Cost-Effective Hyperparameter Optimization for Large Language Model Generation Inference

Abstract Large Language Models (LLMs) have sparked significant interest in their generative capabilities, leading to the development of various commercial applications. The high cost of using the models drives application builders to maximize the value of generation under a limited inference budget. This paper presents a study of optimizing inference hyperparameters such as the number of responses, temperature and max tokens, which significantly affects the utility/cost of text generation. We design a framework named EcoOptiGen which leverages economical hyperparameter optimization and cost-based pruning. Experiments with the GPT-3.5 models on a variety of tasks verify its effectiveness. EcoOptiGen is implemented in `https://anon-github.automl.cc/r/LLM/ecooptigen/README.md`.

1 Introduction

Large language models (LLMs) like GPT-3 and GPT-4 (Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023) have demonstrated impressive capabilities in a wide range of generative tasks, including story telling (Lucy and Bamman, 2021; Chen, 2022), code generation (Trummer, 2022; Poesia et al., 2022), math problem solving (Cobbe et al., 2021; Zong and Krishnamachari, 2022), and many others (Wang et al., 2022). Even though the LLMs do not always generate perfect answers, they have initiated a trend of building powerful user experiences such as coding assistants and chat-enabled search engines. As the interest of building LLM-enabled applications keeps growing, the demand for technologies for getting the best value out of generation inference will also grow.

The research community has recently studied the effect of individual hyperparameters on the inference performance, such as the prompt (Liu et al., 2021; Mishra et al., 2021; Shieh, 2022) and temperature (Branwen, 2020; Nadeem et al., 2020). However, little is known about how to optimize the different hyperparameters collectively and systematically. Moreover, the monetary cost is a concern for most application builders. High costs and implications on energy consumption and environmental impact (Schwartz et al., 2020) provide a strong incentive to systematically optimize the hyperparameters towards maximal utility and minimal cost.

In this paper, we present the first study on the systematic hyperparameter optimization for text generation inference using LLMs. Given the cost concern, we adopt an economical hyperparameter optimization method (Wang et al., 2021), and propose a cost-based pruning strategy to improve the optimization efficiency under budget constraints. We apply our optimization framework, named *EcoOptiGen*, to tune multiple hyperparameters jointly, including the number of responses, max tokens, temperature, probability mass, and prompts.

To study the effectiveness of EcoOptiGen, we evaluate it on the following datasets: APPS (Hendrycks et al., 2021a), HumanEval (Chen et al., 2021) (for code generation); MATH (Hendrycks et al., 2021b) (for math problem solving); and XSum (Narayan et al., 2018) (for text summarization). On all the four datasets, we observe that EcoOptiGen can find higher quality hyperparameter settings than the default settings suggested by a recent LLM benchmark HELM (Liang et al., 2022) or simple modifications for the same budget. Our pruning technique is shown to be effective in increasing the tuning performance significantly. We further find that the holistic hyperparameter optimization can mitigate idiosyncrasies to prevent suboptimal results. For example, it was generally recommended by OpenAI to use "code-davinci-002" for code generation and "text-davinci-003" for other text generation (Shieh, 2022). However, our experiments show that a tuned alternative model can largely outperform the recommended model.

 © 2023 the authors, released under CC BY 4.0

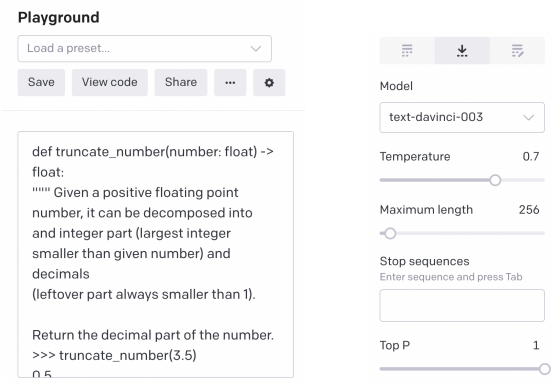

Figure 1: A code generation example using GPT-3.5, and a few examples of hyperparameters users can set for the inference, such as prompt, temperature, and max tokens.

2 Background

2.1 Text Generation with LLMs

The Input and Output of LLM Text Generation. Figure 1 shows one example of the input prompt to LLM. Upon receiving the prompt, LLM performs inference to generate one or more output responses. The input prompt can further include multiple examples to demonstrate what kind of responses are desirable. The output can be consumed by an application in various ways, e.g., code executor (Hendrycks et al., 2021a) or math equation checker (Hendrycks et al., 2021b). It is helpful for the inference to generate multiple responses and search for the best one, e.g., in code generation (Chen et al., 2021) and machine translation (Wullach and Chazan, 2022). The utility of the generated text is determined by the application consuming it. For example, if one of the generated responses is a valid piece of code which passes a predefined programmatic test, the code generation can be considered as successful.

The Cost of Text Generation with LLMs. The cost of using LLMs for text generation is typically measured in terms of the number of tokens in the input and output combined. This is because LLMs are often used as a service, and the cost of a request is proportional to the amount of computations required to generate the output. From the perspective of an application builder using LLMs, the goal is to maximize the utility of the generated text under an inference budget constraint (e.g., measured by the average dollar cost needed to solve a coding problem). This can be achieved by optimizing the hyperparameters of the inference, which can significantly affect both the utility and the cost of the generated text. In the next section, we will discuss how hyperparameters affect the utility and the cost.

2.2 How Do Hyperparameters Affect Text Generation Performance?

The Impact of Individual Hyperparameters. We take a representative API from OpenAI, i.e., the completions API (ope, 2023), to analyze the tunable hyperparameters and their impact on the cost and the utility (e.g., accuracy, success rate): (1) model - this is a required input, specifying the model ID to use. (2) prompt - the input prompt to the model, which provides the context for the text generation task. (3) max_tokens - the maximum number of tokens (words or word pieces) to generate in the output. (4) temperature - a value between 0 and 1 that controls the randomness of the generated text. A higher temperature will result in more random and diverse text, while a lower temperature will result in more predictable text. (5) top_p - a value between 0 and 1 that controls the sampling probability mass for each token generation. A lower top_p value will make it more likely to generate text based on the most likely tokens, while a higher value will allow the

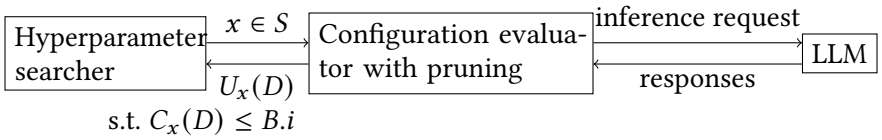

Figure 2: Architecture.

model to explore a wider range of possible tokens. (6) n - the number of responses to generate for a given prompt. Generating multiple responses can provide more diverse and potentially more useful output, but it also increases the cost of the request. (7) stop - a list of strings that, when encountered in the generated text, will cause the generation to stop. This can be used to control the length or the validity of the output. (8) presence_penalty and frequency_penalty - values that control the relative importance of the presence and frequency of certain words or phrases in the generated text. These hyperparameters can be useful for controlling the focus and balance of the generated text. (9) best_of - the number of responses to generate server-side when selecting the "best" (the one with the highest log probability per token) response for a given prompt.

The Joint Impact of Multiple Hyperparameters. It can be seen that the cost and utility of text generation are intertwined with the joint effect of these hyperparameters. There are also complex interactions among subsets of the hyperparameters. For example, the temperature and top_p are not recommended to be altered from their default values together because they both control the randomness of the generated text, and changing both at the same time can result in conflicting effects (ope, 2023); n and best_of are rarely tuned together because if the application can process multiple outputs, filtering on the server side causes unnecessary information loss; both n and max_tokens will affect the total number of tokens generated, which in turn will affect the cost of the request. These interactions and trade-offs make it difficult to manually determine the optimal hyperparameter settings for a given text generation task.

3 EcoOptiGen

We first introduce the following notations and definitions for EcoOptiGen:

Tuning Data D, a small set of examples which can be used to measure the goodness of each hyperparameter configuration. Each data example contains a few text fields. For example, the HumanEval (Chen et al., 2021) dataset contains an input field which is the concatenation of the Python function signature and the doc string, and a test field of the test code. In some places, we use $D.in$ to denote the input field.

Utility Function U, a function that represents the utility (a real-valued score) of the generated text. For example, for code generation, the utility is the success rate of passing the test; for text summarization, the utility is the effectiveness of summarization such as the rouge score. The utility of multiple verifiable responses (i.e., the best response can be selected from them by the application) is defined as the best utility score in all responses.

Budget Constraints $B = (B.i, B.o)$, a tuple of two values: $B.i$ is the average inference budget per example in the tuning data D and $B.o$ is the total optimization budget. Generally, they are measured as the dollar cost which is proportional to the number of tokens in the input and output (ref. Section 2). When the price per token is given, the dollar cost can be converted to the number of tokens. In the following of this paper, we use the number of tokens for simplicity: $B.i$ is the allowed average number of token consumptions per request, such as 1K, and $B.o$ is the the total number of tokens allowed to consume in the tuning process, such as 1M. When $B = (1K, 1M)$, $|D| = 20$, if each request consumes exactly 1K tokens, the number of allowed hyperparameter configurations to try during the optimization is equal to $1M/1K/20 = 50$.

Search Space S, a dictionary where the key is each hyperparameter's name and the value is the range of values to search for the corresponding hyperparameter. Following the analysis in Section 2.2, we design a default search space as shown in Table 1, if users do not provide this input. The necessity of tuning certain hyperparameters depends on the application.

Table 1: Default search space for the optimization framework. Some hyperparameters are fixed as constants by default but users can override all of them according to domain knowledge.

Hyperparameter	Default search range	Note
model	["text-davinci-003"]	The latest GPT-3.5 model
prompt	["{$D.in$}"]	A list of prompt templates. "{$D.in$}" will replaced by the input fields in each instance to produce the actual prompt per instance. Typically overridden by users based on domain knowledge
max_tokens	lograndint(100, 1000)	Random integers, logarithm distributed
temperature_or_top_p	[{"temperature": uniform(0, 1)}, {"top_p": uniform(0, 1)}]	Hierarchical search space: one configuration will either choose a temperature or a top_p
n	randint(1, 100)	Random integers between 1 and 100
stop	None	Users can specify application-dependent stop choices
presence_penalty	0	Users can specify a float range within [-2, 2] if needed
frequency_penalty	0	Users can specify a float range within [-2, 2] if needed
best_of	1	Users can specify an int range lower bounded by 1 and fix n to 1

Average Utility and Cost Consumptions for Configuration x over Instances in D: $U_x(D)$ and $C_x(D)$. A configuration x is *invalid* if $C_x(D) > B.i$.

Within the specified optimization budget $B.o$, the framework iteratively tries different configurations in the given search space S, and outputs a configuration x^* with the maximal average utility $U_{x^*}(D)$ on the tuning data D subject to the average inference budget $C_{x^*}(D) \leq B.i$. The architecture is depicted in Figure 2. A *hyperparameter searcher* proposes configurations, and it invokes a *configuration evaluator* to assess the validity and utility of each configuration.

3.1 Hyperparameter Searcher

We opt for a blackbox optimization approach because we aim to make the framework generically applicable to (1) LLMs as a service, (2) complex utility functions which potentially involve blackbox evaluation of the output returned by LLMs. Our framework abstracts away from the internal process of computing the utility for a configuration, which involves making requests to LLMs and evaluating the responses with application-specific procedures.

There are a variety of blackbox optimization techniques, such as random search (Bergstra and Bengio, 2012), Bayesian optimization (Bergstra et al., 2011), evolutionary search (Goldberg and Deb, 1991), and local search (Wu et al., 2021). We chose a method that combines Bayesian optimization and local search, named BlendSearch, due to its cost efficiency (Wang et al., 2021). The local search method in BlendSearch performs randomized direct search with a provable convergence rate and cost bound. Bayesian optimization is used to generate starting points for the local search, and different local search threads are prioritized adaptively. The original BlendSearch is designed for training hyperparameter optimization and used the training time as the measurement for cost. We adapt it to optimize the inference hyperparameters, and generalize the cost metric.

3.2 Configuration Evaluator

A simple evaluator takes a configuration x as the input and outputs the metric to optimize. It loops over the data examples in D. For each example $d_i \in D$, a request is made to the LLM service using the configuration and the input fields $d_i.in$. The responses from the LLM service are then used to compute the utility U, and measure the cost consumption C. When the loop is over, the

average cost $C_x(D)$ is compared with the user-provided bound $B.i$. If $C_x(D) \leq B.i$, the average utility $U_x(D)$ for all the data in D is returned, otherwise a zero value is returned to indicate that the configuration is invalid. In the following, we present an improvement to the simple evaluator.

If a configuration x is invalid, it is beneficial to terminate the trial early to save unnecessary cost. We design a pruning strategy by judiciously varying two cost-related factors during a trial: the number of tuning data examples, and the number of responses. A full evaluation of a configuration x needs to send $|D|$ LLM requests, each asking for $x.n$ responses, where $x.n$ is the setting of the hyperparameter n in configuration x (or the setting of best_of if best_of is searched instead of n). Our goal is to spend a much smaller cost in invalid trials. The full procedure is detailed in Algorithm 1 in the appendix.

Initial Validity Check. For a given configuration x, before the expensive evaluation starts, we first check whether we could prune the configuration directly (line 4-10 of Algorithm 1). This check is based on an assumption specific to our optimization problem.

Assumption 3.1. Given two configurations x_1 and x_2 with the same setting of model, prompt, and stop, if the number of responses and max_tokens in x_1 are both equal or larger than those in x_2, then we expect x_1 has an equal or higher average token consumption than x_2.

A consequence of the assumption is that if x_2 is invalid, then x_1 is invalid too. If x_1 is valid, then x_2 is valid too. Our pruning leverages this assumption to find a max known valid n and a min known invalid n for a configuration x, using the valid and invalid sets of already tried configurations X_{valid} and X_{invalid} which share the same setting of model, prompt and stop with x.

$$\text{max_valid_n} = \max_{x' \in X_{\text{valid}}, x'.\text{max_tokens} \geq x.\text{max_tokens}} x'.n \tag{1}$$

$$\text{min_invalid_n} = \min_{x' \in X_{\text{invalid}}, x'.\text{max_tokens} \leq x.\text{max_tokens}} x'.n \tag{2}$$

Then, depending on the relation among max_valid_n, min_invalid_n, and $x.n$, we do the following:

1. If $x.n \leq$ max_valid_n, we evaluate this trial as is. This corresponds to the case x is expected to be valid based on Assumption 3.1.

2. Otherwise ($x.n >$ max_valid_n), if $x.n \geq$ min_invalid_n, we prune this trial without any evaluation. This corresponds to the case where x is expected to be invalid based on Assumption 3.1.

3. Otherwise (max_valid_n $< x.n <$ min_invalid_n), we start evaluating the trial from number of responses equal to max_valid_n. This corresponds to the case where x is expected to be either valid or invalid, and max_valid_n is expected to be a valid number of responses to use for x.

The order of our check takes into the consideration that Assumption 3.1 can be violated occasionally, i.e., max_valid_n may be occasionally equal or larger than min_invalid_n. By checking condition 1 before condition 2, we keep the chance of evaluating a trial when the violation happens.

The Outer Loop of Algorithm 1: Varying the Number of Responses. After the initial check is passed (case 1 or 3), we evaluate the configuration by gradually doubling the number of responses until it reaches $x.n$ (line 11-end). The starting point of n is decided according to the rules above. To evaluate a particular n, we temporarily modify $x.n$ as $n - n'$, where n' is the last evaluated number of responses for the current configuration in D (0 at the start point or when data skipping happens, as explained in the next paragraph). That makes use of the results from requests made for evaluating smaller n for the current configuration and saves cost compared to requesting n responses. Note that while the gradual increase of n makes it possible to terminate a trial with a smaller number of responses, it can also increase the total cost of evaluating a valid trial as the input tokens occupy the consumption in every request repeatedly. That issue is mitigated by a

few choices in our design: (a) we start from the max known valid n instead of 1; (b) the geometric increase of n reduces the number of times that n is varied, to a logarithm factor, rather than a linear factor in a linear schedule, and (c) the data skipping described next helps reducing the number of requests for smaller n if the trial is valid.

The Inner Loop of Algorithm 1: Varying the Number of Data Examples. For each fixed number n of responses, we employ progressive subsampling (Provost et al., 1999) over the tuning data D to prune a trial (line 12-32). After we get the responses for k examples in D, we can compute the mean of their cost and utility. We denote the subset of the k examples as D_k. $C_x(D_k)$ is an estimate of $C_x(D)$. Hoeffding-Serfling inequality (Bardenet and Maillard, 2015) can be used to compute the upper (lower, resp.) bound for $C_x(D_k)$ if $C_x(D)$ is indeed below (above, resp.) $B.i$. If $C_x(D_k)$ is larger than the upper bound, we terminate the trial, and update X_{invalid}. If $C_x(D_k)$ is smaller than the lower bound and the current n is smaller than the original $x.n$, we skip the remaining data points in D, reset $n' = 0$, and update X_{valid}. The number k is doubled until it reaches $|D|$. The geometric increase of k limits the number of times this hypothesis test is conducted per trial to reduce the chance of incorrect pruning.

4 Experiments

We are interested in investigating the following research questions: First, for text generation tasks, how much gain can EcoOptiGen achieve by tuning the hyperparameter settings under a budget constraint? Second, how does varying the inference budget affect the optimization result? Third, how does varying the model affect the optimization result? In this section, first, we describe the setting of our experiment in Section 4.1. Then, we investigate the three research questions in Section 4.2 through 4.4. We discuss limitations and future work in Section 4.6.

4.1 Setup

Datasets. To evaluate the performance of EcoOptiGen, we select a diverse set of text generation tasks from the HELM benchmark (Liang et al., 2022) (v1.0): code generation, math problem solving, and text summarization. For code generation, we evaluate EcoOptiGen on two datasets: APPS (Hendrycks et al., 2021a) (a dataset for generating Python code for a coding problem given the problem description) and HumanEval (Chen et al., 2021) (a dataset for generating Python code based on the function name and docstring). For math problem solving, we use the MATH dataset (Hendrycks et al., 2021b) (a dataset for math problems containing chain-of-thoughts, i.e., the derivation steps for the solution). For text summarization, we evaluate EcoOptiGen's performance on the XSum dataset (Narayan et al., 2018) (a large dataset for summarizing news articles).

For tuning, we randomly sample 20 examples for tuning of all the datasets, except for 60 of XSum since it contains more data. For each dataset, we randomly sample a few hundred examples for testing. The test set selection procedure follows HELM.

Evaluation Metric. For the two code generation tasks, test cases are already provided by the datasets for verifying responses at inference time. For each code generation instance, as long as any response passes the test cases, the score that EcoOptiGen receives for this instance is 1 and 0 otherwise. For MATH, if one of the returned chain-of-thought responses has an equivalent final answer with the ground truth, EcoOptiGen receives 1 for this instance and 0 otherwise. For XSum, we use 'best_of' to rerank the generated responses by their mean log probabilities and use the Rouge-2 score for the top response (Lin, 2004).

Comparative Methods. We evaluate the following alternatives to compare with EcoOptiGen:

• HELM. We check the best score evaluated under the HELM benchmark. For each dataset, we find the best performing GPT-3.5 model reported by HELM, and then re-evaluate on our test data using the same model and hyperparameter settings (with modified prompts on APPS and

Table 2: Results using 'best' GPT-3.5 model according to HELM.

Method	APPS	HumanEval	MATH	XSum
HELM	0.03	0.465	0.378	0.140
EcoOptiGen (HELM budget)	0.05	0.521	0.414	0.144
Search	0	0.493	0.769	0.136
Search+PSR	0	0.493	0.739	-
EcoOptiGen	0.05	0.792	0.771	0.144
HELM (modified)	0.03	0.701	0.403	0.140

MATH as explained in the next paragraph) from HELM. The selected model is consistent with the recommendation from OpenAI (Shieh, 2022) except for XSum, where text-davinci-002 has a higher score than text-davinci-003 in HELM. The complete details of these configurations are listed in Table 4 of the appendix. The reason to use this baseline is that HELM has a broad coverage of the latest LLMs with a specific hyperparameter setting per task, which is rare to find elsewhere.

• Search. This is a method that applies the same hyperparameter searcher as EcoOptiGen but does not use pruning or alter the optimization metric.

• Search + PSR. This is the method that is the same as Search, but uses probabilistic success rate instead of success rate as the optimization metric. Not relevant in the XSum dataset.

By default, the input to all the search-based methods is set to $B.i = 1K, B.o = 1M$. The search space follows Table 1, while "model" and "stop" are overriden by the HELM config. For HumanEval, we search the prompts over four templates: "{D.in}", "# Python 3{D.in}", "Complete the following Python function:{D.in}", and "Complete the following Python function while including necessary import statements inside the function:{D.in}". For the purpose of saving inference cost, we use zero-shot prompt rather than the two-shot used in HELM on APPS. For MATH, we use only one fixed demonstration example for all categories in the prompt as opposed to eight per category in HELM. For XSum, the same prompts, n and max_tokens from HELM are used, while best_of is searched in the range of randint(1, 100); and the budget is set to $B.i = 2K, B.o = 4M$.

4.2 EcoOptiGen's Performance

The performance scores of EcoOptiGen are shown in Table 2 along with other comparative methods. For all the 4 datasets, EcoOptiGen outperforms the best untuned GPT-3.5 model in the HELM benchmark. To verify whether the performance gain is simply due to the increased number of responses, we add a 'HELM (modified)' method in Table 2 which modifies the number of responses to match the inference budget consumed by the best configuration from EcoOptiGen. We also add 'EcoOptiGen (HELM budget)' which is EcoOptiGen's performance when using the same inference budget as HELM per task. The comparison between the first two rows or between the last two rows of Table 2 suggests that the hyperparameters in the HELM benchmark are under-tuned, and jointly tuning all the hyperparameters can be better than simply increasing the number of responses.

In Table 2, we also compare EcoOptiGen's performance with the other methods using hyperparameter search. We can see that with pruning, EcoOptiGen consistently outperforms the other non-pruning methods, and by a large margin on APPS and HumanEval. This confirms the effectiveness of the pruning technique. We further compare the number of trials searched by EcoOptiGen and the other search methods in Figure 4 of the appendix. We can observe that EcoOptiGen searches for 2-27× more trials under the same optimization budget, which helps it achieve the better result across all the tasks.

This study finds that, (a) compared to directly using the best evaluated configuration from HELM, one can potentially find much better configurations for a particular application by tuning the inference hyperparameters; and (b) pruning can vastly boost the optimization efficiency.

Table 3: Tuned results using different GPT-3.5 models for $B.i = 2K$, $B.o = 1M$. * indicates the model is the best for that dataset according to HELM.

Model	APPS	HumanEval	MATH	XSum
code-davinci-002	0.07*	0.819*	0.856	**0.198**
text-davinci-002	0.20	0.847	0.785	0.144*
text-davinci-003	**0.21**	**0.861**	**0.863***	0.119

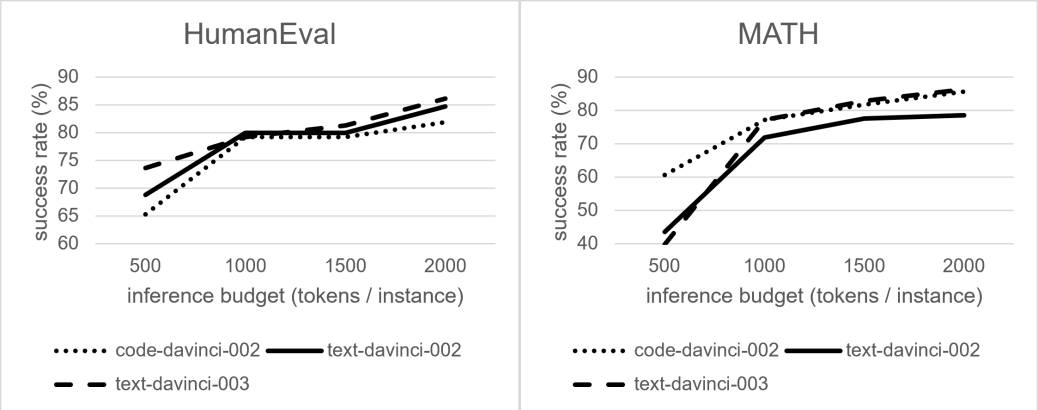

Figure 3: Results using GPT-3.5 models with varying inference budgets and a fixed optimization budget.

4.3 Effect of Inference Budget

To understand how the performance of EcoOptiGen is affected by the inference budget, we further vary the inference budget $B.i$ from 500 tokens to 2000 tokens on APPS, HumanEval and MATH, while fixing the total optimization budget $B.o = 1M$. Table 5 in the appendix displays the performance scores and Figure 5 displays the number of trials finished within the optimization budget. On APPS, the average number of input tokens is larger than 500, so no performance score is available in that case. On HumanEval, the optimized performance score increases from 0.653 to 0.819 as the inference budget increases from 500 to 2000. On MATH, the performance score increases from 0.398 to 0.863 as the inference budget increases from 500 to 2000. On APPS, the performance score drops from 0.10 to 0.07 when the inference budget is increased from 1500 to 2000. Based on Figure 5, we hypothesize that the performance drop is due to the decrease of the number of trials within the total optimization budget as the average cost per trial increases. We perform an additional experiment to test that hypothesis: we increase $B.o$ to 2M for $B.i = 2000$ on APPS. The optimized performance score then increases from 0.07 to 0.12, and the number of trials increases from 86 to 165. The result supports the hypothesis.

The takeaway message in this study is that EcoOptiGen is able to find significantly better configurations with increased inference budget, unless the optimization budget is not enough.

4.4 Effect of Model

In previous experiments, we fixed the model on each dataset according to the HELM benchmark. In this subsection, we first apply EcoOptiGen to other models in the GPT-3.5 family on each dataset, using an inference budget 2K and a total optimization budget of 1M. It was generally recommended by OpenAI to use "code-davinci-002" for code generation and "text-davinci-003" for other text generation (Shieh, 2022).

Table 3 summarizes the results. On APPS, HumanEval and MATH, text-davinci-003 performs the best after tuning. On XSum, code-davinci-002 performs the best after tuning. On three datasets, the best models after tuning are different from the best models according to the HELM benchmark,

as seen by the mismatches of the asterisk (HELM) and bold (after tuning) in each column. On APPS and HumanEval, we find that text-davinci-003 excels in the two code generation tasks. On MATH, code-davinci-002 can get close to text-davinci-003 after tuning. On XSum, we find that code-davinci-002 surprisingly outperforms the text-davinci models by a large margin. These results are quite contradictory with the common beliefs on model selection (Shieh, 2022).

We further plot the model performances with respect to each individual inference budget in Figure 3. We find that for HumanEval, text-davinc-003 model performs the best consistently. On MATH, although Table 3 shows that text-davinci-003 model slightly outperforms code-davinci-002, Figure 3 shows the latter is actually superior in the low inference budget range.

The takeaway message of this study is that with EcoOptiGen, the best performing model is not always the commonly recommended model. This reveals one of the benefits of hyperparameter optimization in avoiding suboptimal choices due to idiosyncrasies: Neither a newer model is certain to outperform an older one, nor the optimal choice between code-davinci and text-davinci models can be simply made according to whether the task involves code generation.

4.5 Limitations

For the summarization task the improvement by tuning is not as large as in code and math tasks. One potential reason is the ranking criterion for selecting one response from the *best_of* responses is not aligned with the final evaluation metric. Another limitation is we have not evaluated on other large language models and tasks.

4.6 Discussions and Future Work

It will be interesting future work to develop methods that help with understanding the optimized hyperparameter choices (ref. Table 6 in the appendix). It is also possible to further automate the tuning. For example, the current solution takes user-specified choices of prompts as the input search space. Automatically searching for optimal numbers and choices of demonstration examples can potentially result in more effective ways of using the inference budget.

5 Related Work

Text Generation with Large Language Models. Large language models are the workhorses of NLP and especially for text generation tasks. Thanks to techniques such as fine-tuning, supervision with human feedback and reinforcement learning (Ouyang et al., 2022), LLMs enjoy the capabilities of zero-shot or few-shot learning on many tasks (Chen et al., 2021; Wang et al., 2022). Following this innovation, the NLP research community has invested in improving the model performance during inference time. For example, lots of existing work is devoted to prompt learning (Liu et al., 2021; Mishra et al., 2021; Shieh, 2022). For text generation, different sampling algorithms are compared and found to perform similarly under matching hyperparameters like the temperature and probability mass (Branwen, 2020; Nadeem et al., 2020).

Automated Hyperparameter Optimization. Hyperparameter optimization methods for generic machine learning models have been studied for a decade (Feurer and Hutter, 2019; Bergstra et al., 2011). Since the training of deep neural networks is very expensive, new HPO methods have been proposed to reduce the cost required. Early stopping methods (Li et al., 2017) stop training with unpromising configurations at low fidelity (e.g., number of epochs) by comparing with other configurations' training performance at the same fidelity. Later, cost effective hyperparameter optimization were proposed. For example, in BlendSearch (Wang et al., 2021), an economical hybrid search strategy was proposed to handle heterogeneous evaluation cost and its effectiveness is demonstrated in fine-tuning a transformer model Turing-NLRv2. (Godbole et al., 2023) offers a generic playbook for tuning hyperparameters of training deep neural networks. Automated hyperparameter optimization is also studied specifically for NLP tasks, e.g., fine-tuning BERT-like language understanding models (Liu and Wang, 2021) and neural machine translation systems (Zhang and Duh, 2020).

6 Broader Impact Statement

Users of LLMs should take environmental impact into consideration when they determine inference budget and optimization budget. Our work helps reduce the energy consumption by helping users find deployable configurations with a budget cap, which is otherwise difficult to find and may result in increased environmental impact. When the budget is not set carefully, the optimization process could cause excessive energy consumption.

7 Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes] In Section 4.5.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Section 6.

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] In the abstract.

   (b) Did you include the raw results of running the given instructions on the given code and data? [No] The raw results have been deleted from a rented server.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [No] No such scripts are ever written.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] Search space: Table 1 in Section 3

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] Section 4.1

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] Section 4.2 contains several ablation methods to assess the impact of the pruning and the holistic tuning. Section 4.3 and Section 4.4 contain results with varying budgets and models.

(h) Did you use the same evaluation protocol for the methods being compared? [Yes] Yes Section 4.1

(i) Did you compare performance over time? [No]

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] We did perform multiple runs in a limited scale to verify the performance gain holds across different random seeds. But we did not do it for the full scale because of the monetary cost. With limited budget, we prioritize doing experiments at varying benchmarks and settings over varying random seeds. One additional issue is that we cannot control the randomness in the OpenAI API calls. So even if we use the same random seed the results will not be exactly the same.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] As explained above.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] The main resource used is by the API calls which is invisible.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [No] There is no hyperparameter to tune. The default inference and optimization budgets are chosen as the most natural numbers (1K and 1M tokens respectively). Section 4.3 and Section 4.4 contain results with varying budgets and models.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes] The datasets are from the HELM Liang et al. (2022), in particular we used APPS Hendrycks et al. (2021a), HumanEval Chen et al. (2021), MATH Hendrycks et al. (2021b), and XSum Narayan et al. (2018), we cited these assets throughout the paper.

(b) Did you mention the license of the assets? [Yes] It is mentioned in our code repository.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

(2023). OpenAI API. `https://beta.openai.com/docs/api-reference/completions/create`.

Bardenet, R. and Maillard, O.-A. (2015). Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*.

Branwen, G. (2020). Gpt-3 creative fiction.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chen, M. (2022). Leveraging natural supervision for language representation learning and generation. *arXiv preprint arXiv:2207.10617*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham.

Godbole, V., Dahl, G. E., Gilmer, J., Shallue, C. J., and Nado, Z. (2023). Deep learning tuning playbook. Version 1.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al. (2021a). Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021b). Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. (2022). Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Liu, X. and Wang, C. (2021). An empirical study on hyperparameter optimization for fine-tuning pre-trained language models. *arXiv preprint arXiv:2106.09204*.

Lucy, L. and Bamman, D. (2021). Gender and representation bias in gpt-3 generated stories. In *Proceedings of the Third Workshop on Narrative Understanding*, pages 48–55.

Mishra, S., Khashabi, D., Baral, C., Choi, Y., and Hajishirzi, H. (2021). Reframing instructional prompts to gptk's language. *arXiv preprint arXiv:2109.07830*.

Nadeem, M., He, T., Cho, K., and Glass, J. (2020). A systematic characterization of sampling algorithms for open-ended language generation. *arXiv preprint arXiv:2009.07243*.

Narayan, S., Cohen, S. B., and Lapata, M. (2018). Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

OpenAI (2023). Gpt-4 technical report.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., and Gulwani, S. (2022). Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*.

Provost, F., Jensen, D., and Oates, T. (1999). Efficient progressive sampling. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 23–32.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green ai. *Communications of the ACM*, 63(12):54–63.

Shieh, J. (2022). Best practices for prompt engineering with openai api.

Trummer, I. (2022). Codexdb: Synthesizing code for query processing from natural language instructions using gpt-3 codex. *Proceedings of the VLDB Endowment*, 15(11):2921–2928.

Wang, C., Wu, Q., Huang, S., and Saied, A. (2021). Economic hyperparameter optimization with blended search strategy. In *International Conference on Learning Representations*.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., et al. (2022). Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *URL https://arxiv. org/abs/2204.07705*.

Wu, Q., Wang, C., and Huang, S. (2021). Frugal optimization for cost-related hyperparameters. In *AAAI'21*.

Wullach, T. and Chazan, S. E. (2022). Don't be so sure! boosting asr decoding via confidence relaxation. *arXiv preprint arXiv:2212.13378*.

Zhang, X. and Duh, K. (2020). Reproducible and efficient benchmarks for hyperparameter optimization of neural machine translation systems. *Transactions of the Association for Computational Linguistics*, 8:393–408.

Zong, M. and Krishnamachari, B. (2022). Solving math word problems concerning systems of equations with gpt-3. In *Proceedings of the Thirteenth AAAI Symposium on Educational Advances in Artificial Intelligence*.

Table 4: Configuration for the HELM baseline.

Dataset	Config
APPS	{'model': 'code-davinci-002', 'prompt': '{D.in}', 'max_tokens': 600, 'temperature': 0.2, 'stop': ['''," – – –,"""",\n\n\n]}
HumanEval	{'model': 'code-davinci-002', 'prompt': '{D.in}', 'max_tokens': 600, 'temperature': 0.2, 'stop': [\nclass, \ndef, \nif, \nprint]}
MATH	{'model': 'text-davinci-003', 'prompt': 1-shot chain-of-thought demonstration*, 'max_tokens': 400, 'temperature': 0, 'stop': [###]}
XSum	{'model': 'text-davinci-002', 'prompt': 5-shot demonstration**, 'max_tokens': 400, 'temperature': 0.3, 'stop': [###]}

* 1-shot chain-of-thought demonstration in MATH:

```
Given a mathematics problem, determine the answer. Simplify your answer as much as possible.
###
Problem: What is the value of $\sqrt{3! \cdot 3!}$ expressed as a positive integer?
Answer: $\sqrt{3!\cdot3!}$ is equal to $\sqrt{(3!)^2}=3!=3\cdot2\cdot1=\boxed{6}$.
###
Problem: {D.in}
Answer:
```

** 5-shot demonstration in XSum:

```
###
Article: Almost one million people visited the city during the six-week festival period
over Christmas and Hogmanay. Organisers said almost 890,000 people visited the Edinburgh's
Christmas events in 2014/15, contributing \u00a3199.5m to the local economy. The three-day
Hogmanay celebrations attracted more than 150,000 people, creating an economic impact of
\u00a341.8m. Charlie Wood, Edinburgh's Christmas festival director, said: "This is great
news for Edinburgh. The revenue generated does not go to the events themselves, the event
organisers or to Edinburgh city council. "This is money, which is going to the businesses
of Edinburgh, be it retail, accommodation, food, drink, shopping and entertainment."

Summarize the above article in 1 sentence.
Edinburgh's winter festivals generated more than \u00a3241m for the city, according to
organisers.

###
Article: A firm in north Wales wants to bring the PooPrints service from the United States
to the UK with up to 15 councils reportedly interested in the scheme. Councils could make
owners in problem areas register their dogs to a database which involves a mouth swab taken.
Then, DNA could be taken from mess left on a street, path or grass and used to find a match
on the database. Gary Downie, managing director of Streetkleen Bio in Ruthin, Denbighshire,
believes local authorities can use new powers granted by the Antisocial Behaviour and
Policing Act 2014 to force dog owners to comply. "The purpose of the system is to get
cleaner, safer open spaces," he said. Councils the company is in talks with include
Kingston-upon-Thames in south-west London, Aberdeen and Cheshire East.

Summarize the above article in 1 sentence.
DNA in dog mess could be used to catch owners who fail to clear up their pet's mess.
```

Algorithm 1 Configuration evaluator with pruning

Inputs: configuration x, tuning data D, target average number of tokens $B.i$, valid and invalid trials X_{valid} and X_{invalid}.
Outputs: utility $U_x(D)$ if cost $C_x(D) \leq B.i$; 0 otherwise.
Initialization: obtain max_valid_n and min_invalid_n based on Eq. (1) and (2). $N \leftarrow x.n, n' \leftarrow 0, R \leftarrow []$
if $x.n \leq$ max_valid_n **then**
  $n \leftarrow x.n$
else if $x.n \geq$ min_invalid_n **then**
  return 0
else
  $n \leftarrow$ max_valid_n
end if
while True **do**
  $x.n \leftarrow n - n', k \leftarrow 1, k' \leftarrow 0$
  while True **do**
    for $i \in [k', k)$ **do**
      get responses from LLM for data point d_i using configuration x and append to R
    end for
    $\rho \leftarrow \begin{cases} (1 - \frac{k}{|D|})(1 + \frac{1}{k}) & 2k > |D| \\ 1 - \frac{k-1}{|D|} & 2k \leq |D| \end{cases}$
    if $C_x(D_k) > B.i(1 + .1\sqrt{\frac{\rho}{k}})$ **then**
      update X_{invalid} and return 0
    end if
    if $C_x(D_k) \leq B.i(1 - .1\sqrt{\frac{\rho}{k}})$ **and** ($n < N$ **or** $k = |D|$) **then**
      update X_{valid}
      if $n < N$ **then**
        $R \leftarrow [], n' \leftarrow 0$, break
      end if
    end if
    if $k < |D|$ **then**
      $k \leftarrow \min(2k, |D|)$
    else
      break
    end if
  end while
  if $n < N$ **then**
    if $R \neq []$ **then**
      $n' \leftarrow n$
    end if
    $n \leftarrow \min(2n, N)$
  else
    Return $U_x(D)$
  end if
end while

###
Article: The works at Nottingham Castle include a chalk portrait of St Anne, sketches of bodies and plants, plus some technical drawings. The artist made only around 20 paintings during his lifetime, including the Mona Lisa and The Last Supper, but left many more drawings. In total, there are almost 600 drawings by da Vinci in the Royal Collection. They were originally bound into a single album, thought to have been acquired in the 17th Century by Charles II. Experts believe Leonardo's drawings are the richest, most wide-ranging and most technically brilliant of any artist. The exhibition is on show at Nottingham Castle Museum and Art Gallery until 9 October.

Summarize the above article in 1 sentence.
Rare drawings by Leonardo da Vinci, which are part of the Queen's royal collection, have gone on show.

###
Article: Distill Ventures, which is part of the Diageo group, said it was investing an unspecified sum in Melbourne-based Starward Whisky. This marks the second whisky investment for Distill, which was set up to back early-stage brands and help them grow. Last week, it announced investment in Denmark-based Stauning Whisky. David Gates, Diageo's global head of premium core spirits, said: "Australian whisky has rightly been gaining increasing global recognition recently and Starward has developed a uniquely positioned whisky to capture this opportunity." Frank Lampen, co-founder of Distill Ventures, added: "The Starward team are exactly the types of entrepreneur we love working with. "Their vision for the future is really exciting and this investment will enable increased production of their signature single malts and continued development of their innovation pipeline." Last year Diageo had a 37%% share of the Scotch whisky market in terms of volumes.

Summarize the above article in 1 sentence.
Diageo, the world's biggest Scotch whisky distiller, has invested in an Australian distillery to help it expand into new export markets.

###
Article: It follows reports of dog fouling and damage at the Camperdown and Caird Park courses. Dogs can still be walked across the courses but not if owners are playing a round of the game at the time. A spokesman for Leisure and Culture Dundee said the rules were changed on 20 April. He said: "This change reflects the concerns of many players and staff about dog fouling and damage being caused to the courses, particularly greens and bunkers. "The new management rules, which do not affect the Right to Roam legislation, are clearly signed at the courses and on the Leisure and Culture Dundee website. "Most golf courses in Scotland do not allow players to bring dogs with them."

Summarize the above article in 1 sentence.
Golfers at Dundee's public courses have been banned from bringing their dogs with them after complaints from fellow players and staff.

###
Article: {D.in}

Summarize the above article in 1 sentence.

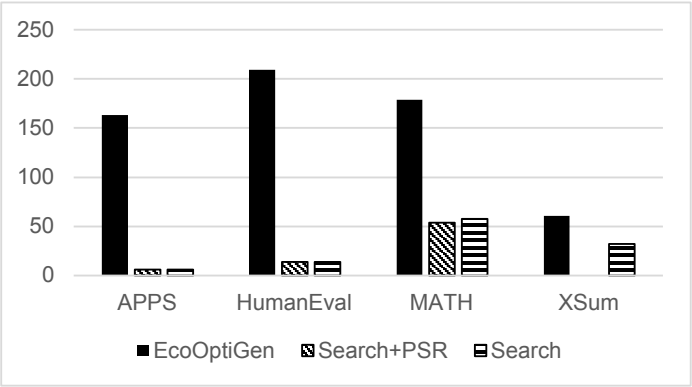

Figure 4: Number of trials within the total optimization budget.

Table 5: Results using varying inference budgets ($B.i$), with a fixed total optimization budget ($B.o$).

$B.i$ ($B.o = 1M$)	APPS	HumanEval	MATH
500	-	0.653	0.398
1000	0.05	0.792	0.771
1500	0.10	0.792	0.828
2000	0.07	0.819	0.863

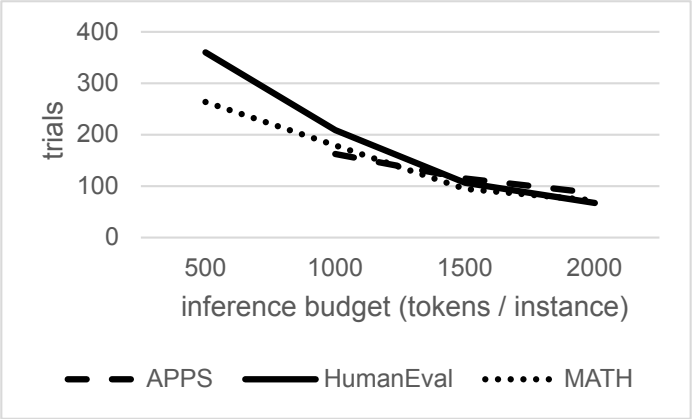

Figure 5: Number of trials with respect to varying inference budgets ($B.i$), with a fixed total optimization budget ($B.o$).

Table 6: Optimized hyperparameter configurations for text-davinci-003 with $B.i = 2K, B.o = 1M$.

Task	max_tokens	temperature_or_top_p	n
APPS	176	top_p: 0.982	15
HumanEval	517	top_p: 0.682	18
MATH	193	temperature: 1	26