Prompts As Programs: A Structure-Aware Approach to Efficient Compile-Time Prompt Optimization

Anonymous ACL submission

Abstract

Large language models (LLMs) can now handle longer and more complex inputs, which facilitate the use of more elaborate prompts. However, prompts often require some tuning to improve performance for deployment. Recent work has proposed automatic prompt optimization methods, but as prompt complexity and LLM strength increase, many prompt optimization techniques are no longer sufficient and a new approach is needed to optimize *meta prompt programs*. To address this, we introduce SAMMO, a framework for *compile-time* optimizations of metaprompt programs, which represent prompts as structured objects that 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.

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 dynamic templates (aka metaprompts), where static information (eg, instructions) is combined with input-dependent information (e.g., user queries, retrieved documents in Retrieval Augmented Generation (RAG) systems) at runtime to generate the desired output. There is a clear trend towards prompts becoming complex programs in and of themselves.

To achieve an acceptable level of accuracy for deployment, prompts are often manually fine-tuned. This process typically involves adding information to handle exceptions and corner cases. To improve this process, there has been a great deal of recent work on automatic optimization methods. The first wave of work in this area focused on simpler prompts (i.e., with less structure) and unstructured mutators such as text rewrite operations (Chen et al., 2023; Pryzant et al., 2023; Zhou et al., 2023). The second wave of work has focused on optimizing prompts with more complex programmatic structure such as meta-prompts. Initial work in this direction focused on applying textual mutators to different prompt components (Fernando et al., 2023; Ye et al., 2023). Subsequent work considered both textual mutation and hyperparameter selection (Sclar et al., 2023). However, with the increasing strength of LLMs and increasing complexity of prompt structure, many prompt optimization techniques are no longer applicable and a new approach is needed that is able to optimize *metaprompt programs*.

To address this, we introduce SAMMO, a general purpose framework for *compile-time* optimizations of prompt programs. Individual operations and parts of the prompt are represented through components in a call graph, in spirit similar to DSpy (Khattab et al., 2023). However, SAMMO goes beyond this previous work by (i) also representing the internal structure of prompts and (ii) offering "compile-time" optimizations over prompt components (eg. textual content, hyperparameters). It considers metaprompts as programs—to allow for modular construction of complex prompts and facilitate *compilation* of more effective prompts.

Specifically, SAMMO represents metaprompts as structured objects, which allows for a much richer set of transformations to be searched over during optimization. We formalize the optimization as a problem of searching over the structure of a complex *metaprompt* π, which may include many structured components such as: task descrip-

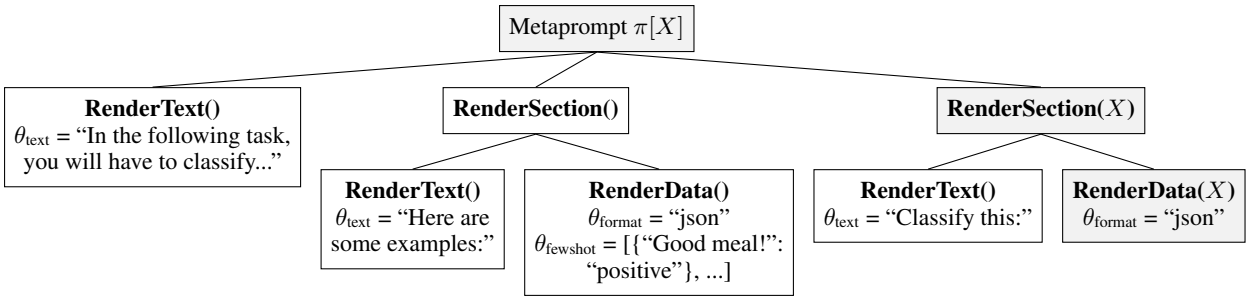

Figure 1: A simple metaprompt for a review classification task. SAMMO represents metaprompt as a dynamic function graph where each node computes a new value based on the results of its children, the input data X as well as node-specific parameters θ. Here, nodes that use the input data X are marked in light gray.

tion, guidelines, examples, and input/output format. Note, a metaprompt is designed to be combined dynamically with different input data X at run time, i.e., $\text{LLM}(\pi[X]) = Y$. Figure 1 shows a possible structured program for a review classification task.

Drawing from approaches for neural architecture search, SAMMO carries out a genetic search over a set of mutation operators that can change the structure and information contained in non-trivial ways. We assume that a small labeled training dataset is available as supervision for the search process and explicitly formulate the search as a multiple objective optimization, which allows practioners to balance cost and accuracy.

SAMMO is a general framework for prompt optimization in a black-box setting where the practitioner can only sample from the output of LLMs, reflecting common API capabilities (Sun et al., 2022). We show that SAMMO encompasses several instruction optimization and compression techniques as special cases. We demonstrate the utility of SAMMO in three scenarios: (1) instruction tuning, (2) retrieval augmented generation (RAG) pipeline tuning, and (3) prompt compression for annotation tasks. Our experiments show the SAMMO generalizes previous methods and provides gains of 10-100% in instruction tuning, 2-350% 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. We make all code available open-source at (TBD) under an MIT license.

2 Problem Definition & Notation

Let π refer to a metaprompt function that takes input data X and maps it to a string that is fed to an LLM to generate an output, i.e., $\hat{Y} = LLM(\pi[X])$. Our goal is to optimize the performance of π by transforming it into a more performant metaprompt π^* via modifications to its structure, parameters, and content. More precisely, our goal is to find a metaprompt π with minimal loss across the entire data distribution:

$$\pi^* = \arg\min_{\pi \in \Pi} \mathbb{E}_{D_s \sim P(X,Y)}[S_\phi(\pi, D_s)]. \quad (1)$$

Here, $P(X, Y)$ is the distribution that the labeled samples D_s are drawn from, Π represents the set of all metaprompts, and $S_\phi \in \mathbb{R}$ is a scalarized loss function that specifies trade-offs between potentially multiple individual quality objectives S_i, (e.g., $S_i := \mathcal{L}_{S_i}(Y, LLM(\pi[X]))$.

Since the data distribution $P(X, Y)$ is unknown and the evaluation cost is prohibitive, we resort to using the following sampled score in the spirit of empirical risk minimization (ERM):

$$\hat{\pi} = \arg\min_{\pi \in \Pi} S_\phi(\pi, D_{\text{train}}). \quad (2)$$

2.1 Compile-time Optimization

In this paper, we focus on optimization that can only be done once before deploying the metaprompt, which we refer to *compile-time optimization* and correponds to the problem of solving Eq. 2. More formally, the optimization itself is a function returning another function (the metaprompt):

$$\hat{\pi} = \rho_{\text{compile}}(\Pi, D_{\text{train}}, S_\phi). \quad (3)$$

In this case, the optimization can only be done once before deploying the metaprompt. This is different from *run-time* optimization where the optimizer ρ_{run} gets called with the filled in metaprompt, i.e., $\rho_{\text{run}}(\pi[X]) = \tilde{X}$ and then the LLM is prompted as with $LLM(\tilde{X})$. This optimization

procedure would need to be executed repeatedly for each different input data X. Since we aim to amortize the optimization cost over multiple uses of the metaprompt, we do not consider run-time optimization further. However, we note that run-time optimizations can compliment compile-time optimizations if sufficient resources are available.

Motivated by real-world needs, we make the following assumptions:

1. We only have small amounts of data for optimization (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.

2. The language model is a closed black box and does not output probabilities, reflecting recent changes in how access to LLMs is provided. However, we can sample from it: $\mathbf{y} \sim \text{LLM}(Y \mid \pi[X])$.

2.2 Metaprompts As Programs

To search efficiently over the exponentially large space of all metaprompts Π, we consider their complex programmatic structure, which typically comprises parameterized components executed in sequence. More specifically, we represent the structure of π as a *directed acyclic* function graph G, i.e., $\pi[X] = f_{G_\pi}(X)$. Here $G_\pi = (V, E)$ is a DAG with a single root node v_r. Directed edges $e_{ij} \in E$ represent parent (v_i) to child (v_j) relationships in the metaprompt structural components.

Each node $v \in V$ has a function type ψ_v and static parameters θ_v. We use θ_v to refer to both static *textual* input in the meta prompt that is common across calls (e.g., instructions) and any hyperparameters of the components (e.g., format specifiers). Each node also gets dynamic information from (1) X (which may be transformed by its parents) and (2) messages from its children.

Then the graph is recursively evaluated from the rode node v_r as follows:

$$f_{G_\pi}(X) = \psi_{v_r}\left((X \cup \psi^r_{\text{children}}), \theta_{v_r}\right),$$

where

$$\psi^r_{\text{children}} = \left\{\psi_{v_i}\left(\dot{X}_{\psi_{v_r}}(X), (\theta_{v_i} \cup \theta_{v_r})\right)\right\}_{v_i}$$
$$s.t.\ e_{ri} \in G_\pi.$$

Here X to refer to dynamic input that changes based on the call to the meta prompt (e.g., user query) and we note that X may consist of a batch of more than one query. We use $\dot{X}_{\psi_v}(.)$ to refer to a possible transformation of the dynamic input that a parent v can send its children, e.g., to perform minibatching.Note that the input can also be passed on directly without transformation, e.g., $\dot{X}_{\psi_v}(X) = X$.

We denote the space of metaprompts as Π, and each $\pi \in \Pi$ consists of a different program structure G. With this notation, we can see how searching over Π in Eq. 2 amounts to combinatorial search over G. We initialize the search with a baseline prompt π_0 and generate successor functions through mutations to G_{π_0}. Mutators (see Sec. 3.1) operate on G_π at the node or neighborhood level. They can reorder, replace, or delete nodes. They can also modify θ_v associated with a node (eg. by modifying instructions or parameter values).

We note the similarity of metaprompt search with neural architectures search—nodes functions that ignore X can be seen as global parameters, ψ_{v_r} correspond to projection operators or activation layers as they aggregate input from their predecessors.

To give some intuition for what this looks like in practice, Figure 1 shows a metaprompt for a review classification task which has already been instantiated with some input data X. It has a section with task instructions at the beginning, followed by a set of static in-context examples (although they could be selected dynamically as well) and the unlabeled input data. Note that the data format is explicitly represented—here it serializes data to a JSON array with each example being a dictionary.

3 SAMMO: Structure-Aware Multi-objective Metaprompt Optimization

We outline SAMMO, which is a general framework for optimizing the performance of metaprompts. SAMMO uses general-purpose search algorithms with a rich set of mutation operators to explore the space of prompts. Figure 2 presents an overview of SAMMO's main components that allow for efficient prompt optimization. First, there is a base layer of programming primitives from which more complex prompt programs are being built (cf. Figure 1). These comprise the nodes of the metaprompt function graph G, which we described in Section 2.2. On top of that sits a layer of prompt mutators which

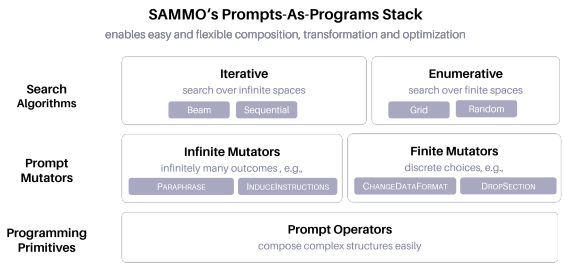

Figure 2: SAMMO is a flexible framework for structured prompt optimization offers two classes of search algorithms, depending on the set of mutators used.

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**
    $M_{\text{valid}} = \{m \text{ can be applied to } \pi \mid m \in M\}$
    M_π = SAMPLEMUTATORS(M_{valid})
    $\forall m \in M_\pi$: Add MUTATE(m, Π_{active}) to Π_{nextgen}
  end for
  $\Pi_{\mathcal{C}} \leftarrow \Pi_{\mathcal{C}} \cup$ PRUNE($\Pi_{\text{nextgen}}, D_{\text{train}}, S_\phi$)
end while
return best candidate from $\Pi_{\mathcal{C}}$

are then used with the search algorithms in the topmost layer. We outline these two layers next.

3.1 Prompt Mutation Operators

At the heart of SAMMO optimization are mutation operators. Formally, a mutation operator is a probabilistic function $m : \Pi \times \Pi_\Omega \longrightarrow [0, 1]$ that specifies how to transition from a metaprompt π to an edited version $\pi' \in \Pi_\Omega$. This *structure-aware* component of SAMMO opens up a new and exciting class of operators. These can range from trivial (e.g., rephrasing a sentence) to complex (e.g., inducing a new set of task instructions). We call an operator *finite*, if for all inputs π, the set of possible outputs is finite, and *infinite* otherwise. A mutation operator's type will also determine what search algorithms can be used.

Table 1 shows a non-comprehensive set of mutation operators, grouped by what part of a metaprompt 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.

3.2 Search Algorithms

There are two classes of search algorithms that SAMMO provides. First, if all mutators used are finite, then SAMMO can explore the space via *enumerative search*, either exhaustively (grid search) or by sampling search points at random (random search). As we show in Section 5, this simple approach can be very effective in practice. When some of the mutators are infinite, SAMMO offers *iterative search* algorithms to optimize prompts. Algorithm 1 shows the skeleton of how SAMMO implements iterative search. Starting with an initial baseline metaprompt (π_0) and small amount of training data as input, 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. For example, beam search typically starts with one candidate and then selects all parents at the current depth, keeping only the top k performing children for the next round. The novelty of SAMMO lies not in this outer search but in the fact that we represent the higher level structure of a metaprompt explicitly which in turn allows us to define a rich set of operators that can both transform the structure as well the content.

3.3 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 for special cases of 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 performs beam search with add, delete, swap and paraphrase mutation operations on the constituents.

Table 1: Examples for mutation operators, grouped by what part of a metaprompt π they affect. SAMMO allows for a rich set of operations whereas traditional prompt optimization techniques have 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

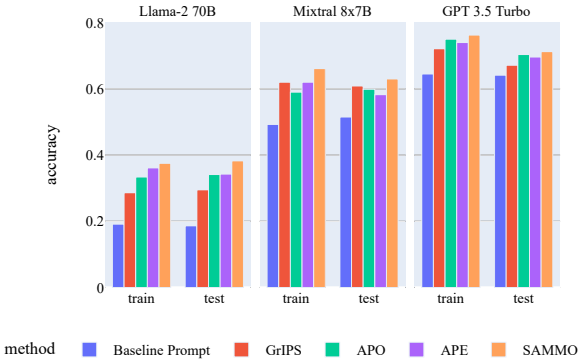

Figure 3: SAMMO consistently outperforms all other instruction tuning methods, independent of the backend LLM.

4 Use Case: Instruction Tuning

In theory, SAMMO subsumes many existing methods that do instruction tuning where the task is to generate static instructions that tell the LLM how a task should be solved. To better align with previous work, we use eight zero-shot BigBench classification tasks (Srivastava et al., 2023) that still have headroom to improve (i.e., the accuracy of the baseline prompt π_0 with GPT-3.5 is < 0.9). We subsample training and test sets of size $n = 100$ from the official splits.

We compare against APE (Zhou et al., 2023), Automatic Prompt Optimization (Pryzant et al., 2023) and GrIPS (Prasad et al., 2023). For the backend LLMs, we consider two open-source models, Mixtral 7x8B (Jiang et al., 2024) and Llama-2 70B (Touvron et al., 2023) and GPT-3.5 (Brown et al., 2020). We do not include GPT-4 since it showed negligible headroom for improving instructions in pilot experiments (cf. also Section 5).

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. 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). However, the zero-shot tasks here are relatively simple, with more headroom existing for more complex tasks as the next use case shows.

5 Use Case: 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. Our goal is to show how application of SAMMO can yield substantial gains in a realistic few-shot scenario. Towards that end, we subsample a small set of annotated fewshot examples ($n = 600$) from each official split, and sample $n = 100$ for training and testing. To the best of our knowledge, there are no existing baselines for this scenario.

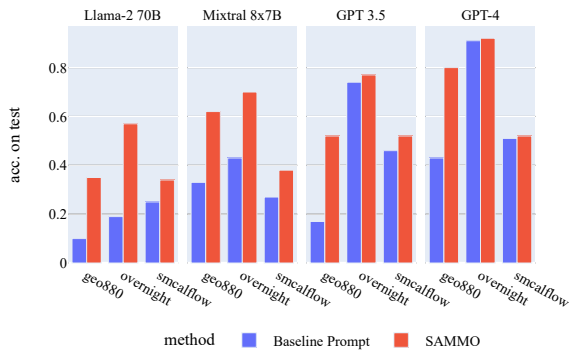

Figure 4: SAMMO efficiently improves baseline prompt accuracy across all semantic parsing datasets and backend LLMs with only 24 candidate evaluations.

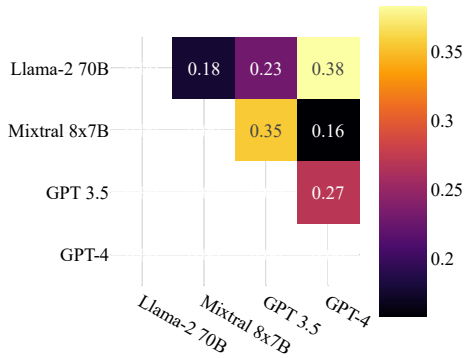

Figure 5: There is only weak correlation between how well candidates do during enumerative search between LLMs.

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 exact match accuracy over a search space of size 24, trying out different data formats, number of few shot examples and DSL specifications. For details, see Appendix A.5. We consider all backend LLMs as before, plus GPT-4.

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 65%. However, even GPT-4 can benefit from changes explored by SAMMO with a maximum relative gain of 86% on GeoQuery. 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.

6 Use Case: Prompt Compression

In these experiments, our goal is to optimize the weighted costs of a metaprompt subject its accuracy not dropping below a certain threshold τ_{acc} and its parse error rate being below 10%.

The weighted costs are given as

$$\text{cost}(\pi, D) = S_{\text{in}}(\pi, D) + 2 \cdot S_{\text{out}}(\pi, D). \quad (4)$$

where we weight the number of output tokens double output tokens S_{out} compared to the input tokens S_{in} to reflect current billing schemes for publicly available LLMs, but the loss function is easily configurable upfront. Assuming a baseline prompt π_0 that has a reasonable level of accuracy, we set the threshold τ_{acc} such that it is above the baseline prompt performance with $\epsilon = 0.02$.

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:

- **Baseline Prompt**: Our baseline prompt π_0 uses the official instructions provided with a task followed by $k = 5$ in-context examples.

- **APE**: Automatic Prompt Engineering (Zhou et al., 2023): Instead of optimizing for accuracy, we used the same objective as for SAMMO (Eq. 4)

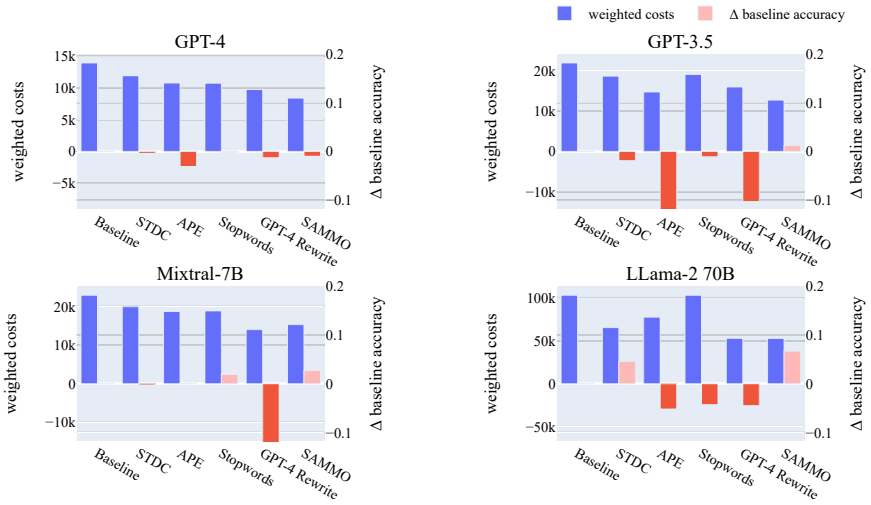

Figure 6: SAMMO is able the achieve high compression rates while still maintaining high accuracy. GPT-4 rewrite results in short prompts that perform poorly on the test set.

- **STDC**: Syntax-guided Task Definition Compression (Yin et al., 2023a) runs a sequential search to prune syntax tree constituents.
- **Stopwords**: We search over two different stop word lists as well as whether to apply them to examples, to the task definition, or both.
- **GPT-4 Rewrite**: Using the templates from Li et al. (2023), we try out ten different instructions to compress the task definition and the in-context examples.
- **SAMMO**: We use beam search as the backbone of Algorithm 1 and initialize Π_C with four variants for θ_{format} of the baseline prompt π_0. We use all mutation operators listed in Table 1 as possible operations, and choose mutators uniformly at random during the search.

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 back-end 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 find prompts that do not generalize well to the test set and experience large drops in accuracy.

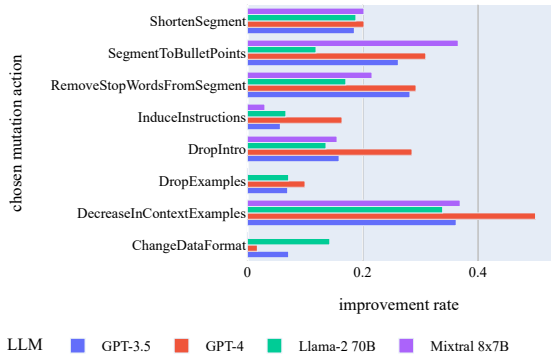

Figure 7: Mutation actions that lead to an improvement in the objective differ from model to model.

For each mutation operation chosen by SAMMO during the search process, we recorded whether it resulted in an improvement of the objective (Eq. 4). 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.

7 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, *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. Complementary 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 optmization, 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.

Limitations

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. Finally, SAMMO adopts a supervised learning scenario where labels are required; we plan to address more unsupervised tasks in the future.

8 Conclusion

In this paper, we introduced a new framework, Structure-aware Multi-Objective Metaprompt Optimization (SAMMO) to enable efficient optimization of metaprompt programs during compile-time.

SAMMO represents metaprompts as dynamic function graphs, and employs a set of mutation operators to alter the structure and content of metaprompts. This approach notably outperforms and generalizes existing methods of prompt optimization and compression, as demonstrated through several use cases tasks in our experimental evaluation.

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*.

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.

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

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*.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. In *ICML*.

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 Appendix

A.1 Implementation Details

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`

A.2 Instruction Tuning

See Table 2

A.3 Prompt Compression: Table form of main results

See Table 3 for numeric results.

A.4 Prompt Compression: Examples prompts

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

A.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.

A.5.1 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
```

A.5.2 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
```

A.5.3 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
```

A.5.4 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
```

A.5.5 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
```

A.5.6 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 2: 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 3: 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