Learning to Instruct: Fine-Tuning a Task-Aware Instruction Optimizer for Black-Box LLMs

Yunzhe Qi[1], Jinjin Tian[2], Tianci Liu[3], Ruirui Li[2], Tianxin Wei[1], Hui Liu[2], Xianfeng Tang[2], Monica Cheng[2], Jingrui He[1]
[1]University of Illinois Urbana-Champaign
[2]Amazon
[3]Purdue University
{yunzheq2,jingrui}@illinois.edu, jinjint@amazon.com

Abstract

The performance of Large Language Models (LLMs) critically depends on designing effective instructions, which is particularly challenging for black-box LLMs with inaccessible internal states. To this end, we introduce *Learning to Instruct*, a novel paradigm that formulates instruction optimization as an LLM fine-tuning objective for a white-box "instruction engineer" LLM, leveraging its rich learning capacity and vast pre-trained knowledge to enable efficient and effective instruction optimization. Within this paradigm, we propose *Automatic Instruction Optimizer* (AIO), a novel framework that fine-tunes a white-box LLM into a capable instruction engineer. AIO learns to optimize task-aware, human-comprehensible instructions by incorporating task nuances and feedback from the task-solving black-box LLM. To overcome the challenges of inaccessible black-box gradients and high API costs, AIO introduces a novel zeroth-order (ZO) gradient approximation mechanism guided by Thompson Sampling (TS), which reuses informative black-box LLM feedback for improved query efficiency. Extensive experiments show that AIO generally outperforms strong baselines in both effectiveness and efficiency, establishing *Learning to Instruct* as a promising new direction for black-box LLM instruction optimization.

1 Introduction

Large Language Models (LLMs) have shown remarkable capabilities. However, their strong performance generally relies on high-quality instructions that provide clear guidance and relevant context. Crafting such instructions is particularly challenging for powerful black-box (API-based) LLMs (Achiam et al., 2023; Anthropic, 2024), due to their inaccessible internal states. Meanwhile, optimal instruction strategies can vary significantly across different LLMs, model versions, and downstream tasks (Zhou et al., 2022; Khattab et al., 2023, 2022). Manual instruction crafting, while sometimes effective, can be costly, labor-intensive, and scales poorly with increasing task complexity (Brown et al., 2020; Reynolds and McDonell, 2021; Shin et al., 2020), highlighting the need for automated and task-aware mechanisms to optimize black-box LLM instructions (Sun et al., 2024).

Formally, instruction optimization can be viewed as a special case of adapting powerful black-box models to specific downstream tasks, by identifying effective task-specific instructions. To this end, a line of existing research casts a white-box LLM into a capable "instruction engineer" to instruct the black-box model through in-context learning frameworks (Zhou et al., 2022; Pryzant et al., 2023; Chen et al., 2024b; Lin et al., 2024). On the other hand, fine-tuning offers a mechanism for integral adaptation of the LLM. The capacity to adjust model parameters allows for a deeper assimilation of task-specific nuances and complex patterns (Liu et al., 2022; Mosbach et al., 2023; Han et al., 2024b). Such fine-grained adaptability is particularly important for *casting the "instruction engineer" LLM*, where accurately capturing and conveying task requirements is crucial for generating high-quality instructions, representing a promising yet underexplored direction of instruction optimization.

Motivated by these insights, we propose *Learning to Instruct*, a novel paradigm that re-frames instruction optimization as a fine-tuning objective for a white-box "instruction engineer" LLM. This formulation helps unlock the full potential of the optimizer by enabling parameter updates through methods like Parameter-Efficient Fine-Tuning (PEFT) (Hu et al., 2021a), allowing it to effectively and efficiently leverage pre-trained knowledge to capture subtle task nuances. This conceptual shift from in-context learning to a fine-tuning objective is our first contribution, offering a principled path to instruction optimization.

However, applying the *Learning to Instruct*

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 7707–7733
November 4-9, 2025

paradigm to black-box LLMs presents a significant technical hurdle: the inaccessible internal gradients of the task-solving LLM. This prevents direct, gradient-based fine-tuning of the white-box instruction engineer. While zeroth-order (ZO) approximation methods can bypass the need for direct gradients (Malladi et al., 2023), their reliance on random sampling can lead to potentially prohibitive query complexity, making them impractical given the generally expensive API costs of powerful black-box models.

To overcome this challenge and make our paradigm practical, we introduce the *Automatic Instruction Optimizer* (AIO) framework, centered on a novel and query-efficient ZO fine-tuning pipeline. Our approach tackles the potentially high cost of ZO approximation through two synergistic advances. First, we identify that the ZO approximation difficulty (e.g., slower convergence) increases with the number of parameters, i.e., the dimensionality of the gradients to be estimated (Chen et al., 2024a; Qiu and Tong, 2024). In light of this, AIO proposes to reduce the cost by estimating gradients only with respect to the output logits of the white-box LLM (which generate the instruction), rather than its entire parameter set. This consequently lowers the gradient estimation dimensionality, thus reducing the number of queries required. Second, to maximize the information gained from each query, AIO reformulates the selection of perturbation directions as a Contextual Bandit problem. It employs a Thompson Sampling (TS)-based strategy to intelligently reuse feedback from past queries, prioritizing directions most likely to improve the instruction. This TS-aided ZO method is our second core contribution, providing a feasible and efficient mechanism to realize the *Learning to Instruct* paradigm for black-box LLMs.

Next, we review **related works** in Sec. 2 and define **problem formulation** in Sec. 3. Sec. 4 details the **AIO framework** with TS-aided ZO method. **Experiments** showcasing AIO's effectiveness and efficiency are in Sec. 5, and we **conclude** in Sec. 6.

2 Related Works

Instruction Optimization for Black-box LLM. For black-box LLM instruction optimization, a line of existing works (Zhou et al., 2022; Prasad et al., 2022) performs instruction search based on manually defined criteria. Chen et al. (2024b); Lin et al. (2024); Hu et al. (2024) also apply an LLM with *frozen* parameters to generate instructions for the black-box LLM, and gradually update the generated instruction based on Bayesian Optimization (Frazier, 2018; Wang et al., 2023; Shahriari et al., 2015), Contextual Bandit approaches (Chu et al., 2011; Li et al., 2010; Valko et al., 2013; Zhou et al., 2020; Agrawal and Goyal, 2013; Zhang et al., 2021), or localized instruction optimization guided by Gaussian Process (Schulz et al., 2018). To adapt the vast pre-trained knowledge to the nuances of instruction generation, we instead fine-tune a white-box LLM to learn from task information and black-box LLM feedback for instruction optimization.

LLM-based Instruction Generation. LLM-based instruction optimization is an emerging research topic (Zhou et al., 2022; Ma et al., 2024; Schnabel and Neville, 2024), where LLMs are applied as instruction optimizer and their instructing strategies are gradually refined based on target model feedback. In particular, there are a series of works leveraging meta-prompts, which can be manually designed by humans (Yang et al., 2024), or optimized by LLMs (Tang et al., 2024). Meanwhile, Pryzant et al. (2023) perform in-context "Gradient Descent" on instructions based on interactions with an "instruction engineer" LLM. Fernando et al. (2023); Guo et al. (2024) propose in-context evolutionary algorithms to refine LLM-generated instructions. AIO alternatively leverages rich learning capacity of fine-tuned LLMs to model the relationship between task information and black-box LLM feedback, without relying on human experts.

3 Problem Formulation

Under the proposed *Learning to Instruct* paradigm, given a target task $\mathcal{T}$, we involve two LLMs: (1) A black-box LLM $\mathcal{F}_B(\cdot)$ is for task-solving, i.e., generating answers for task queries. The black-box LLM is treated as part of learning objective, as we aim to optimize instructions that enhance its performance. (2) A white-box LLM $\mathcal{F}_W(\cdot;\mathbf{\Theta}_W)$ with trainable parameters $\mathbf{\Theta}_W$ will generate and refine human-comprehensible instructions, with information from task $\mathcal{T}$ and feedback from $\mathcal{F}_B(\cdot)$.

Task data and instruction generation. Suppose the target task $\mathcal{T}$ is associated with three separate data collections (i.e., query-answer pairs (X, Y)): (1) Training data (i.e., task exemplars) $\mathcal{D}_{\text{Train}}$ guides the white-box LLM in generating and optimizing task-specific instructions; (2) Validation data $\mathcal{D}_{\text{Valid}}$ is used for performance evaluation during optimization; (3) Final evaluation is conducted on a separate test set. Let $\mathcal{F}_W(\mathcal{D}_{\text{Train}};\mathbf{\Theta}_W)$

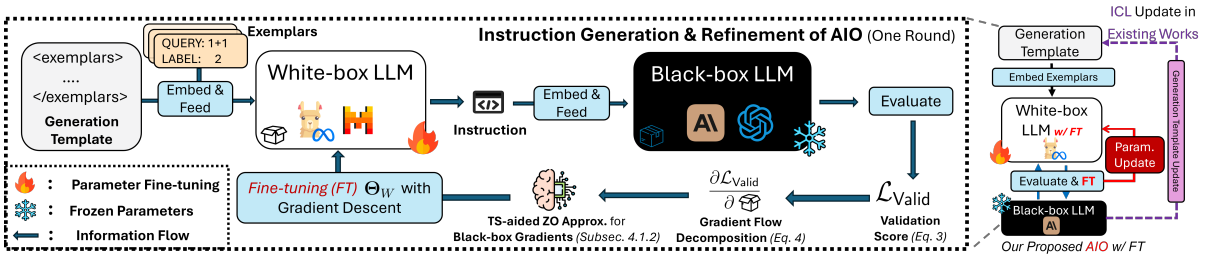


Figure 1: AIO Pipeline with information flow: The white-box LLM generates an instruction from exemplars, evaluated to produce validation score. We then decompose the inaccessible black-box LLM gradients from the main gradient flow, and approximate them using our proposed Thompson Sampling (TS)-aided zeroth-order (ZO) method. Finally, we update white-box LLM parameters via Gradient Descent (GD) that is compatible with PEFT.

be the instruction generated by the white-box LLM, with exemplars $\mathcal{D}_{\text{Train}}$ and tunable parameters $\boldsymbol{\Theta}_W$. Black-box LLM then generates output $\mathcal{F}_B\big([\mathcal{F}_W(\mathcal{D}_{\text{Train}}; \boldsymbol{\Theta}_W); X]\big)$ for query X, where $[\cdot;\cdot]$ operation embeds query X to instruction.

Learning objective. Given exemplars $\mathcal{D}_{\text{Train}}$ and an evaluation function $\mathcal{L}(\cdot,\cdot)$, instruction optimization is framed as a fine-tuning objective. We aim to find the optimal white-box LLM parameters $\boldsymbol{\Theta}_W$ that optimizes task-specific score.

$$\min_{\boldsymbol{\Theta}_W} \Big[\mathbb{E}_{(X,Y)\sim\mathcal{T}}\big[\mathcal{L}\big(\mathcal{F}_B\big([\phi(\boldsymbol{\Theta}_W); X]\big), Y\big)\big]\Big], \tag{1}$$

with a shorthand $\phi(\boldsymbol{\Theta}_W) := \mathcal{F}_W(\mathcal{D}_{\text{Train}}; \boldsymbol{\Theta}_W)$ for generated instruction. Intuitively, we utilize fine-tuning to guide how the white-box LLM comprehends task exemplars and composes task-specific instructions, based on task-solver black-box LLM feedback during optimization.

4 Automatic Instruction Optimizer (AIO)

Given our learning objective (Eq. 1), a straightforward approach is to update the white-box LLM parameters $\boldsymbol{\Theta}_W$, using gradients computed from instruction evaluation results. However, the nested black-box LLM makes direct back-propagation towards $\boldsymbol{\Theta}_W$ via the chain rule infeasible.

AIO Overview. Shown in Fig. 1, to achieve efficient and effective white-box LLM fine-tuning for instruction optimization, AIO involves two major components. (1) *Instruction Generation & Evaluation*: Given the exemplars $\mathcal{D}_{\text{Train}}$, the white-box LLM with parameters $\boldsymbol{\Theta}_W$ generates an instruction $\phi(\boldsymbol{\Theta}_W) := \mathcal{F}_W(\mathcal{D}_{\text{Train}}; \boldsymbol{\Theta}_W)$ as in Eq. 1. The instruction is then evaluated based on the black-box LLM output, which produces validation score. (2) *White-box "instruction engineer" LLM Fine-tuning*: With the validation score, we decompose the gradient flow towards $\boldsymbol{\Theta}_W$ into two components: (i) white-box LLM gradients, which can be obtained through back-propagation, and (ii) inaccessible black-box LLM gradients, approximated by our proposed TS-aided ZO method (Subsec. 4.1.2), which enables efficient and effective gradient approximation. Pseudo-code is presented in Alg. 1.

Validation score. Given our optimization objective in Eq. 1, since a comprehensive overview of the task distribution $\mathcal{T}$ can be inaccessible, we evaluate the performance of the generated instruction on the validation data $\mathcal{D}_{\text{Valid}}$, as per validation score

$$\mathcal{L}_{\text{Valid}}\big(\phi(\boldsymbol{\Theta}_W)\big) := \frac{1}{|\mathcal{D}_{\text{Valid}}|} \sum_{(X,Y)\in\mathcal{D}_{\text{Valid}}} \mathcal{L}\left(\mathcal{F}_B\big([\phi(\boldsymbol{\Theta}_W); X]\big), Y\right), \tag{2}$$

where $\phi(\boldsymbol{\Theta}_W)$ is the instruction generated by the white-box LLM $\mathcal{F}_W(\cdot; \boldsymbol{\Theta}_W)$ as in Eq. 1.

4.1 TS-aided ZO Gradient Approximation

To tackle this challenge of inaccessible black-box LLM gradients, by applying chain rule on Eq. 2, we can decompose the gradient flow with respect to white-box LLM parameters $\boldsymbol{\Theta}_W$ into two components: (1) gradients involving the black-box LLM; (2) and white-box LLM gradients that can be obtained by back-propagation, as

$$\frac{\partial\left[\frac{1}{|\mathcal{D}_{\text{Valid}}|}\sum_{(X,Y)\in\mathcal{D}_{\text{Valid}}}\mathcal{L}\big(\mathcal{F}_B\big([\phi(\boldsymbol{\Theta}_W); X]\big), Y\big)\right]}{\partial \boldsymbol{\Theta}_W} = \underbrace{\frac{\partial\big[\mathcal{L}_{\text{Valid}}(\phi(\boldsymbol{\Theta}_W))\big]}{\partial\phi(\boldsymbol{\Theta}_W)}}_{\text{Black-box LLM Gradients}} \times \underbrace{\frac{\partial\phi(\boldsymbol{\Theta}_W)}{\partial\boldsymbol{\Theta}_W}}_{\text{White-box LLM Gradients}}, \tag{3}$$

by plugging in the definition of $\mathcal{L}_{\text{Valid}}\big(\phi(\boldsymbol{\Theta}_W)\big)$ from Eq. 2. For white-box LLM gradients, we can also integrate back-propagation with PEFT techniques, such as LoRA (Hu et al., 2021a), to improve

fine-tuning efficiency while preserving strong performance. Meanwhile, recall that directly computing the first term on the right-hand side via back-propagation is infeasible, since the parameters and gradients of the nested black-box LLM $\mathcal{F}_B(\cdot)$ are inaccessible. We propose to leverage ZO gradient approximation to address this challenge.

4.1.1 Zeroth-order (ZO) Black-box LLM Gradient Approximation

Zeroth-order gradient approximation has been proved effective and efficient for LLM fine-tuning (Malladi et al., 2023), achieving satisfactory results with only forward (inference) passes of LLMs. This makes the ZO method a promising approach for approximating black-box LLM gradients.

Analogous to existing ZO approximation works (e.g., Nesterov and Spokoiny (2017); Ghadimi et al. (2016); Duchi et al. (2015); Shu et al. (2023); Malladi et al. (2023)), we can first assume a *linear optimization landscape* around white-box LLM output ϕ. With validation score of Eq. 2, it leads to $\mathcal{L}_{\text{Valid}}(\phi+\boldsymbol{z}) \approx [\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi)]^\intercal \cdot \boldsymbol{z} + \mathcal{L}_{\text{Valid}}(\phi)$, with $\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi) := \partial \mathcal{L}_{\text{Valid}}(\phi)/\partial\phi$. Here, $\boldsymbol{z}$ is a small perturbation applied to the predicted next-token distribution, for all tokens in the output ϕ, specifically on *LLM-header output logits* (i.e., predicted logits over the vocabulary). We include supplementary explanations for auto-regressive generation and perturbation in Appendix D.1. Due to linearity, this formulation holds as the gradients $\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi)$ remain constant for all ϕ within the *linear* landscape.

Black-box LLM Gradient Approximation. Next, we propose to impose small token-level perturbations to gather information about the optimization landscape, as even slight perturbations in token-level outputs can effectively influence the auto-regressive generation process (Han et al., 2024a). Inspired by Malladi et al. (2023), we can approximate black-box LLM gradients by

$$\nabla_\phi \big[\mathcal{L}_{\text{Valid}}\big(\phi(\boldsymbol{\Theta}_W)\big)\big] \approx \frac{\mathcal{L}_{\text{Valid}}\big(\phi(\boldsymbol{\Theta}_W) + \epsilon\boldsymbol{z}\big) - \mathcal{L}_{\text{Valid}}\big(\phi(\boldsymbol{\Theta}_W) - \epsilon\boldsymbol{z}\big)}{2\epsilon} \cdot \boldsymbol{z} \tag{4}$$

where we apply $\mathcal{L}_{\text{Valid}}(\cdot)$ as the evaluation function (Eq. 2), and $\epsilon \in \mathbb{R}^+$ controls the perturbation intensity. Here, $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I}) \in \mathbb{R}^d$ stands for a random Gaussian perturbation vector applied to each token of the output ϕ, and it preserves the same dimensionality as the token-level output dimensionality. Consequently, d will correspond to the vocabulary size of the white-box LLM, and the first term on the right-hand side of Eq. 3 can be approximated with only black-box LLM forward passes, without accessing its internal gradients or parameters.

Remark 4.1 (Gradient Approximation Formulation)**.** We propose to approximate black-box LLM gradients, rather than using ZO method to directly estimate whole gradient flow (left-hand side of Eq. 3), as the error of ZO methods can grow along with target dimensionality (Chen et al., 2024a; Qiu and Tong, 2024). As the number of white-box LLM parameters $\boldsymbol{\Theta}_W$ generally far exceeds its output dimensionality (i.e., vocabulary size), we only approximate $\partial\big[\mathcal{L}_{\text{Valid}}(\phi(\boldsymbol{\Theta}_W))\big]/\partial\phi(\boldsymbol{\Theta}_W)$ to reduce estimation error (ablation study: Appendix C.5).

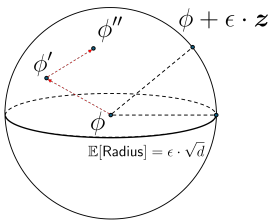
Here, one potential drawback is that as perturbation vectors $\boldsymbol{z}$ are randomly sampled (Eq. 4), gradient perturbation directions within the optimization landscape will be random and potentially inefficient (Cai et al., 2022) (e.g., sampled $\boldsymbol{z}$ being orthogonal to the target gradient). Thus, we propose reusing collected feedback, by formulating above ZO-based gradient approximation process as a sequential decision-making problem, and utilize Contextual Bandit techniques to effectively determine which perturbation directions are beneficial and worth selecting, in terms of improving instruction quality and black-box LLM performance.

4.1.2 TS-aided gradient direction selection

Recall that ZO-based gradient approximation methods (e.g., Nesterov and Spokoiny (2017); Ghadimi et al. (2016); Duchi et al. (2015); Malladi et al. (2023)) commonly assume a linear optimization landscape around the objective, as in Eq. 4.

Leveraging the linear optimization landscape. In this case, as long as the updated ϕ stays within the assumed linear landscape, we can intuitively formulate the ZO approximation into a *sequential decision-making process*, and leverage collected information of the linear landscape to select informative perturbation directions $\boldsymbol{z}$ for Eq. 4.

Contextual Bandit algorithms (Li et al., 2010; Agrawal and Goyal, 2013; Qi et al., 2023a, 2024) are designed to identify the optimal choice among a set of candidate arms (i.e., actions) based on arm contextual information, while addressing the exploitation-exploration dilemma in sequential decision-making processes (Auer et al., 2002; Qi et al., 2023b). Under Contextual Bandit settings and the gradient approximation formulation in Eq. 4, we define arm reward $r = \big[\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi)\big]^\intercal \boldsymbol{z}$



$$\mathcal{L}_{\text{Valid}}(\phi + \epsilon \cdot \boldsymbol{z}) \approx [\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi)]^\intercal (\epsilon \boldsymbol{z}) + \mathcal{L}_{\text{Valid}}(\phi)$$

Figure 2: A linear optimization landscape (as a sphere) around output ϕ. With vector $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$, the expected radius is $\epsilon\sqrt{d}$, for the L_2 distance of average token-level output. Updated ϕ', ϕ'' can remain within this landscape, motivating our TS-based ZO method.

for each perturbation direction $\boldsymbol{z} \in \mathbb{R}^d$ (i.e., a *candidate arm* in Contextual Bandit), such that it can be approximated by

$$r \approx \frac{\mathcal{L}_{\text{Valid}}(\phi + \epsilon \boldsymbol{z}) - \mathcal{L}_{\text{Valid}}(\phi - \epsilon \boldsymbol{z})}{2\epsilon}, \quad (5)$$

with respect to current white-box LLM output ϕ.

Here, the arm reward quantifies the benefit of perturbing in direction $\boldsymbol{z}$ (i.e., updating output ϕ towards direction $-\boldsymbol{z}$). Given our arm reward formulation within a linear landscape, we employ a *linear* Thompson Sampling (TS) model to select perturbation directions (i.e., arms) accordingly. The TS model parameters, with dimensionality d, are $\boldsymbol{\theta} \in \mathbb{R}^d$. Analogous to Agrawal and Goyal (2013); Zhang et al. (2021), we initialize the TS model parameters with a prior $\mathcal{N}(0, \boldsymbol{I})$, progressively refine the posterior using collected information.

With our arm reward formulation in Eq. 5 and the nature of linear TS, TS parameters $\boldsymbol{\theta}$ serve as an estimate of the uniform gradients $\nabla_\phi \mathcal{L}_{\text{Valid}}(\phi)$ within the optimization landscape. This enables the use of a *highly efficient* linear TS algorithm for rapid arm selection and parameter updates, and we gradually refine our arm selection strategy with collected black-box LLM feedback.

TS-aided perturbation direction (arm) selection. We consider a T-round fine-tuning process. Let pre-trained white-box LLM parameters without fine-tuning be $\boldsymbol{\Theta}_0 := \boldsymbol{\Theta}_W$, with initial generated instruction $\phi_0 := \phi(\boldsymbol{\Theta}_0)$. We let $\boldsymbol{\Theta}_{t-1}, t \in [T]$ be white-box LLM parameters *before* t-th round fine-tuning. In each round t, we first sample $K \in \mathbb{N}^+$ candidate arms (i.e., perturbation directions) $\mathcal{Z}_t$ for selection from standard Gaussian distribution:

$$\mathcal{Z}_t := \{\boldsymbol{z}_{t,1}, \boldsymbol{z}_{t,2}, \ldots, \boldsymbol{z}_{t,K}\} \sim \mathcal{N}(0, \boldsymbol{I}), \quad (6)$$

which helps control the arm context norm magnitude, while ensuring the direction randomness. Initial TS model parameters are instantiated by sampling from the prior $\boldsymbol{\theta}_0 \sim \mathcal{N}(0, \boldsymbol{I})$. We also let $\boldsymbol{\theta}_{t-1}$ represent the TS model parameters in round t before the refinement. We will discuss later how to update TS parameters using a refined TS parameter posterior in Eq. 8. Following lines 4-6 of Alg. 1, for these K candidate arms, we formulate *estimated rewards* as the inner product $\boldsymbol{z}_{t,k}^\intercal \boldsymbol{\theta}_{t-1}, \forall \boldsymbol{z}_{t,k} \in \mathcal{Z}_t$, and select the top-$B$, $B \ll K$, arms with the *highest estimated rewards* to form collection $\widetilde{\mathcal{Z}}_t \subset \mathcal{Z}_t$. Setting $B = 3$ is sufficient in our experiments to achieve satisfactory performance while maintaining query efficiency. The chosen arms $\widetilde{\mathcal{Z}}_t$ represent perturbation directions beneficial for optimizing the validation score. We empirically compare against ZO methods with random perturbations in Appendix E.1, showcasing that our TS-aided ZO approach is more effective for the optimization.

Querying arm rewards. Next, we query the black-box LLM (i.e., reward oracle) for validation score results $\mathcal{L}_{\text{Valid}}$ to obtain rewards for the chosen arms $\widetilde{\mathcal{Z}}_t$. Using the shorthand $\phi_{t-1} := \phi(\boldsymbol{\Theta}_{t-1})$, for each chosen arm $\boldsymbol{z}_{t,k} \in \widetilde{\mathcal{Z}}_t$, we calculate its arm reward by following Eq. 5, leading to

$$r_{t,k} = \frac{\mathcal{L}_{\text{Valid}}(\phi_{t-1} + \epsilon \boldsymbol{z}_{t,k}) - \mathcal{L}_{\text{Valid}}(\phi_{t-1} - \epsilon \boldsymbol{z}_{t,k})}{2\epsilon} \quad (7)$$

which measures the benefit of involving direction (arm) $\boldsymbol{z}_{t,k}$ for gradient approximation.

Fine-tuning White-box LLM. With each chosen arm (perturbation direction), the queried validation score results are also *recycled* to approximate the black-box LLM gradients, based on Eq. 4. Here, analogous to Malladi et al. (2023), estimated gradient results from the B chosen perturbation directions (arms) are averaged for a more accurate approximation. Then, white-box LLM parameters are updated based on Gradient Descent (GD), with the gradient flow (Eq. 3) and the estimated black-box LLM gradients (Eq. 4).

Validating optimization landscape. With the queried arm rewards (Eq. 7), we then update TS parameters. Recall that we operate within a linear optimization landscape, if generated instructions do not significantly deviate from previous ones. Therefore, we apply a threshold parameter $\beta > 0$ to quantify the landscape. For the optimized instruction ϕ_t in round t, we apply condition $\|\phi_t - \phi_{\text{Check}}\| > \beta$ to verify whether the assumed linear landscape remains valid. We set β to a large value in our experiments, allowing TS to sufficiently leverage col-

lected information. The initial checkpoint ϕ_{Check} is set as the instruction ϕ_0. The output distance is computed as the L_2 distance, between the averaged token-level logits of the current output ϕ_t and those of the checkpoint ϕ_{Check}, inspired by Joshi et al. (2023); Manakul et al. (2023). Collected records are initialized as an empty set $\Omega_0 \leftarrow \emptyset$.

Updating TS parameters $\boldsymbol{\theta}$. In each round $t \in [T]$, we consider two scenarios (lines 9-16, Alg. 1). *Scenario (1)*: If white-box LLM output is far enough from the checkpoint, s.t. $\|\phi_t - \phi_{\text{Check}}\| > \beta$, our current knowledge can be invalid because current white-box LLM output has significantly deviated from the checkpoint. In this case, we set the new checkpoint as ϕ_t, discard collected records, and reinitialize TS parameters $\boldsymbol{\theta}_t$ from the prior $\mathcal{N}(0, \boldsymbol{I})$. *Scenario (2)*: Otherwise, if the distance is small enough s.t., $\|\phi_t - \phi_{\text{Check}}\| \leq \beta$, the chosen arms and their true rewards from this round will be integrated into collected records Ω_t. Afterwards, analogous to existing TS methods (Agrawal and Goyal, 2013; Zhang et al., 2021), with exploration variance $\nu \geq 0$, covariance matrix $\boldsymbol{\Sigma}_t := \boldsymbol{I} + \sum_{(\boldsymbol{z},r)\in\Omega_t} \boldsymbol{z} \cdot \boldsymbol{z}^\intercal$, and reward vector $\boldsymbol{b}_t := \sum_{(\boldsymbol{z},r)\in\Omega_t} \boldsymbol{z} \cdot r$, we update the posterior:

$$\mathcal{N}(\boldsymbol{\Sigma}_t^{-1}\boldsymbol{b}_t,\ \nu \cdot \boldsymbol{\Sigma}_t^{-1}). \tag{8}$$

Finally, we sample updated TS parameters $\boldsymbol{\theta}_t$ from the posterior.

Remark 4.2 (Reducing Computational Costs)**.** To reduce computational costs, motivated by Johnson-Lindenstrauss (JL) Lemma (Johnson and Lindenstrauss, 1984), we adopt random Gaussian projection to map d-dimensional arm contexts into a lower-dimensional space (Matoušek, 2008; Larsen and Nelson, 2017), where we select from candidates $\mathcal{Z}_t$ with TS. Similar ideas are also applied to reduce soft prompt dimension (Chen et al., 2024b; Lin et al., 2024). Meanwhile, for matrix inversion $\boldsymbol{\Sigma}_t^{-1}$ (Eq. 8), we apply Sherman-Morrison formula (Bartlett, 1951; Maponi, 2007), to avoid direct matrix inversions. Details are in Appendix D.2.

4.2 Summary of AIO Workflow

In Algorithm 1, for each round $t \in [T]$, we sample K candidate arms (gradient directions) $\mathcal{Z}_t$ (line 4). Our TS model then estimates their rewards (quantifying benefits of perturbation directions). To minimize API costs, only $B \ll K$ arms $\widetilde{\mathcal{Z}}_t \subset \mathcal{Z}_t$ will be selected (lines 4-5). Next, we query the black-box LLM to obtain rewards of the chosen arms

Algorithm 1 AIO Workflow

1: **Input:** T, K, B, β.
2: **Init.:** $\boldsymbol{\theta}_0 \sim \mathcal{N}(0, \boldsymbol{I})$. $\boldsymbol{\Theta}_0 \leftarrow \boldsymbol{\Theta}_W$. Checkpoint $\phi_{\text{Check}} \leftarrow \phi(\boldsymbol{\Theta}_0)$. TS records $\Omega_0 \leftarrow \emptyset$.
3: **for** each optimization round $t \in [T]$ **do**
▷ TS-aided ZO Direction Selection
4: Sample K directions (arms) $\mathcal{Z}_t$ (Eq. 6).
5: $\widetilde{\mathcal{Z}}_t \leftarrow \operatorname{argmax}_{\widetilde{\mathcal{Z}}_t \subset \mathcal{Z}_t,\ |\widetilde{\mathcal{Z}}_t|=B} \left[\sum_{\boldsymbol{z}_{t,k}\in\widetilde{\mathcal{Z}}_t} \boldsymbol{z}_{t,k}^\intercal \boldsymbol{\theta}_{t-1}\right]$.
▷ White-box LLM Fine-tuning
6: Query reward r for each chosen arm $\boldsymbol{z} \in \widetilde{\mathcal{Z}}_t$ (Eq. 7 and Eq. 2).
 # Only query $\widetilde{\mathcal{Z}}_t$ to reduce API cost.
7: Fine-tune white-box LLM parameters to $\boldsymbol{\Theta}_t$ via gradient flow (Eqs. 3-4).
8: Generate updated instruction $\phi_t := \phi(\boldsymbol{\Theta}_t)$.
▷ Updating Linear TS Model
9: **if** $\|\phi_t - \phi_{\text{Check}}\| > \beta$ **then**
10: Reset TS prior $\mathcal{N}(0, \boldsymbol{I})$. Reset checkpoint $\phi_{\text{Check}} \leftarrow \phi_t$, and TS records $\Omega_t \leftarrow \emptyset$.
11: Sample TS parameters $\boldsymbol{\theta}_t \sim \mathcal{N}(0, \boldsymbol{I})$.
12: **else**
13: With chosen arms $\widetilde{\mathcal{Z}}_t$ and their rewards, update $\Omega_t \leftarrow \Omega_{t-1} \cup \left[\bigcup_{\boldsymbol{z}\in\widetilde{\mathcal{Z}}_t} (\boldsymbol{z}, r)\right]$.
14: Update posterior $\mathcal{N}(\boldsymbol{\Sigma}_t^{-1}\boldsymbol{b}_t,\ \nu\boldsymbol{\Sigma}_t^{-1})$.
15: Update $\boldsymbol{\theta}_t \sim \mathcal{N}(\boldsymbol{\Sigma}_t^{-1}\boldsymbol{b}_t,\ \nu\boldsymbol{\Sigma}_t^{-1})$.
16: **end if**
17: **end for**

(line 6), and perform GD to fine-tune the white-box LLM parameters with the gradient flow described in Eqs. 3 and 4 (line 7). If white-box LLM's output significantly differs from its checkpoint, we reset records and the TS parameter distribution (line 10). Otherwise, the TS parameter posterior is updated (lines 13-14). Finally, we sample the TS parameters $\boldsymbol{\theta}_t$ accordingly (lines 11, 15).

5 Experiments

For our experiments, `Llama-3-8B-Instruct` (Dubey et al., 2024) is applied as our tunable white-box LLM $\mathcal{F}_W(\cdot; \boldsymbol{\Theta}_W)$, and `Claude-3-Sonnet` (Anthropic, 2024) as black-box LLM $\mathcal{F}_B(\cdot)$. An outline for our results in main body: (1) zero-shot instruction induction experiments on 15 tasks are in Subsec. 5.1; (2) analysis of API token costs and instruction trajectory is in Subsec. 5.2; (3) a case study on AIO's transferability is in Subsec. 5.3. Due to page limitations, we include detailed experimental settings in Appendix B, and comprehensive

Tasks \ Methods	Black-box LLM		White-box LLM w/o FT			White-box LLM w/ FT (Ours)		
	APE	ProTeGi	InstructZero	INSTINCT	ZOPO	AIO + LoRA	AIO + LP	AIO w/ Full FT
antonyms	0.893	0.861	0.843	0.881	0.853	0.898	0.857	**0.901**
sentiment	0.911	0.928	0.941	0.920	0.921	0.947	**0.967**	0.949
larger_animal	0.914	0.932	0.827	0.857	0.842	**0.950**	0.945	0.912
taxonomy_animal	0.491	0.970	0.598	0.782	0.887	0.935	0.979	**0.983**
object_counting	0.319	**0.550**	0.522	0.537	0.503	0.479	0.401	0.543
navigate	0.580	0.624	0.556	0.577	0.604	0.627	0.623	**0.644**
winowhy	0.022	0.703	0.671	**0.725**	0.026	0.635	0.646	0.622
implicatures	0.806	0.826	0.816	0.837	0.784	**0.849**	0.836	0.811
logical_fallacy_detection	0.820	0.826	0.790	0.826	0.829	0.836	0.824	**0.868**
hyperbaton	0.515	0.499	0.467	0.502	0.512	0.527	0.518	**0.538**
epistemic_reasoning	0.604	0.459	0.667	0.580	0.678	0.719	**0.784**	0.766
movie_recommendation	0.348	0.847	0.895	0.866	0.880	0.883	0.857	**0.902**
timedial	0.532	0.718	0.786	0.712	0.735	0.759	0.734	**0.814**
presuppositions_as_nli	0.458	0.488	0.503	0.482	0.479	0.493	0.486	**0.523**
question_selection	0.712	0.667	**0.718**	0.605	0.656	0.622	0.628	0.648
Average Rank	6.3	4.4	5.0	5.3	5.4	3.1	4.1	**2.4**

Table 1: Zero-shot instruction induction results. For each task, **bold** number is the best result, while underlined number is the second-best one. AIO and its two variants can outperform four baselines on 12 out of a total of 15 tasks. Average rank results of each method across tasks are reported.

complementary experiments in Appendix C.

5.1 Zero-shot Instruction Induction

Analogous to existing instruction optimization works (Zhou et al., 2022; Chen et al., 2024b; Lin et al., 2024), our empirical analysis involves 15 different tasks, including instruction induction tasks from Honovich et al. (2022), and more challenging reasoning tasks from BigBench (bench authors, 2023). We consider two baselines using black-box LLMs for instruction optimization: (1) APE (Zhou et al., 2022) and (2) ProTeGi (Pryzant et al., 2023), and three baselines utilizing white-box LLMs: (3) InstructZero (Chen et al., 2024b), (4) INSTINCT (Lin et al., 2024), (5) ZOPO (Hu et al., 2024).

AIO with PEFT. Recall that our AIO is compatible with many existing PEFT methods for efficient white-box LLM fine-tuning. Therefore, we include empirical results of incorporating Linear Probing (LP) (Kumar et al., 2022) and LoRA (Hu et al., 2021a) to our proposed AIO, denoted as "AIO + LP" and "AIO + LoRA" respectively. "AIO + LP" fine-tunes just $\sim 6.54\%$ and "AIO + LoRA" only fine-tunes $\sim 0.04\%$ of the white-box LLM parameters, highlighting their parameter efficiency.

Main results. In Table 1, our proposed AIO generally outperforms strong baselines on challenging reasoning tasks from BigBench (bench authors, 2023), owing to its *Learning to Instruct* paradigm and TS-aided ZO fine-tuning process. *An ablation study* in Appendix C.5 further supports the *effectiveness of AIO's components*. In particular, our lightweight PEFT variants demonstrate strong performance while requiring significantly fewer tunable parameters. "AIO + LoRA", despite its remarkable parameter efficiency (fine-tuning only $\sim 0.04\%$ of parameters), can generally outperform

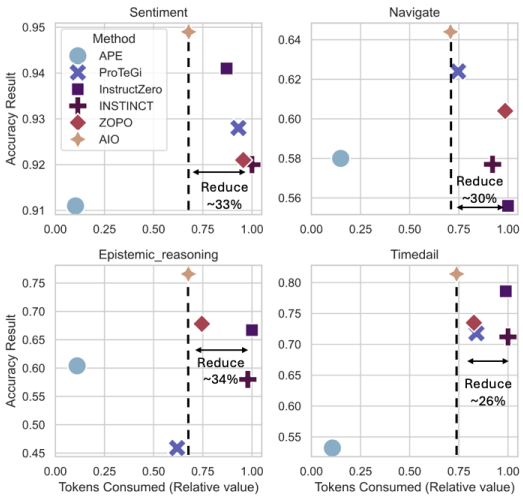


Figure 3: Token consumption vs. performance. Token consumption results are normalized into $[0, 1]$ range.

baselines and deliver impressive performance. This highlights not only the overall effectiveness of the AIO framework, but also its excellent efficiency-to-performance balance. Meanwhile, AIO with LoRA can also achieve generally stronger performance and *consistently outperforms the baselines* for *few-shot induction* (Appendix C.2), highlighting AIO's capabilities for effective instruction optimization under different scenarios.

5.2 API Efficiency and Instruction Trajectory

For Claude-3, API query costs are charged on a token-basis. As shown in Fig. 3, AIO can maintain a good balance between token cost and performance, starting from early optimization stages when small amounts of tokens are consumed. InstructZero and INSTINCT generally have higher token consumption than AIO. Applying white-box LLMs with frozen parameters, they primarily rely on in-context learning with a tunable soft prompt

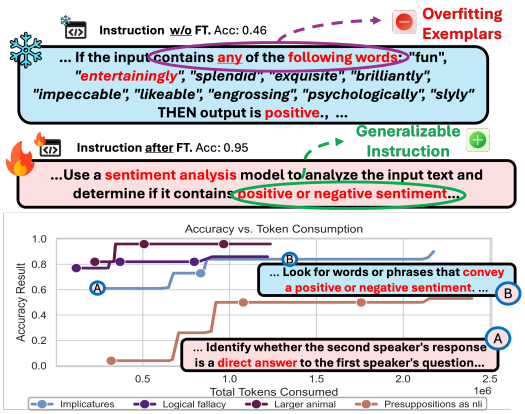


Figure 4: **[Upper]** Two generated instructions for "Sentiment" task: w/o FT and after FT. Instructions with FT generalize to task contexts instead of over-fitting task exemplars. **[Lower]** Token consumption vs. best accuracy results for certain token consumption levels on four tasks, with two instruction snippets for "Implicatures".

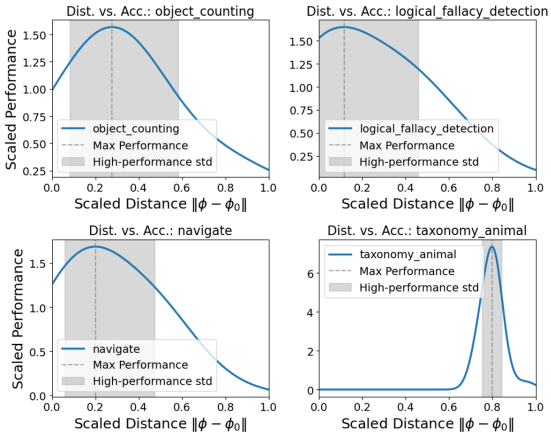


Figure 5: Performance vs. scaled instruction distance from the original ϕ_0. Shaded standard deviation regions highlight where high-performance instructions are concentrated.

(vector) within the generation template, which is selected by a kernel-based learner or a small neural model. In this case, their in-context learning capacity can limit full utilization of black-box LLM feedback, resulting in higher token costs. In contrast, AIO leverages white-box LLM fine-tuning to better exploit black-box LLM feedback for more efficient instruction optimization.

Meanwhile, in the upper figure of Fig. 4, without fine-tuning, $\mathcal{F}_W$ can over-fit to specific keywords, failing to generalize to unseen data. After fine-tuning, it corrects its misinterpretation of task exemplars, expanding "positive connotation" into a broader "sentiment analysis" perspective. We present additional instruction trajectory results and examples in Appendix G.

On the other hand, in Fig. 5, the subplots show smoothed curves of *scaled distance* $\|\phi - \phi_0\|$ vs. performance, representing the distribution of instruction ϕ. The shaded gray region represents the standard deviation around the *reward-weighted mean distance*. We observe that the best instruction generally remains within a certain neighborhood around ϕ_0, and performance will not monotonically increase along with distance. A narrow shaded region (e.g., "taxonomy_animal") indicates that high-performance instructions are concentrated within a localized landscape. Hence, small perturbations can be applied to enable fine-grained optimization. In contrast, wide regions (e.g., "object_counting") suggest AIO can benefit from exploring a larger search space, to sufficiently distinguish effective instructions and learn from collected information.

5.3 Case Study: Transferability of AIO

We also study the transferability of AIO across induction and summarization tasks, where each AIO model is fine-tuned with LoRA on a source task, and evaluated across target tasks. We also include a baseline generating instructions with the pre-trained `Llama-3-8B-Instruct` w/o FT.

Source \ Target	**hyperbaton**	**logical**	**implicatures**	**epistemic**
Llama w/o FT	0.341	0.753	0.564	0.530
hyperbaton	0.527	0.814	0.826	0.628
logical_fallacy	0.217	0.836	0.846	0.616
implicatures	0.180	0.796	0.849	0.598
epi._reasoning	0.335	0.832	0.831	0.719

Table 2: Transferability of AIO across reasoning tasks.

Source \ Target	**podcast**	**qmsumm**	**samsum**	**scitldr**
Llama w/o FT	0.121	0.268	0.281	0.257
epi._reasoning	**0.151**	0.254	0.288	0.279
logical_fallacy	0.129	**0.329**	**0.317**	0.259
presuppositions	0.111	0.291	0.302	**0.304**
movie_rec.	0.130	0.289	0.312	0.263

Table 3: Transferability of AIO on SummEdits tasks.

As in Tables 2 and 3, AIO can generally enhance accuracy on transferred target tasks, demonstrating the generalization potential of our instruction optimization strategy. AIO fine-tuned on "logical_fallacy" transfers well to "implicatures" and "epistemic_reasoning", indicating shared reasoning capabilities and highlighting the potential of leveraging white-box LLM fine-tuning for transferable instruction optimization. However, cross-task generalization to "hyperbaton" is less effective, likely due to task dissimilarities. Meanwhile, as each fine-tuned model performs best on its own source task, this underscores the necessity of our proposed *task-aware* fine-tuning strategy, in order

to achieve optimal performance. In addition, we evaluate transfer performance (with F1 score) on summarization tasks from the SummEdits data set (Laban et al., 2023). In Table 3, AIO fine-tuned on reasoning tasks generalizes well to distinct summarization tasks, highlighting its strong generalizability potential across diverse types of target tasks.

6 Conclusion

In this paper, we introduce *Learning to Instruct* paradigm for black-box LLM instruction optimization, which formulates the instruction optimization process as an LLM fine-tuning objective, to harness the representation power of fine-tuned LLMs. We also propose AIO, a novel framework for optimizing task-specific black-box LLM instructions. By fine-tuning a white-box LLM as an instruction optimizer, AIO learns from task nuances and black-box LLM feedback to achieve adaptive instruction optimization. Given inaccessible black-box LLM gradients, we introduce a novel TS-aided zeroth-order gradient approximation method, for tractable and efficient white-box LLM fine-tuning. Extensive experiments demonstrate AIO's superiority in both performance and API token efficiency, with additional analysis highlighting its properties. Future extensions of AIO are discussed in Appendix F.

Limitations

We propose a new *Learning to Instruct* paradigm to achieve automatic instruction optimization for black-box LLMs. We also introduce a novel AIO framework, compatible with both PEFT and full fine-tuning, to establish *Learning to Instruct* as a new promising direction, paving the way for future developments in instruction optimization. While this work does not fully explore AIO's computational efficiency, fortunately, its design keeps the overall computational costs manageable. In this context, we view further improving AIO's efficiency as a valuable direction and propose to explore it in our future work. Analogous to many related works (e.g., (Pryzant et al., 2023; Chen et al., 2024b; Guo et al., 2024)), we perform our black-box LLM experiments using a single seed and omit standard deviation reporting, due to budget constraints associated with black-box API usage.

Acknowledgment

This work is supported by Agriculture and Food Research Initiative (AFRI) grant no. 2020-67021-32799/project accession no.1024178 from the USDA National Institute of Food and Agriculture. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shipra Agrawal and Navin Goyal. 2013. Thompson sampling for contextual bandits with linear payoffs. In *ICML*, pages 127–135. PMLR.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. Technical report, Anthropic.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Maurice S Bartlett. 1951. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, 22(1):107–111.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning*, 79:151–175.

BIG bench authors. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

HanQin Cai, Daniel McKenzie, Wotao Yin, and Zhenliang Zhang. 2022. Zeroth-order regularized optimization (zoro): Approximately sparse gradients and adaptive sampling. *SIAM Journal on Optimization*, 32(2):687–714.

Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Konstantinos Parasyris, Jiancheng Liu, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. 2024a. Deepzero: Scaling up zeroth-order optimization for deep model training. In *The Twelfth International Conference on Learning Representations*.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2024b. Instructzero: Efficient instruction optimization for black-box large language models. In *Forty-first International Conference on Machine Learning*.

Qi Chen, Changjian Shui, and Mario Marchand. 2021. Generalization bounds for meta-learning: An information-theoretic analysis. *Advances in Neural Information Processing Systems*, 34:25878–25890.

Alexandra Chronopoulou, Matthew E Peters, and Jesse Dodge. 2022. Efficient hierarchical domain adaptation for pretrained language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1336–1351.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *AISTATS*, pages 208–214.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. 2015. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Peter I Frazier. 2018. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Noa Garcia, Chentao Ye, Zihua Liu, Qingtao Hu, Mayu Otani, Chenhui Chu, Yuta Nakashima, and Teruko Mitamura. 2020. A dataset and baselines for visual question answering on art. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 92–108. Springer.

Saeed Ghadimi, Guanghui Lan, and Hongchao Zhang. 2016. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155(1):267–305.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Chi Han, Jialiang Xu, Manling Li, Yi Fung, Chenkai Sun, Nan Jiang, Tarek Abdelzaher, and Heng Ji. 2024a. Word embeddings are steers for language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16410–16430.

Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, and 1 others. 2024b. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

Or Honovich, Uri Shaham, Samuel R Bowman, and Omer Levy. 2022. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*.

Yutai Hou, Hongyuan Dong, Xinghao Wang, Bohan Li, and Wanxiang Che. 2022. Metaprompting: Learning to learn better prompts. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3251–3262.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021a. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiangqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. 2024. Localized zeroth-order prompt optimization. *arXiv preprint arXiv:2403.02993*.

Yifan Hu, Xin Chen, and Niao He. 2021b. On the bias-variance-cost tradeoff of stochastic optimization. *Advances in Neural Information Processing Systems*, 34:22119–22131.

William B. Johnson and Joram Lindenstrauss. 1984. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206.

Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radiček. 2023. Repair is nearly generation: Multilingual program repair with llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5131–5140.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*.

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. *arXiv preprint arXiv:2310.03714*.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*.

Philippe Laban, Wojciech Kryściński, Divyansh Agarwal, Alexander R Fabbri, Caiming Xiong, Shafiq Joty, and Chien-Sheng Wu. 2023. Llms as factual reasoners: Insights from existing benchmarks and beyond. *arXiv preprint arXiv:2305.14540*.

Kasper Green Larsen and Jelani Nelson. 2017. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. 2024. Use your instinct: Instruction optimization for llms using neural bandits coupled with transformers. In *Forty-first International Conference on Machine Learning*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*.

Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9004–9017.

Pierluigi Maponi. 2007. The solution of linear systems by using the sherman–morrison formula. *Linear algebra and its applications*, 420(2-3):276–294.

Jiří Matoušek. 2008. On variants of the johnson–lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*.

Yurii Nesterov and Vladimir Spokoiny. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2022. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Yunzhe Qi, Yikun Ban, Arindam Banerjee, and Jingrui He. 2024. Robust neural contextual bandit against adversarial corruptions. *Advances in Neural Information Processing Systems*, 37:19378–19446.

Yunzhe Qi, Yikun Ban, and Jingrui He. 2023a. Graph neural bandits. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 1920–1931, New York, NY, USA. Association for Computing Machinery.

Yunzhe Qi, Yikun Ban, Tianxin Wei, Jiaru Zou, Huaxiu Yao, and Jingrui He. 2023b. Meta-learning with neural bandit scheduler. *Advances in Neural Information Processing Systems*, 36:63994–64028.

Ruizhong Qiu and Hanghang Tong. 2024. Gradient compressed sensing: A query-efficient gradient estimator for high-dimensional zeroth-order optimization. In *Forty-first International Conference on Machine Learning*.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Tobias Schnabel and Jennifer Neville. 2024. Prompts as programs: A structure-aware approach to efficient compile-time prompt optimization. *arXiv preprint arXiv:2404.02319*.

Eric Schulz, Maarten Speekenbrink, and Andreas Krause. 2018. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of mathematical psychology*, 85:1–16.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

Yao Shu, Zhongxiang Dai, Weicong Sng, Arun Verma, Patrick Jaillet, and Bryan Kian Hsiang Low. 2023. Zeroth-order optimization with trajectory-informed derivative estimation. In *The Eleventh International Conference on Learning Representations*.

James C Spall. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341.

Haotian Sun, Yuchen Zhuang, Wei Wei, Chao Zhang, and Bo Dai. 2024. Bbox-adapter: Lightweight adapting for black-box large language models. *arXiv preprint arXiv:2402.08219*.

Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, Siyuan Lu, Yaliang Li, and Ji-Rong Wen. 2024. Unleashing the potential of large language models as prompt optimizers: An analogical analysis with gradient-based model optimizers. *arXiv preprint arXiv:2402.17564*.

Michal Valko, Nathan Korda, Rémi Munos, Ilias Flaounas, and Nello Cristianini. 2013. Finite-time analysis of kernelised contextual bandits. In *Uncertainty in Artificial Intelligence*.

Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. 2023. Recent advances in bayesian optimization. *ACM Computing Surveys*, 55(13s):1–36.

Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. 2022. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, 135:364–381.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. 2021. Neural thompson sampling. In *International Conference on Learning Representations*.

Dongruo Zhou, Lihong Li, and Quanquan Gu. 2020. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Potential Risks

This paper introduces the *Learning to Instruct* paradigm and the AIO framework for automating the optimization of effective and human-comprehensible instructions for black-box LLMs using a fine-tuned optimizer, thereby enhancing LLM performance and user transparency. While this work aims to improve the systematic and effective use of LLMs, we do not foresee significant negative potential risks and societal impacts from the methodology itself that warrant particular emphasis.

B Complementary Details for Implementation and Experiments

B.1 Descriptions for Tasks Involved in Our Experiments

For evaluation metrics: we have (1) Exact Match: the generated answer needs to exactly match the label; (2) Multiple Choice: task-solver LLM needs to choose one correct option out of several given candidate choices; (3) Binary Choice: task-solver LLM needs to choose one correct option out of two candidate choices; (4) Exact Set: whether the predicted set of items (e.g., animals) exactly matches the label set in both content and size, regardless of the item order.

B.2 Templates Applied for AIO Instruction Generation and Black-box LLM Inference (Zero-shot Induction and Few-shot Induction)

For zero-shot instruction induction and few-shot instruction induction, after obtaining generated instructions, we follow analogous ideas as in existing works (Zhou et al., 2022; Chen et al., 2024b; Lin et al., 2024), when formulating our instruction induction templates.

- **Instruction Generation Template:**

```
<examples>
Exemplary data: [EXEMPLARS]
</examples>.

Provide an instruction to infer the
    output of the input.
```

- **Evaluation Template:**

```
Instruction: [INSTRUCTION]
Input: [Query INPUT]
Output: [OUTPUT Placeholder]
```

- **Few-shot Instruction Induction Settings:**

```
<examples>
Exemplary data: [Exemplar data ([
    DEMO_DATA])]
</examples>.
Instruction: [INSTRUCTION]\n\n
Input: [Query INPUT] Output: [OUTPUT
     Placeholder]
```

- **Zero-shot Instruction Induction Settings:**

```
 Instruction: [INSTRUCTION]\n\n
Input: [Query INPUT] Output: [OUTPUT
     Placeholder]
```

Obviously, the main difference between these two templates is that few-shot template will also involve task exemplars during the inference stage, which can provide additional reference for the task-solver black-box LLM.

B.3 Baseline Descriptions.

Recall that we involve four baselines for comparison, including two kinds of methods that utilize LLMs for instruction optimization. The first kind methods leverages black-box LLM for instruction generation: (1) APE (Zhou et al., 2022) which generates instruction using a *black-box* LLM with designated templates and instruction search mechanism; (2) ProTeGi (Pryzant et al., 2023) applies a black-box LLM for instruction generation, and optimizes "Gradient Descent" on the generated instruction by integrating with a *black-box* LLM. Meanwhile, we also include (ii) Methods that utilize white-box LLM for instruction generation: (3) InstructZero (Chen et al., 2024b) generates instructions using a *white-box* LLM, while controlling the generation process by optimizing a prefix soft prompt, based on a kernel-based Bayesian Optimization approach; (4) INSTINCT (Lin et al., 2024) adopts an analogous approach as InstructZero, but alternatively applies a neural bandit model in replace of the kernel-based Bayesian Optimization for soft prompt selection; (5) ZOPO (Hu et al., 2024) formulates prompt selection as a discrete maximization problem and employs a zeroth-order optimization algorithm using a neural tangent kernel (NTK) surrogate built from a small neural network, along with fixed embeddings derived from a frozen white-box LLM. APE and ProTeGi will use `Claude-3-Sonnet` for instruction generation. Meanwhile, `Llama-3-8B-Instruct` is

Task	Metric	Descriptions
antonyms	Exact Match	Find the antonym for the given word.
sentiment	Binary Choice	Judge the sentiment preference of the given review.
larger_animal	Binary Choice	Identify which of the input animals is larger.
taxonomy_animal	Exact Set	Identify all the animal words out of input word sequence
object_counting	Exact Match	Enumerate objects of different types and output the total number.
navigate	Binary Choice	Given a series of navigation instructions, determine whether one would end up back at the starting point.
winowhy	Binary Choice	Evaluate the reasoning in answering Winograd Schema Challenge questions.
implicatures	Binary Choice	Predict whether Speaker 2's answer to Speaker 1 counts as a yes or as a no.
logical_fallacy_detection	Binary Choice	Detect informal and formal logical fallacies.
hyperbaton	Binary Choice	Order adjectives correctly in English sentences.
epistemic_reasoning	Binary Choice	Determine whether one sentence entails the next.
movie_recommendation	Multiple Choice	Recommend a movie that is similar to the given list of movies.
timedial	Multiple Choice	Pick the correct choice for a masked (temporal) span given the dialog context.
presuppositions_as_nli	Multiple Choice	Determine whether the first sentence entails or contradicts the second.
question_selection	Multiple Choice	Given a short answer along with its context, select the most appropriate question which has the given short answer as its answer.

Table 4: Task descriptions and corresponding metrics.

applied for InstructZero and INSTINCT, same as AIO. For the fixed embedding LLM of ZOPO, we follow the settings in the original ZOPO paper by applying a larger Vicuna-13B model with more parameters to generate initial instructions for ZOPO.

B.4 Implementation Details

For our zero-shot and few-shot instruction induction experiments, we consider each task is associated with 20 task exemplars denoted by $\mathcal{D}_{\text{Train}}$, as well as 100 validation samples $\mathcal{D}_{\text{Valid}}$, which will remain the same for AIO and all other baseline methods. From the remaining samples (i.e., not used for training or validation) of task $\mathcal{D}$, we sample test sets of size $|\mathcal{D}_{\text{test}}| = \frac{1}{5}|\mathcal{D}|$ (capped at 167), and compute the final result as the average performance over $\left\lceil \frac{1000}{|\mathcal{D}_{\text{test}}|} \right\rceil$ such test sets. They are shared across different methods for fair comparison, and this approach also helps maintain the black-box LLM cost budget. Analogous to many closely related works (e.g., (Pryzant et al., 2023; Chen et al., 2024b; Guo et al., 2024)), we conduct our black-box LLM experiments using a single seed and do not report standard deviations, due to our limited budget for black-box API usage costs. Nevertheless, our experiments span over 15 tasks and include extensive parameter, ablation and generalization results. Combined with AIO's consistently strong performance and token efficiency, our results provide meaningful and significant empirical evidence for our findings in this paper.

For AIO, when choosing our threshold parameter β, we initially set β as an infinitely large value to enable all collected black-box LLM feedback to be reused. Afterwards, we will experiment with $\beta = \mathcal{O}(\epsilon\sqrt{d})$ and choose the constant for $\mathcal{O}(\cdot)$ notation with grid search $\{1, 10, 100\}$. We perform the fine-tuning process for $T = 10$ rounds, as well as set the exploration variance $\nu = 0.1$ for all experiments For the perturbation magnitude parameter ϵ, we choose its value with grid search from $\{10^{-3}, 10^{-4}, 10^{-5}\}$. In each optimization round, we will draw $K = 10000$ arms from $\mathcal{N}(0, \boldsymbol{I})$ where we choose $B = 3$ arms for fine-tuning white-box LLM as well as update the TS model parameters. Regarding our TS model, after applying JL-Lemma and random Gaussian matrix projection (Matoušek, 2008; Larsen and Nelson, 2017) for dimension reduction, we will have the reduced dimension of TS model to be approximately $d' \approx 10^4$, which leads to ~ 0.4 seconds for selecting chosen arms $\widetilde{\mathcal{Z}}_t$ and ~ 3 seconds for TS model parameters update in each round t. For "AIO + LoRA", we set its "intrinsic rank" of low-rank approximation to 8. As we mentioned in the main body, we apply `Llama-3-8B-Instruct` (Dubey et al., 2024) as our tunable white-box LLM $\mathcal{F}_W(\cdot; \boldsymbol{\Theta}_W)$, and adopt `Claude-3-Sonnet` (Anthropic, 2024) as our black-box LLM $\mathcal{F}_B(\cdot)$. We perform all the experiments on a server with Intel Xeon CPU and NVIDIA V100 GPUs.

Tasks \ Methods	Black-box LLM		White-box LLM w/o FT			White-box LLM w/ FT (Ours)	
	APE	ProTeGi	InstructZero	INSTINCT	ZOPO	AIO + LoRA	AIO w/ Full FT
antonyms	0.901	0.889	0.894	0.905	0.882	0.895	**0.912**
sentiment	0.932	0.944	0.940	0.933	0.937	0.946	**0.950**
larger_animal	0.939	0.915	0.922	0.874	0.913	0.957	**0.961**
taxonomy_animal	0.708	0.972	0.835	0.869	0.911	**0.976**	0.967
object_counting	0.511	**0.583**	0.520	0.541	0.567	0.473	0.555
navigate	0.734	0.724	0.701	0.757	0.734	0.772	**0.776**
winowhy	0.563	0.674	0.673	0.682	**0.686**	0.621	0.628
implicatures	0.846	0.826	0.859	0.847	0.854	**0.867**	0.836
logical_fallacy_detection	0.850	**0.892**	0.877	0.881	0.852	0.885	0.880
hyperbaton	0.556	0.595	0.580	0.641	0.634	**0.660**	0.634
epistemic_reasoning	0.712	0.802	0.622	0.765	0.831	**0.884**	0.774
movie_recommendation	0.930	0.948	0.955	0.960	0.953	0.963	**0.979**
timedial	0.760	0.820	0.748	0.784	0.817	**0.832**	0.779
presuppositions_as_nli	0.557	0.564	0.591	0.598	0.575	0.594	**0.619**
question_selection	0.879	0.882	0.781	0.822	0.884	0.887	**0.916**
Average Rank	5.9	4.0	5.1	3.9	4.0	**2.5**	2.7

Table 5: Few-shot instruction induction results on 15 data sets. For each task (row), **bold** number refers to the best result, while underlined number refers to the second-best one. Similar to our zero-shot instruction induction experiments in Table 1, average rank results across tasks are reported.

C Complementary Experiments

C.1 Complementary Experiments Outline.

Due to strict page limit for the main body, we choose to include complementary experiments here in this section. As an outline, we have (1) few-shot experiments on our 15 tasks located in Subsec. C.2; (2) Chain-of-Thought (CoT) experiments in Subsec. C.3; (3) experiments with different combinations of white-box and black-box LLMs in Subsec. C.4; (4) an ablation study on AIO components in Subsec. C.5; (5) Parameter study for β and B in Subsec. C.6; (6) Empirical results with additional kinds of white-box LLMs in Subsec. C.7; (7) Effects of exemplar quantity in Subsec. C.8; (8) Transferability across black-box LLMs in Subsec. C.9; (9) Empirical comparison with an additional baseline EvoPrompt in Subsec. C.10; (10) Experiments of allowing AIO to leverage additionally available task prior information in Subsec. C.11; (11) Additional Experiments with a larger white-box LLM Qwen-2.5-14B in Subsec. C.12.

C.2 Complementary Experiments with Few-shot AIO

In this subsection, we include experiment results that examine AIO performance under few-shot settings, where training samples (or exemplars) will be provided to black-box LLM for reference. In this case, generated instructions will need to provide high-level reference and guidance, to assist the answer generation of task-solver black-box LLM in observation of task exemplars. The template applied is also shown in Subsec. B.2.

The experiment results are shown in Table 5. Here, we see that under few-shot settings, there exist performance improvements for most baselines compared with zero-shot settings, due to the help of additionally available task exemplars. By leveraging the sufficient representation power of fine-tuned white-box LLM, AIO can still generally maintain the best performance compared with baseline methods. Similar to our zero-shot induction results, AIO and its PEFT variant (AIO + LoRA) consistently maintain strong average rankings across tasks.

Data set	Method	Instruction	Accuracy Result
GSM8K	CoT	Let's think carefully step by step	0.724
	AIO	Use your math skills and logic to break down the problem into manageable parts.	0.862
AQUA	CoT	Let's think carefully step by step	0.317
	AIO	Think critically and break down the problem into smaller parts to solve it.	0.410
SVAMP	CoT	Let's think carefully step by step	0.766
	AIO	Let us think critically and break it down!	0.898

Table 6: Chain-of-Thought (CoT) results.

C.3 Chain-of-Thought (CoT) Results

We also include additional Chain-of-Thought (CoT) experiments on three data sets including GSM8K (Cobbe et al., 2021), AQUA (Garcia et al., 2020), and SVAMP (Patel et al., 2021), where results are shown in Table 6. For comparison, we include a baseline instruction "Let's think carefully

Task	White-box LLM	Task-solver LLM	Best Instruction	Accuracy Result
navigate	`Llama-3-8B-Instruct`	`Llama-3-70B-Instruct`	If the instructions are able to return to the starting position after following all the instructions, then the output is True. Otherwise, the output is False.	0.567
larger_animal	`Llama-3-8B-Instruct`	`Llama-3-70B-Instruct`	First identify the animals in the input. Then, sort the animals in descending order based on their average adult body mass. If there are multiple animals with the same average adult body mass, sort them in alphabetical order. Finally, return the first animal in the sorted list as the output.	0.872

Table 7: Applying a white-box LLM (`Llama-3-70B-Instruct`) as problem-solving LLM.

step by step", which is commonly applied for solving CoT tasks, following the settings from Chen et al. (2024b); Lin et al. (2024).

With results in Table 6, we see that AIO can significantly improve black-box induction performance under CoT reasoning settings compared with the task-agnostic instruction "Let's think carefully step by step", where AIO's performance improvements can be credited to the utilization of task-aware instructions. For instance, since GSM8K is a math reasoning task, AIO choose to introduce additional background information by asking the task-solver black-box LLM to "use your math skills" and decompose the target math problem into "manageable parts". This can help the black-box LLM determine which part of or what kinds of learned knowledge should be applied for problem solving, with higher levels of clarity than task-agnostic instructions.

C.4 Different Combinations of White-box and Black-box LLMs

Recall that for our previous experiments, we have applied `Llama-3-8B-Instruct` (Dubey et al., 2024) as our tunable white-box LLM $\mathcal{F}_W(\cdot;\boldsymbol{\Theta}_W)$, and adopt `Claude-3-Sonnet` (Anthropic, 2024) as our black-box LLM $\mathcal{F}_B(\cdot)$. Here, for two tasks "navigate" and "larger animal", we include experiments with one more recent white-box LLM `Llama-3.1-8B-Instruct`, as well as a relatively light-weight black-box LLM `Claude-3-Haiku` for comparisons. Meanwhile, we also include experiments by substituting our black-box LLM with a powerful white-box LLM `Llama-3-70B-Instruct` for comparisons.

Results are shown in Tables 7 and 8. Here, we see that using a more recent and capable white-box LLM can generally lead to slightly better performance. However, during our experiments, we also notice that fine-tuning `Llama-3.1-8B-Instruct` can be slightly more time consuming then tuning `Llama-3-8B-Instruct`. On the other hand, during our experiments, using a more light-weight black-box LLM can significantly accelerate the inference speed in terms of answer generation with relatively less API token costs. It can still achieve relatively good performance on "larger animal" task with a slightly inferior performance on "navigate" task. We also notice that the large white-box LLM `Llama-3-70B-Instruct` tends to perform slightly inferior compared with Claude family black-box LLMs, when applying AIO for optimization.

Task	White-box LLM	Black-box LLM	Accuracy Result
navigate	`Llama-3-8B-Instruct`	`Claude-3-Sonnet`	0.644
	`Llama-3.1-8B-Instruct`	`Claude-3-Sonnet`	0.689
	`Llama-3-8B-Instruct`	`Claude-3-Haiku`	0.612
	`Llama-3.1-8B-Instruct`	`Claude-3-Haiku`	0.643

Task	White-box LLM	Black-box LLM	Accuracy Result
larger_animal	`Llama-3-8B-Instruct`	`Claude-3-Sonnet`	0.912
	`Llama-3.1-8B-Instruct`	`Claude-3-Sonnet`	0.927
	`Llama-3-8B-Instruct`	`Claude-3-Haiku`	0.935
	`Llama-3.1-8B-Instruct`	`Claude-3-Haiku`	0.887

Table 8: Different combinations of white-box LLMs vs black-box LLMs.

C.5 Ablation Study on AIO Components

Recall that AIO has two main components: white-box LLM back-propagation and TS-aided ZO black-box LLM gradient approximation, to derive the gradients for white-box LLM and black-box LLM respectively, based on the decomposed gradient flow in Eq. 3. Here, we include an ablation study for these two components for gradient derivation: (1) the first baseline is "AIO w/ MeZO" which directly use MeZO (Malladi et al., 2023) for approximating white-box LLM parameter gradients instead of our gradient decomposition formulation (Remark 4.1); (2) the second baseline is "AIO w/o TS Scheduling" where we do not apply Thompson Sampling for selecting perturbation directions and use completely random perturbation vectors $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ for white-box gradient approximation, formulated in Eq. 4.

Experiment results are shown in Table 9. We can see that our proposed AIO with TS-aided ZO

Methods	antonyms	sentiment	l_animal	t_animal	navigate	implicatures	logical_fallacy_detection	e_reasoning
AIO	**0.901**	**0.949**	**0.912**	**0.983**	0.644	**0.811**	**0.868**	**0.766**
AIO w/ MeZO	0.852	0.930	0.847	0.279	**0.654**	0.675	0.812	0.748
AIO w/o TS Scheduling	0.870	0.929	0.760	0.957	0.604	0.787	0.832	0.742

Table 9: Ablation study on AIO with two variants: (1) "AIO w/ MeZO" directly applies MeZO for fine-tuning white-box LLM $\mathcal{F}_W$, instead of using our proposed gradient flow decomposition and TS-aided ZO gradient approximation; (2) "AIO w/o TS Scheduling" refers to a variant where perturbation directions $\boldsymbol{z}$ are sampling randomly from $\mathcal{N}(0, \boldsymbol{I})$, instead of being chosen by our TS model.

Number of chosen arms (directions) B				
Task \ B value	1	2	3	4
Larger Animal	0.915	0.931	**0.950**	0.941
Navigate	0.610	0.632	0.627	**0.664**
logical_fallacy_detection	0.815	0.824	0.836	**0.849**
hyperbaton	0.514	0.489	0.527	**0.565**
epistemic_reasoning	0.685	0.723	0.719	**0.755**
implicatures	0.804	0.821	**0.849**	0.846
presuppositions_as_nli	0.437	0.479	0.493	**0.512**
Threshold parameter β for optimization landscape				
Task \ β value	1	10	100	∞
Larger Animal	0.922	0.915	0.932	**0.950**
Navigate	0.611	0.634	**0.676**	0.627
logical_fallacy_detection	0.796	0.832	0.823	**0.836**
hyperbaton	0.538	**0.550**	0.527	0.521
epistemic_reasoning	0.677	0.701	0.680	**0.719**
implicatures	0.794	0.833	**0.849**	0.827
presuppositions_as_nli	0.450	0.442	0.467	**0.493**

Table 10: Experiment results with different B values and β values.

White-box LLM \ Task	larger animal	navigate	sentiment	movie_recom.
Llama-3-8B-Instruct	**0.950**	0.627	**0.947**	**0.883**
Mistral-7B-Instruct-v0.2	0.890	0.634	0.907	0.874
Qwen2.5-7B-Instruct	0.919	**0.682**	0.922	0.835

Table 11: Experiment results with different kinds of white-box LLMs. Claude-3-Sonnet is applied as black-box task-solving LLM.

Different numbers of exemplars $\lvert\mathcal{D}_{\text{train}}\rvert$				
Task \ $\lvert\mathcal{D}_{\text{train}}\rvert$	5	10	20	30
Larger Animal	0.868	0.921	**0.950**	0.947
Navigate	0.622	0.634	0.627	**0.681**

Table 12: Experiment results with different numbers of exemplars $\lvert\mathcal{D}_{\text{train}}\rvert$.

Gradient Approximation can still maintain superior performance compared with the other two baselines with substituted modules. This helps to reinforce our claim that our proposed gradient flow decomposition approach (Eq. 3) as well as the ZO black-box LLM gradient approximation method guided by Thompson Sampling are necessary for AIO to achieve optimal performance. In particular, the supposed linear optimization landscape enables us to utilize a linear Thompson Sampling model for ZO perturbation direction selection, which is effective and computationally efficient for perturbation direction selection.

C.6 Parameter Study for β and B

We include additional study for parameters β and B of AIO. Here, additional experiment results for zero-shot instruction induction with the LoRA module are presented in the table below.

Results are shown in Tables 10. For parameter B, we observe that setting $B = 3$ can achieve promising performance, which can help balance computational (and token) costs with performance. For parameter β, it is recommended to start the tuning process with a large β, as suggested in our Appendix B.4, to effectively leverage past received records of the optimization landscape. On the other hand, a sufficiently small threshold β will cause the method to degenerate into "AIO w/o TS Scheduling", as in our ablation study (Subsec. C.5). In this case, the TS model will be excluded from selecting ZO approximation directions, leading to relatively inferior performance.

C.7 Combinations of Different White-box LLMs

We include additional experiments with other types of white-box LLMs can benefit the audience. We conduct experiments with the LoRA module on two additional types of white-box LLMs: Mistral-7B-Instruct-v0.2 and Qwen2.5-7B-Instruct. With black-box LLM being Claude-3-Sonnet, we have zero-shot instruction induction results shown in Table 11. We see that our proposed AIO framework achieves promising performance with other types of white-box LLMs other than the Llama family. Meanwhile, we also would like to mention that Llama 3 (Llama-3-8B-Instruct) generally retains a slight advantage over the other two white-box LLMs. One possible reason is that Llama 3 consists of 8 billion parameters, slightly more than the other two 7-billion-parameter models. This can provide an advantage in terms of the representation power to some extent.

Black-box LLM Transferability						
Black-box LLM \ Task	epi._reasoning	logical fallacy	hyperbaton	movie_recom.	navigate	implicatures
Claude 3 Sonnet	0.719	0.836	0.527	0.883	0.627	0.849
Claude 3.5 Sonnet	**0.844**	**0.886**	**0.562**	**0.924**	**0.639**	**0.918**
GPT-3.5-Turbo	0.737	0.811	0.556	0.825	0.580	0.855
GPT-4o	0.768	0.854	0.540	0.910	0.594	0.886

Table 13: Experiment results with different kinds of black-box LLMs.

Comparison with EvoPrompt				
Method \ Task	hyperbaton	navigate	movie_recom.	obj._counting
AIO	**0.538**	**0.644**	**0.902**	**0.543**
EvoPrompt	0.526	0.627	0.895	0.466

Table 14: Experiment results in comparison with EvoPrompt (Guo et al., 2024).

With additional textual task description information				
Method \ Task	epi._reasoning	logical_fallacy	hyperbaton	movie_recom.
AIO	0.719	0.836	0.527	0.883
AIO w/ Task Info	**0.811**	**0.872**	**0.639**	**0.895**

Table 15: Experiment results with additional textual task description information.

C.8 Different Numbers of Exemplars

As mentioned in our experimental settings (Appendix Subsec. B.2), we use 20 training exemplars, a reasonably small amount of training data, as the reference for the white-box LLM to generate and optimize instructions. We also include additional zero-shot instruction induction experiments with varying exemplar quantities and the LoRA module. This is to investigate how the performance changes when altering the number of exemplars $|\mathcal{D}_{\text{Train}}|$ for AIO, in terms of instruction generation and optimization.

From the results in Table 12, we observe that the "larger animal" task can be more sensitive to the number of exemplars $|\mathcal{D}_{\text{Train}}|$. However, providing as few as $|\mathcal{D}_{\text{Train}}| = 10$ query-answer pairs as exemplars, which is a modest quantity, will allow AIO to achieve relatively promising performance. Meanwhile, we see that for the "navigate" task, a small number of $|\mathcal{D}_{\text{Train}}| = 5$ exemplars can lead to promising results, which shows that it is more stable under the sparse data settings. Therefore, when fine-tuning AIO from scratch, the overall performance of AIO can possibly be influenced by data scarcity, with its impact varying based on the specific application scenarios of practitioners.

C.9 Transferability of Fine-tuned Instruction Optimizers across Different Black-box Task-solving LLMs

In this subsection, we investigate if the optimized instructions can generalize to different black-box LLMs. We transfer our optimized white-box LLM with the LoRA module, which is fine-tuned under the settings of Llama 3 + Claude 3 Sonnet, to other black-box task-solving LLMs. They include Claude 3.5 Sonnet and two OpenAI black-box LLMs (GPT-3.5-Turbo and GPT-4o).

The accuracy results in terms of zero-shot instruction induction are shown in Table 13. Here, we can observe that the optimized "instruction engineer" white-box LLM can maintain strong performance when being applied to other black-box LLMs. Meanwhile, we also notice that the latest language models (Claude 3.5 and GPT-4o) can generally outperform the older ones (Claude 3 and GPT-3.5-Turbo), due to their stronger reasoning capabilities.

C.10 Additional Baseline: EvoPrompt

In this subsection, we included an additional baseline EvoPrompt (Guo et al., 2024), which alternatively utilizes evolutionary algorithms to refine LLM-generated instructions in an in-context learning manner. Different from our original problem settings, where instructions are generated from scratch, EvoPrompt requires initial instructions. In this case, we follow the settings in the original paper (Guo et al., 2024) and the official source code of EvoPrompt, by using instructions generated by APE (Zhou et al., 2022) as the initial instructions for EvoPrompt, while AIO does not rely on such information. We compare AIO with EvoPrompt, in terms of zero-shot instruction induction on four BigBench (bench authors, 2023) tasks, as BigBench is also applied in Guo et al. (2024).

Based on the results in Table 14, we see that our proposed AIO can manage to achieve generally better performance compared with EvoPrompt. Meanwhile, comparing with the two black-box LLM baselines: APE (Zhou et al., 2022) and ProTeGi (Pryzant et al., 2023), EvoPrompt can achieve relatively better performance than them, by utilizing task prior knowledge provided by initial instructions as the starting point for optimization.

Experiments with Qwen-2.5-14B					
Task \ Method	ProTeGi	INSTINCT	AIO	AIO+LoRA	AIO+LoRA (Qwen-2.5-14B)
object_counting	0.550	0.537	0.543	0.479	**0.561**
larger_animal	0.932	0.857	0.912	**0.950**	0.922
winowhy	0.703	0.725	0.622	0.635	**0.748**
hyperbaton	0.499	0.502	0.538	0.527	**0.645**
epistemic_reasoning	0.459	0.580	0.766	0.719	**0.791**

Table 16: Experiments with a larger Qwen-2.5-14B, fine-tuned with LoRA, including five zero-shot instruction induction and reasoning tasks. We see that integrating AIO with a more powerful white-box LLM can generally improve downstream black-box LLM performance, by leveraging an enhanced learning capability to capture the complex correlation between instruction strategies and the black-box LLM performance.

C.11 Additionally Available Task Narrative / Description

In addition, if task descriptions are provided to AIO as prior knowledge, we can incorporate this information into the instruction generation template (left-hand side of Fig. 1). In this case, the input to the white-box LLM will include both the task exemplars and the textual task description. We conduct additional experiments in terms of zero-shot instruction induction on four BigBench tasks with the LoRA module, incorporating the one-sentence task descriptions provided by the BigBench data set into our generation template.

Results are shown in Table 15. We see that involving additional textual task information into the instruction generation and optimization process can generally improve performance, particularly for logical reasoning tasks (e.g., epistemic reasoning, logical fallacy, hyperbaton), as task descriptions can help provide extra background information to prevent potential misinterpretations of exemplars.

C.12 Additional Experiments with Qwen-2.5-14B

To further demonstrate AIO's compatibility with white-box LLMs of different scales, we include additional experiments aopting a larger Qwen-2.5-14B model fine-tuned with LoRA for our proposed AIO. With the results shown in Table 16, our results indicate that integrating AIO with a more powerful white-box LLM generally can improve downstream task performance, by leveraging enhanced learning capacity to capture the complex correlation between instruction strategies and black-box LLM performance. This further highlights the scalability and adaptability of our AIO framework.

D Supplementary Technical Details

D.1 Details for Auto-regressive Generation and Perturbation

Analogous to existing works (e.g., Li and Liang (2021)), we formulate the LLM generation process of a token sequence $\boldsymbol{x}$. We begin with an initial input context $\boldsymbol{x}_{<1}$ that can be empty or contain special tokens like start-of-sequence token. Then, the language model will sequentially generate each token in the output, by sampling each generated token x_i from the conditional distribution $p_{\boldsymbol{\Theta}}(x_i \mid \boldsymbol{x}_{<i})$. In particular, the output logits for the i-th token will go through a softmax function, after applying a language model header (with weight $\boldsymbol{\Theta}_{\text{header}}$ from language model parameters $\boldsymbol{\Theta}$) to map i-th token hidden representation to the vocabulary distribution, as

$$p_{\boldsymbol{\Theta}}(x_i \mid \boldsymbol{x}_{<i}) = \text{softmax}(\boldsymbol{\Theta}_{\text{header}} \cdot \boldsymbol{h}_i),$$

where $\boldsymbol{h}_i$ represents the transformer-embedded hidden representation of i-th generated token. Each token $\boldsymbol{x}_i$ will be sampled from the vocabulary, based on $p_{\boldsymbol{\Theta}}(x_i \mid \boldsymbol{x}_{<i})$. The generation process will complete if special tokens (e.g., an EOS token) is encountered, or the maximum length is met.

Here, we slightly abuse notation by using the operations "$+$" and "$-$" to apply perturbation $\boldsymbol{z}$ to the token-level output of each token from ϕ. Following our gradient approximation formulation in Eq. 4, with random perturbation vector (gradient approximation vector) $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ and perturbation magnitude $\epsilon > 0$, we recall that the perturbation is imposed on LLM-header output logits (distributed over the vocabulary) of *each* i-th generated token. This leads to the positively perturbed generation process $p_{\boldsymbol{\Theta}}(x_i \mid \boldsymbol{x}_{<i}) \leftarrow \text{softmax}(\boldsymbol{\Theta}_{\text{header}} \cdot \boldsymbol{h}_i + \epsilon \boldsymbol{z})$, as well as the negatively perturbed generation process $p_{\boldsymbol{\Theta}}(x_i \mid \boldsymbol{x}_{<i}) \leftarrow \text{softmax}(\boldsymbol{\Theta}_{\text{header}} \cdot \boldsymbol{h}_i - \epsilon \boldsymbol{z})$. Consequently, the per-

turbation $\boldsymbol{z} \in \mathbb{R}^d$ will be of the same as the vocabulary dimension d.

D.2 Details for Efficient Covariance Matrix Inversion Update with Sherman-Morrison Formula

Recall that in Remark 4.2, for updating covariance matrix inversion $\boldsymbol{\Sigma}_t^{-1}$ efficiently in each round t, we apply Sherman-Morrison Formula (Bartlett, 1951; Maponi, 2007) by updating matrix inversion incrementally. Here, suppose that current white-box LLM output ϕ is close enough to the checkpoint ϕ_{Check}, which means that we can update currently possessed covariance matrix inversion $\boldsymbol{\Sigma}_{t-1}^{-1}$ to obtain $\boldsymbol{\Sigma}_t^{-1}$. Then, we recall that the arm context covariance matrix in each round t is constructed by $\boldsymbol{\Sigma}_t = \boldsymbol{I} + \sum_{(\boldsymbol{z},r)\in\Omega_t} \boldsymbol{z}\cdot\boldsymbol{z}^\intercal = \boldsymbol{\Sigma}_{t-1} + \sum_{\boldsymbol{z}\in\widetilde{\mathcal{Z}}_t} \boldsymbol{z}\cdot\boldsymbol{z}^\intercal$, where $\widetilde{\mathcal{Z}}_t$ refers to the collection of chosen arms in round t. Since we have each $\boldsymbol{z}\boldsymbol{z}^\intercal$ being a rank-one matrix for every $\boldsymbol{z} \in \widetilde{\mathcal{Z}}_t$, we can follow Sherman-Morrison Formula to perform one-step update

$$\left(\boldsymbol{\Sigma}_{t-1} + \boldsymbol{z}\boldsymbol{z}^\intercal\right)^{-1} = \boldsymbol{\Sigma}_{t-1}^{-1} - \left[\frac{\boldsymbol{\Sigma}_{t-1}^{-1}\boldsymbol{z}\boldsymbol{z}^\intercal\boldsymbol{\Sigma}_{t-1}^{-1}}{1 + \boldsymbol{z}^\intercal\boldsymbol{\Sigma}_{t-1}^{-1}\boldsymbol{z}}\right].$$

In this case, by iteratively repeating this process for $|\widetilde{\mathcal{Z}}_t| = B$ times (since we have $B \ll K$ chosen arms in each round t, and B is a considerably small integer), we will have the updated covariance matrix inverse $\boldsymbol{\Sigma}_t^{-1}$. Recall that each covariance matrix will have a shape of $d \times d$, where d is the dimensionality of perturbation vector $\boldsymbol{z}$. We then have the overall computational costs as approximately $\mathcal{O}(Bd^2)$ instead of the naive $\mathcal{O}(d^3)$, where we also intuitively have $B \ll d$. Moreover, with dimension reduction approach motivated by JL-Lemma (Remark 4.2), we can have the projected context dimension $d' \ll d$, which leads to computational complexity of $\mathcal{O}\big(B \cdot (d')^2\big)$, instead of the naive $\mathcal{O}\big((d')^3\big)$ with the direct matrix inversion.

E Complementary discussions

E.1 Motivation of Thompson Sampling and Bias Associated with This Design

E.1.1 Additional discussion on Motivations

As mentioned in our main body (paragraph below Remark 4.1), conventional ZO gradient approximation methods have a potential drawback: the perturbation vectors $\boldsymbol{z}$ are randomly sampled. Consequently, their gradient approximation directions in the optimization landscape are random, which can result in inefficient gradient estimation process. To overcome this challenge, numerous ZO methods have been proposed to incorporate guided or chosen directions for gradient optimization, enabling more efficient estimation of the target gradient (e.g., Cai et al. (2022); Qiu and Tong (2024)).

In this work, we propose reusing collected feedback by framing the ZO-based fine-tuning process as a sequential decision-making problem, and applying Contextual Bandit techniques to effectively identify beneficial perturbation directions worth exploring. With the assumed linear optimization landscape as in existing works (e.g., Spall (1992); Malladi et al. (2023)), we apply linear Thompson Sampling to leverage previously collected information, which includes arms (previous gradient approximation directions) and corresponding rewards (benefits of going along these directions), to achieve a more efficient gradient estimation (Subsec. 4.1.2). Within the linear optimization landscape, our TS model aids in selecting gradient directions for white-box LLM fine-tuning, by properly reducing the possibility of choosing low-value directions (e.g., directions that are orthogonal to the true gradient, which can provide limited information for gradient approximation). This approach helps optimize the validation score through instruction generation and ZO approximation direction selection, aligned with our formulation of the arm reward (Eq. 7).

Ablation study on AIO components. We also would like to mention that the effectiveness of our TS modeling is supported by our ablation study (Appendix Subsec. C.5), where we compare our AIO framework with two alternatives: (1) "AIO w/ MeZO", where MeZO (Malladi et al., 2023) is used directly for approximating white-box LLM parameter gradients instead of our gradient decomposition formulation (Remark 4.1). (2) "AIO w/o TS Scheduling", where Thompson Sampling is not applied for selecting perturbation directions, and completely random perturbation vectors $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ are used for zeroth-order gradient approximation as in Eq. 4. We observe that our proposed AIO with TS-aided ZO gradient approximation generally achieves better performance, compared with the two baselines with substituted modules.

E.1.2 Bias associated with TS

By balancing exploitation and exploration, TS will not introduce considerable bias. Under the settings of gradient approximation, our proposed TS-

based zeroth-order gradient approximation method itself does not inherently introduce considerable bias in terms of gradient estimation, as Thompson Sampling techniques can naturally tackle the exploitation-exploration dilemma (Agrawal and Goyal, 2013; Zhang et al., 2021) by exploring various gradient directions instead of greedily exploiting only certain ones. Here, TS model will choose from sampled candidate approximation directions, instead of actually altering the direction vector value. Meanwhile, in the short term, it is possible that TS may introduce temporary bias if it focuses on some perturbation directions that can lead to high rewards (i.e., directions that can help optimize validation score, Eq. 7). Due to the *bias-variance trade-off* (Hu et al., 2021b), the temporary bias can also be beneficial under our gradient approximation settings, and we will elaborate on this point in the next paragraph. On the other hand, with only a few gradient directions applied for LLM fine-tuning (Malladi et al., 2023) in each optimization round, directly applying conventional unbiased estimators with randomly sampled directions can possibly result in high variance in terms of gradient estimation (Cai et al., 2022), which also reflects the *bias-variance trade-off* in terms of zeroth-order gradient approximation.

We introduce the TS to balance the "bias" and "variance" trade-off for ZO gradient approximation, under query-limited scenarios. Here, we would like to mention that state-of-the-art ZO gradient approximation methods with a guided approximation process (e.g., Qiu and Tong (2024)) can also introduce temporary bias in terms of gradient direction selection. However, they have been shown highly effective and efficient, particularly under sample-efficient settings, due to their ability of involving informative gradient approximation directions to balance "bias" and "variance" for gradient approximation. As a result, due to the bias-variance trade-off, it can lead to the case that *no algorithm can serve as a universal solution to various application scenarios of zeroth-order approximation.* Therefore, practitioners need to tailor solutions to their specific application scenarios. In our case, by balancing high-reward directions (exploitation) and sampling TS parameters from the posterior (exploration), our TS-aided approximation can help stabilize the gradient estimates, making it advantageous in query-limited scenarios like ours. In particular, by leveraging a highly efficient linear TS model, the arm selection process (~ 0.4 seconds per round) will be fast, ensuring the efficiency in terms of gradient direction selection.

Ablation study on TS-aided zeroth-order gradient approximation. The effectiveness of our modeling is also validated by our ablation study (Appendix Subsec. C.5), where we include a variant, "AIO w/o TS Scheduling", for comparison. Instead of using TS, "AIO w/o TS Scheduling" applies randomly chosen perturbation vectors $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ for zeroth-order gradient approximation, as described in Eq. 4. In contrast, our AIO with TS-aided ZO gradient approximation achieves better performance by strategically selecting informative gradient approximation directions, based on collected information and knowledge from the optimization landscape.

E.2 Reasoning of Our Choice of Data Sets and Baselines

Under our problem settings (Section 2), the black-box LLM is considered as part of the learning objective rather than the learning model, and we have no control over its parameters while can only interact with it through the API. Therefore, based on our problem settings, which align with those of closely related works (Chen et al., 2024b; Lin et al., 2024), the learning model will solely be the white-box LLM, while the black-box LLM will serve as part of the learning objective.

E.2.1 Choice of data sets.

Different from conventional instruction optimization settings, in this work, we address the challenge of automatically optimizing instructions for a task-solving black-box LLM in terms of the given target task. The process requires only a few exemplars as training data and eliminates the need for human expert intervention. This is an emerging topic of automatic instruction generation that has been explored in several related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024)). Unlike existing approaches, we propose utilizing a white-box LLM for instruction generation and optimization, coupled with LLM fine-tuning to effectively learn optimized instructions for highly complex modern black-box LLMs.

As discussed in the Introduction and Related Works sections, the most relevant state-of-the-art works to this paper are (Chen et al., 2024b; Lin et al., 2024), where a white-box LLM is also used as an instruction optimizer to tailor instructions specifically for the downstream task-

solving black-box LLM. Therefore, for the data sets, we follow the most closely related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024)) by utilizing instruction induction tasks from (Honovich et al., 2022), reasoning tasks from BigBench (bench authors, 2023), and Chain-of-Thought (CoT) data sets (Cobbe et al., 2021; Garcia et al., 2020; Patel et al., 2021) to benchmark our proposed AIO against baselines. These data sets are both common and widely adopted in this line of research, particularly in works closely aligned with ours in terms of problem settings (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024)).

The instruction induction tasks from (Honovich et al., 2022) and reasoning tasks from BigBench (bench authors, 2023) are used to evaluate the zero-shot instruction induction (Subsec. 5.1) and few-shot instruction induction (Subsec. C.2) quality of the optimized instructions. In particular, since no auxiliary task-relevant information is provided to the task-solving black-box LLM, the zero-shot instruction induction results on these data sets (Honovich et al., 2022; bench authors, 2023) are widely adopted to assess instruction quality by closely related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024); Fernando et al. (2023); Hu et al. (2024)). Additionally, the applied CoT data sets are also commonly used in the aforementioned related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024)), as they effectively test the instruction optimizers' ability to solve complex reasoning tasks, such as math problems. Therefore, our data set selections are standard and widely adopted in closely related works, in order to demonstrate the effectiveness of AIO in terms of instructing the task-solving black-box LLM.

E.2.2 Choice of baselines.

In this paper, we primarily focus on the challenge of automatically optimizing instructions for a task-solving black-box LLM given the target task. This distinguishes our problem settings from those of conventional instruction optimization works. In this case, the most relevant works to ours are (Chen et al., 2024b; Lin et al., 2024), which propose to deal with an analogous problem. From an in-context learning perspective instead, they apply a white-box LLM as an instruction generator to tailor instructions specifically for the downstream task-solving black-box LLM. Therefore, we have included these two works (InstructZero (Chen et al., 2024b), INSTINCT (Lin et al., 2024)) as our baselines to emphasize the advantages of LLM fine-tuning over their in-context learning approaches under our instruction optimization settings, given the superior representation and learning capabilities provided by LLM fine-tuning.

Meanwhile, there is also a line of research (e.g., Zhou et al. (2022); Pryzant et al. (2023)) that employs a black-box LLM instead for instruction generation, resulting in a pipeline involving two black-box LLMs. In this setup, the "instruction engineer" black-box LLM perceives the feedback of the task-solver black-box LLM, and optimizes its instructing strategy from an in-context learning perspective. Since these methods are also commonly adopted by our closely related works (Chen et al., 2024b; Lin et al., 2024), we have also included APE (Zhou et al., 2022) and ProTeGi (Pryzant et al., 2023) as our baselines. Therefore, our choice of baselines is standard in this line of research, including state-of-the-art baselines closely related to this paper, as well as common baselines adopted by other existing works in this research direction.

F Generalizing AIO to other application scenarios

F.1 Involving Domain Experts in the Optimization of Instructions

As shown in our pipeline illustration (the Generation Template in Fig. 1), we only need a few exemplars from the target task as the reference to the white-box LLM for instruction generation and optimization. In this case, we do not need specific human-crafted task context for our AIO, which avoids the need for human expert intervention. This emerging topic in automatic instruction generation has been explored in several related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024)). Unlike prior approaches, we propose utilizing a white-box LLM for instruction generation and optimization, paired with LLM fine-tuning to efficiently learn optimized instructions for modern, highly complex black-box LLMs. On the other hand, if domain experts (e.g., human engineers) or domain knowledge (e.g., textual task descriptions or narratives, Subsec. C.11) are available, there are several ways of integrating them to our instruction optimization process.

F.1.1 Involving Human experts

When human experts are available, following the idea of "Human-in-the-loop" (Wu et al., 2022), we can involve these experts to help AIO in terms of instruction optimization and evaluation. For instance, in terms of instruction improvement for medical diagnostics, AIO can generate instructions such as "Please analyze patient symptoms to identify possible illnesses". Human evaluators could define metrics emphasizing clarity (e.g., suggesting tools like "blood test" or "MRI scan") or appropriateness (e.g., ensuring instructions do not encourage unsafe practices). This can enhance AIO's applicability in sensitive fields where clarity and accuracy are crucial. Compared to composing instructions from scratch that requires more efforts, this formulation can help reduce the workload of human experts.

Meanwhile, human experts can be involved into the instruction evaluation process. In this case, AIO can be requested to generate multiple candidate instructions after fine-tuning, and then human experts can proceed to examine which of the candidates is most appropriate from multiple dimensions, such as interpretability. This will also help to enhance AIO's applicability for real-world scenarios.

F.2 Generalizing Optimized Instruction to Related Tasks or Domains

Recall that this work focuses on the problem of automatically optimizing instructions for a task-solving black-box LLM given the target task. The instruction generation and optimization require only a few exemplars as training data and do not involve human expert intervention. This problem is an emerging topic in the field of automatic instruction generation, with several related works (e.g., Zhou et al. (2022); Chen et al. (2024b); Lin et al. (2024); Hu et al. (2024)). Different from existing works, in this paper, we propose leveraging a white-box LLM for instruction generation and optimization, combined with LLM fine-tuning to effectively learn optimized instructions for modern black-box LLMs of extreme complexity. However, while this paper focuses on optimizing instructions for a given target task, our AIO framework can be potentially extended to generalize across multiple related tasks or domains.

Transfer learning and domain adaptation. First, we can leverage ideas from transfer learning and domain adaptation to address this issue. Here, we can consider two related domains (tasks): the source domain and the target domain. The source domain will refer to the domain that we train the original instruction optimizer (white-box LLM), typically having larger amount of training data (exemplars) than the target domain. Afterwards, to adapt the trained white-box LLM to the target domain, we can apply few-shot domain adaptation techniques (e.g., Chronopoulou et al. (2022)) to efficiently fine-tune the white-box LLM parameters for the target domain, even with a relatively smaller number of samples. In the meantime, in the context of transfer learning, we can possibly have additional theoretical insights. This is because the generalization loss on the target domain can be theoretically upper bounded by the empirical loss on the source domain, the learning power of the neural model, and the discrepancy between the source and target domains (Ben-David et al., 2010).

Meta-learning. On the other hand, we can leverage ideas from meta-learning (Finn et al., 2017) to develop an instruction "meta-optimizer", which is capable of abstracting high-level information across multiple domains and generating high-quality meta-instructions (analogous to meta-prompts (Hou et al., 2022)). Using this approach, the "meta-optimizer" can be adapted to downstream tasks with only a few exemplars from the target task, without significant modifications of the AIO parameters. The performance of this approach is expected to depend on the neural model's learning capacity and the discrepancy among different tasks (Chen et al., 2021). With a white-box LLM as the learning model powered by fine-tuning, providing sufficient representation power, AIO can help acquire high-level instruction knowledge using meta-learning techniques.

G Additional Instruction Optimization Trajectory results

To provide additional insights for our instruction optimization process, apart from our instruction optimization trajectory results from Subsec. 5.2, we include complementary results on extra tasks, in terms of how generated instructions evolve over the fine-tuning process, as well as how their accuracy results change with the instruction optimization.

G.1 "Sentiment" Task

- Accuracy: 46%

IF the input contains any of the following words: "fun", "entertainingly", "splendid", "exquisite", "brilliantly", "impeccable", "likeable", "engrossing", "psychologically", "slyly" THEN output is positive., ELSE IF the input contains any of the following words: "worst", "dislikable", "backlash", "mess", "pastiche", "warmed" THEN output is negative.

- Accuracy: 91%

The output is positive if the input contains words with positive connotations, such as "fun", "entertainingly", "brilliantly", "exquisite", "likeable", "gentle", and "engrossing".
The output is negative if the input contains words with negative connotations, such as "worst", "low", "pretentious", "dislikable", "mess", "warmed over", and "backlash".

G.2 "Navigate" Task

- Accuracy: 33%

To determine if a robot will return to the starting point after a series of movements, follow these steps:
1. Count the total number of steps taken by the robot.
2. If the robot turns around at any point, it will change direction. So, count the number of turns and subtract 2 from the total number of steps for each turn.
3. If the robot always faces forward, it will not change direction. So, do not count the turns.
4. If the total number of steps minus the adjustments for turns is a multiple of the initial number of steps, the robot will return to the starting point. Example: If the robot takes 10 steps forward and then 5 steps backward, the total number of steps is 15. If the robot turns around at some point, adjust the total number of steps by subtracting 2 for each turn. If the robot always faces forward, do not adjust the total number of steps. If the total number of steps minus the adjustments is a multiple of the initial number of steps, the robot will return to the starting point.

- Accuracy: 71%

To determine if a robot will return to its initial position after a series of instructions, follow these steps:
1. Initialize the robot's position and direction to the starting point.
2. Iterate through the instructions: - If the instruction is "Turn around", update the robot's direction to the opposite direction. - If the instruction is "Turn right" or "Turn left", update the robot's direction accordingly. - If the instruction is "Take x steps", update the robot's position by moving x steps in the current direction.
3. After iterating through all instructions, check if the robot's final position is the same as the initial position.
4. If the final position is the same as the initial position, the output is True; otherwise, the output is False.

G.3 "Object Counting" Task

- Accuracy: 27%

To infer the output for a given input, count the number of items in the input that are not fruits or vegetables. Then, return the count. For example, if the input is "I have a violin, a piano, an accordion, a clarinet, an apple, a trumpet, and a banana"., you would count the number of non-food items, which are 7 (violin, piano, accordion, clarinet, trumpet), and return 7. If the input is "I have a yam, a carrot, a cabbage, two garlics, a potato, a stalk of celery, a head of broccoli, and a cauliflower"., you would count the number of non-food items, which are 0, and return 0.

- Accuracy: 52%

To infer the output for a given input, follow these steps:
1. Identify the types of items in the input. For example, if the input contains "violin", "piano", "accordion", "clarinet", "ap-

ple", "trumpet", and "banana", the types of items are musical instruments and fruits.
2. Count the number of items of each type. For example, there is 1 violin, 1 piano, 1 accordion, 1 clarinet, 1 apple, 1 trumpet, and 1 banana.
3. Add the counts of all types of items together to get the total number of items.
4. The output is the total number of items. For example, if the input is "I have a violin, a piano, an accordion, a clarinet, an apple, a trumpet, and a banana"., the output would be 7.

G.4 "Taxonomy Animal" Task

- Accuracy: 0%

Sort the input list in descending order based on the length of the words. Then, select the words that are animals. Finally, sort the remaining words in descending order based on the length of the words.

- Accuracy: 95%

1. First, identify all the words that are not animals. These words are usually nouns that are not animals, such as food items, colors, objects, etc.
2. Sort the remaining animal words in alphabetical order.
3. The output is the list of animal words in alphabetical order, excluding the non-animal words.

G.5 "Larger Animal" Task

- Accuracy: 50%

1. Take the first animal from the input.
2. If the first animal is a predator, compare it with the second animal.
3. If the first animal is a predator and the second animal is not a predator, then the output is the second animal.
4. If the first animal is not a predator, compare it with the second animal.
5. If the first animal is not a predator and the second animal is a predator, then the output is the first animal.
6. If neither of the animals is a predator, compare their sizes.
7. If the first animal is larger than the second animal, then the output is the first animal.
8. If the first animal is smaller than the second animal, then the output is the second animal.
9. If the first animal and the second animal are of the same size, then the output is the first animal.

- Accuracy: 92%

If the input consists of two animals, compare their sizes. If the first animal is larger, output the first animal. If the first animal is smaller, output the second animal. If both animals have the same size, output the first animal.

G.6 "Implicatures" Task

- Accuracy: 62%

Given two speakers' dialogue, analyze the context and sentiment of the speakers' statements to determine if the second speaker is thanking the first speaker (output: 'yes') or not (output: 'no'). To do this, consider the following factors:
1. **Gratitude tone**: If the second speaker's statement conveys gratitude, appreciation, or a positive sentiment towards the first speaker, infer 'yes'.
2. **Question-answer format**: If the first speaker's statement is a question and the second speaker's response is an affirmative answer or a statement that implies agreement, infer 'yes'.
3. **Contrast**: If the first speaker's statement is a negative or neutral statement, and the second speaker's response is a contrasting positive statement, infer 'yes'.
4. **Contextual understanding**: Consider the context of the conversation and the speakers' intentions. If the second speaker's statement seems to be responding to the first speaker's action or suggestion in a positive manner, infer 'yes'.

When applying these factors, consider the nuances of language, idioms, and figurative expressions. If the analysis is ambiguous or unclear, default to 'no'.

- Accuracy: 84%

To infer the output for a given input, follow these steps:
1. Identify the speaker's tone and intention in the input. Are they expressing gratitude, surprise, or skepticism?
2. Look for words or phrases that convey a positive or negative sentiment. If the sentiment is positive, the output is likely to be 'yes'. If the sentiment is negative, the output is likely to be 'no'.
3. Check for words or phrases that indicate a question or a request. If the input contains a question, the output is likely to be 'yes' if the speaker is seeking confirmation or agreement. If the input contains a request, the output is likely to be 'yes' if the speaker is seeking permission or approval.
4. Consider the context of the conversation. Is the speaker discussing a specific event, situation, or decision? If so, the output may be influenced by the speaker's perspective or attitude towards the topic.
5. Use your judgment and common sense to make an educated guess about the output. If you're still unsure, try re-reading the input and re-evaluating the tone, sentiment, and context.
By following these steps, you can make an informed decision about the output for a given input.

G.7 "Epistemic Reasoning" Task

- Accuracy: 41%

For each input, check if the premise and hypothesis are about the same individual or group of individuals. If they are, the output is entailment. If they are not, the output is non-entailment.

- Accuracy: 76%

Given a premise and a hypothesis, determine the entailment relationship between them. For a premise, identify the main clause and the subordinate clause (if present). For the hypothesis, identify the main clause and the subordinate clause (if present). If the main clause in the premise is identical to the main clause in the hypothesis, and the subordinate clause (if present) is also identical, then the output is entailment. If the main clause in the premise is identical to the main clause in the hypothesis, but the subordinate clause (if present) is different, then the output is non-entailment. If the main clause in the premise is not identical to the main clause in the hypothesis, then the output is non-entailment.

G.8 "Timedial" Task

- Accuracy: 41%

1. Read the context carefully and identify the main idea or topic of the conversation.
2. Identify the speaker and the listener.
3. Determine the purpose of the conversation.
4. Identify the key information or events mentioned in the conversation.
5. Analyze the conversation to determine the relationship between the speakers and the context.
6. Based on the analysis, infer the output for the given input.

- Accuracy: 79%

To infer the output for a given input, follow these steps:
1. Identify the context and the entities mentioned in the input, such as people, places, and times.
2. Look for keywords and phrases that indicate the type of information being requested or discussed, such as dates, times, quantities, or specific events.
3. Analyze the relationships between the entities and the context, and try to identify the most relevant and important information.

4. If the output is a date or time, check if it is a specific date or time, or if it is a range or duration.
5. If the output is a quantity, check if it is a specific number, or if it is a range or interval.
6. Finally, use the inferred output to generate a response that is relevant and accurate to the input and the context.

G.9 "Question Selection" Task

- Accuracy: 39%

To infer the output for a given input, follow these steps:
1. Identify the context of the input, which typically consists of a passage of text.
2. Determine the type of question being asked, which is usually indicated by the choice options provided.
3. Look for specific keywords or phrases in the input that are related to the question being asked.
4. Identify the relevant information in the input that answers the question, which may be a specific fact, statistic, or quote.
5. Match the relevant information to the corresponding choice option in the question.
6. Select the choice option that best answers the question based on the information provided in the input.

- Accuracy: 64%

To infer the output for a given input, follow these steps:
1. Identify the context of the input, which is typically a passage of text.
2. Determine the type of question being asked, which is usually indicated by the format of the choices provided.
3. Scan the context to identify relevant information, such as key phrases, names, and dates.
4. Match the context to the corresponding choices, considering the question type and the information gathered.
5. Select the most likely output based on the context and the choices.

For example, if the input is a passage about a football game, the output might be a question about the teams involved, the score, or the MVP. If the input is a passage about a historical event, the output might be a question about the date, location, or significance of the event.

G.10 "Movie Recommendation" Task

- Accuracy: 48%

1. Identify the most frequent genre or theme in the given context.
2. Compare the given context with the choices and find the one that best matches the identified genre or theme.
3. Output the matching choice.

- Accuracy: 88%

The output is the choice that is most commonly associated with the given context, based on the frequency of co-occurrences of movies in the input list with each choice.