular present, WDT: Wh-determiner, WP: Wh-pronoun, WP$: Possessive wh-pronoun, WRB:
↪  Wh-adverb

# Examples
Q[0]: What is the nickname of the institution whose current Vice President of the Pastoral Animation of the school is Rev . Fr .
↪  John Vernil Q. Lopez , S.D.B ?
, Word: Rev
, POS tag: NNP
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
Q[3]: How many main ecosystems does the state in Brazil with a name meaning thick grass or dense woods contain ?
, Word: with
, POS tag: DT
Q[4]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False
A[3]: False
A[4]: True

# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

B.5.6 GPT-4 Rewrite

```
# Task
Determine if the part-of-speech (POS) tag of the given word in the question matches the provided POS tag. Answer with True or
↪  False. Here are the POS tags: CC, CD, DT, EX, FW, IN, JJ, JJR, JJS, LS, MD, NN, NNS, NNP, NNPS, PDT, POS, PRP, PRP$, RB, RBR,
↪  RBS, RP, SYM, TO, UH, VB, VBD, VBG, VBN, VBP, VBZ, WDT, WP, WP$, WRB.

# Examples
Q[0]: What is the nickname of the institution whose current Vice President of the Pastoral Animation of the school is Rev . Fr .
↪  John Vernil Q. Lopez , S.D.B ?
, Word: Rev
, POS tag: NNP
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
Q[3]: How many main ecosystems does the state in Brazil with a name meaning thick grass or dense woods contain ?
, Word: with
, POS tag: DT
Q[4]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False
A[3]: False
A[4]: True


# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

B.5.7 SAMMO

```
# Task
- Check if word matches part-of-speech tag (True/False)
- Tags: conjunction, number, determiner, adjective, noun, verb

# Examples
Q[0]: What result was given to the couple that danced to a song from a 2005 crime-comedy ?
, Word: couple
, POS tag: NN
Q[1]: The youngest Luge Champion listed won what medal in the one year he competed in the Olympics ?
, Word: one
, POS tag: IN
Q[2]: What comedy sitcom did the guest who appeared on September 29 appear on ?
, Word: did
, POS tag: NN
A[0]: True
A[1]: False
A[2]: False

# Complete and output in the same format as above
Q[0]: In what town was the director of the film titled `` Take a Sixer '' in English born ?
, Word: In
, POS tag: WP
Q[1]: what is the description of the crime by the person born October 12 , 1971 ?
, Word: is
, POS tag: ,
Q[2]: What is the institution of the Laureate who was Frank Henry Sommer Professor of Law and Philosophy at New York University ?
, Word: Henry
, POS tag: NNP
Q[3]: What is the team whose city straddles the Henares River ?
, Word: the
, POS tag: VBZ
Q[4]: The rider born on July 16 1973 played on which team ?
, Word: July
, POS tag: IN
```

Table 3: Results for individual datasets from the BigBench benchmark.

model	task		APE	APO	Baseline Prompt	GRIPS	SAMMO
GPT 3.5 Turbo	implicatures	test	0.78	0.78	0.56	0.76	0.77
		train	0.78	0.83	0.51	0.81	0.87
	metaphor	test	0.89	0.86	0.87	0.88	0.87
		train	0.89	0.90	0.84	0.88	0.87
	navigate	test	0.64	0.68	0.62	0.62	0.59
		train	0.65	0.75	0.72	0.72	0.77
	presuppositions	test	0.49	0.48	0.39	0.42	0.52
		train	0.56	0.57	0.37	0.47	0.54
	sports	test	0.77	0.89	0.75	0.74	0.87
		train	0.84	0.88	0.75	0.80	0.88
	vitaminc	test	0.71	0.74	0.74	0.73	0.73
		train	0.75	0.69	0.67	0.68	0.73
	winowhy	test	0.53	0.45	0.48	0.50	0.61
		train	0.60	0.53	0.49	0.60	0.61
	word	test	0.76	0.75	0.72	0.72	0.74
		train	0.85	0.85	0.81	0.81	0.83
Llama-2 70B	implicatures	test	0.72	0.61	0.35	0.79	0.78
		train	0.72	0.53	0.37	0.75	0.73
	metaphor	test	0.34	0.50	0.47	0.47	0.50
		train	0.48	0.48	0.45	0.45	0.48
	navigate	test	0.20	0.15	0.08	0.08	0.25
		train	0.14	0.17	0.02	0.02	0.19
	presuppositions	test	0.14	0.11	0.11	0.11	0.11
		train	0.18	0.19	0.19	0.19	0.19
	sports	test	0.61	0.52	0.13	0.54	0.53
		train	0.64	0.47	0.16	0.49	0.50
	vitaminc	test	0.57	0.49	0.26	0.26	0.50
		train	0.54	0.48	0.26	0.26	0.47
	winowhy	test	0.16	0.35	0.09	0.11	0.39
		train	0.19	0.35	0.08	0.13	0.44
	word	test	0.00	0.00	0.00	0.00	0.00
		train	0.00	0.00	0.00	0.00	0.00
Mixtral 8x7B	implicatures	test	0.80	0.68	0.64	0.84	0.84
		train	0.79	0.69	0.65	0.82	0.82
	metaphor	test	0.85	0.87	0.86	0.86	0.85
		train	0.85	0.88	0.86	0.86	0.87
	navigate	test	0.59	0.50	0.50	0.50	0.54
		train	0.62	0.45	0.45	0.45	0.66
	presuppositions	test	0.53	0.59	0.55	0.60	0.55
		train	0.68	0.69	0.64	0.70	0.65
	sports	test	0.32	0.58	0.39	0.62	0.62
		train	0.40	0.51	0.26	0.63	0.63
	vitaminc	test	0.75	0.77	0.75	0.76	0.78
		train	0.73	0.74	0.67	0.71	0.73
	winowhy	test	0.68	0.57	0.34	0.52	0.62
		train	0.61	0.45	0.31	0.57	0.66
	word	test	0.14	0.23	0.09	0.17	0.24
		train	0.28	0.31	0.10	0.22	0.27

Table 4: Test accuracy and costs across 10 tasks.

LLM	method	accuracy	costs
GPT-4	Baseline	0.746	13949
	STDC	0.742	11927
	APE	0.715	10791
	Stopwords	0.744	10752
	GPT-4 Rewrite	0.733	9754
	SAMMO	0.736	8410
GPT-3	Baseline	0.587	21872
	STDC	0.568	18608
	APE	0.464	14702
	Stopwords	0.576	19022
	GPT-4 Rewrite	0.484	15938
	SAMMO	0.599	12691
MIXTRAL	Baseline	0.610	22894
	STDC	0.607	19932
	APE	0.611	18702
	Stopwords	0.629	18854
	GPT-4 Rewrite	0.485	13999
	SAMMO	0.637	15292
LAMA	Baseline	0.380	103606
	STDC	0.426	65728
	APE	0.328	77980
	Stopwords	0.337	103573
	GPT-4 Rewrite	0.335	53192
	SAMMO	0.447	53087