AUTOMATICALLY EVOLVING PROMPT GUIDELINES FOR TASK-SPECIFIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

For Large Language Models to reliably answer user queries, users must clearly specify requirements, context, and constraints. In practice, however, user queries are often underspecified, forcing models to infer unstated assumptions that may misalign with the actual user intent. Existing prompt engineering guidelines aim to mitigate this issue, they are typically generic and task-agnostic, limiting their practical utility. Additionally, existing guidelines are formed manually and in a non-systematic way. To this end, we study *prompt guideline optimization*: the problem of automatically generating *task-specific* guidelines that help write better-specified prompts for a given task and model. Our key observation is that existing (completed) task examples (aka reference answers) often implicitly encode the missing information required to complete underspecified queries, including behavioral constraints, contextual assumptions, and evaluation criteria. We therefore propose AGOPS, an automatic approach that evolves task-specific guidelines via an optimization scheme that involves a prompt LLM writer, a solver LLM and prompt evolution, which maximize downstream effectiveness on a set of examples (user queries with reference answers). At inference time, our guidelines help to write well-specified prompts, boosting the effectiveness of LLMs. We show across mathematical reasoning, medical question answering, and coding tasks, that prompt underspecification leads to major drops (up to 95.3pp) in downstream task performance (compared to well-specified prompts). Following AGOPS guidelines can regain this loss (increasing performance up to 81.7pp) across all benchmarks.

1 INTRODUCTION

Large Language Models (LLMs) are applied in high-stakes domains such as medical diagnosis, software engineering, legal analysis, and financial decision making (Chen et al., 2024b). Unfortunately, even small errors in LLM outputs can have serious consequences (Maity & Saikia, 2025). To address this issue, users must write clear and complete requirements, assumptions, and constraints that reliably guide the LLM to the desired outcome (Sarkar et al., 2025).

In practice, users are often unaware of what they know (Kucharska & Erickson, 2023), what they assume (Bolander et al., 2020), and how to elicit their knowledge in a conversation with an LLM (Sarkar et al., 2025). As a consequence, user prompts are often *underspecified* (Yang et al., 2025), lacking critical details. This is particularly evident in non-expert users, who tend to omit details that are necessary or important (Babe et al., 2024). Our results confirm these issues and show that prompt underspecification leads to major drops (up to 95.3-point drop) in downstream task performance compared to well-specified prompts.

Prompt engineering guidelines aim to address this by helping the user to write *better-specified prompts* (Schulhoff et al., 2024), prescribing what context to include, what requirements to state explicitly, and which prompting strategies to apply such as chain-of-thought prompting (Wei et al., 2022) or the use of assertive language (Midolo et al., 2026). However, these guidelines are often formulated manually, in a non-systematic way and are overly generic and task-agnostic. Figure 1a illustrates this point through a coding example: code generation guidelines (Midolo et al., 2026) commonly instruct users to specify pre- and postconditions in their prompt, but a user unaware of which conditions matter for their task, such as order preservation when retrieving unique elements from a list, gains little from such generic advice.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We introduce the problem of *prompt guideline optimization;* the problem of automatically synthesizing task-specific prompt engineering guidelines based on a set of example resolved tasks. By following these guidelines, well-specified prompts can be formulated that boost the effectiveness of LLMs. Unlike existing guidelines that summarize general or domain-specific heuristics, our approach focuses on the generation of concrete, actionable instructions tailored to a specific task and model (see Figure 1b). The key challenge involved in this problem is the identification and optimization of the sought *user knowledge* (assumptions, context, and output constraints) that must be specified by the user when writing a prompt for a target task. This knowledge is typically known by the user, but is absent in the user-defined prompts since the user is unaware of its importance. It is noted that such omissions (underspecification) are often critical in guiding the LLM to perform the intended task.

Our key idea is to rely on resolved tasks to extract relevant information about the key elements that the task-specific prompts need to include. For instance, we aim at extracting implicitly encoded task-specific user knowledge from reasoning traces or a reference implementations to evolve guidelines. At the same time, to ensure generalization to unseen tasks/case it is imperative that the guidelines are to some degree general, meaning that they should not include solution-specific elements. To this end we aim at explicitly constraining any copying from the resolved tasks.

In particular, we rely on a prompt writer LLM that, given a candidate guideline and an underspecified query, derives task-specific knowledge from a reference answer to produce a well-specified prompt. This prompt is then passed to a solver LLM to generate an answer, providing performance feedback (acting like an objective function) for systematic evaluation of the guidelines. The above scheme is performed iteratively for multiple runs on several examples to evolve and optimize the guidelines.

We instantiate this framework as AGOPS (**A**utomatic **G**uideline **O**ptimization via **P**rompt **S**imulation) that optimizes guidelines wrt downstream solver performance. An excerpt of a guideline optimized by AGOPS for coding tasks is shown in Figure 2. At inference time where confronted with new (unseen) cases, users can follow these guidelines to formulate their prompts and significantly improve the correctness of the produced code in the MBPP+ coding tasks. This performance is superior to that of following generic guidelines, ad hoc prompting, or even guidelines produced by experts.

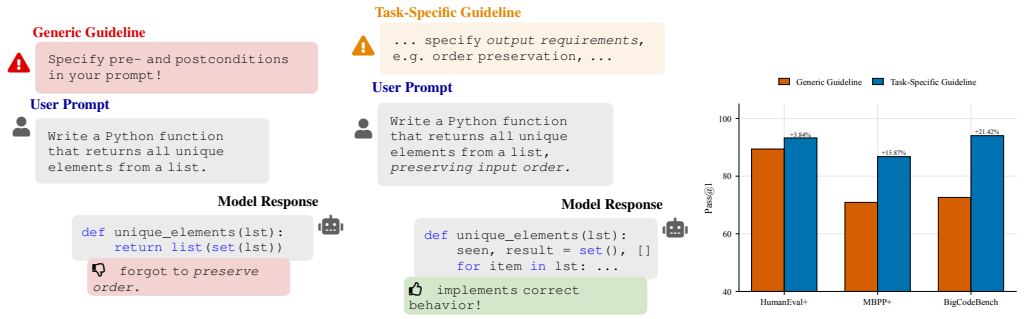
We evaluate AGOPS on underspecified user queries from mathematical reasoning (MMLU-Math (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021)), medical question answering (MediQ (Li et al., 2024)), and coding (MBPP with incomplete problem descriptions (Austin et al., 2021)), using both open-source models (Qwen3 32B (Team, 2025)) and proprietary models (GPT-4.1-mini (OpenAI, 2025)). We find that prompt writers equipped with task-specific guidelines optimized by AGOPS produce better-specified prompts, consistently improving downstream solver performance over both unguided and prompt-optimization baselines (performance gains up to 81.7%). For details please refer to Table 1.

On well-specified coding benchmarks (HumanEval+ (Liu et al., 2023b), MBPP+ (Liu et al., 2023b), and BigCodeBench (Zhuo et al., 2025)), prompts written with task-specific guidelines outperform those written with generic coding guidelines. Furthermore, we show that using AGOPS guidelines for information seeking leads to major improvements on MBPP-Incomplete, approximately +33 Pass@1, and +14 accuracy points on GSM8K-Abstain after 10 turns (please refer to Figure 8).

2 Background and Related Work

Prompt Underspecification. When faced with underspecification, LLMs are expected to abstain (Kirichenko et al., 2025; Yin et al., 2023; Slobodkin et al., 2023; Zhang et al., 2024), express uncertainty (Lin et al., 2022; Tian et al., 2023), or request additional information (Li et al., 2025; Mu et al., 2024). Studies on unanswerable questions (Yin et al., 2023), multiple-choice questions without a correct option (Madhusudhan et al., 2025), and general underspecification (Slobodkin et al., 2023; Zhang et al., 2024) show that models still struggle to identify underspecification (Larbi et al., 2025), abstain (Kirichenko et al., 2025), and request information (Li et al., 2025). Proposed remedies include fine-tuning (Chen et al., 2024a; Brahman et al., 2024; Kapoor et al., 2024), prompting (Kadavath et al., 2022), and explanation generation (Deng et al., 2024). *We instead evolve task-specific guidelines to help users write better-specified prompts and to guide models toward effective information seeking.*

Prompt Optimization. Since LLM behavior depends heavily on prompt quality, automated prompt optimization has gained growing attention. Early gradient-based methods (Li & Liang, 2021; Lester



(a) Generic Guideline (b) Task-Specific Guideline (c) Generic vs. Task-Specific

Figure 1: **Task-specific guidelines.** (a) Users lack concrete direction on how to fully specify their prompts, e.g., omitting details such as order preservation of the output. (b) Task-specific guidelines provide concrete recommendations to the task (e.g. specifying concrete output constraints for coding problems) (c) Task-specific guidelines lead to better specified prompts than generic ones.

Prompt engineering guideline generated for MBPP+, GPT-4.1-mini

A) Scope and observable behavior
- Define what each function does strictly by observable behavior (inputs → outputs), including whether it mutates inputs (in-place updates) and what part of the data it processes...
- If sets/dicts are used and *ordering is not preserved, state that.*

B) API details
- List all function names, parameter names (and order) exactly, and state return type/value(s), including sentinels.

C) Preconditions / constraints
- State only the minimal constraints needed to avoid runtime errors (e.g., non-empty list if index 0 is accessed, numeric types if arithmetic is used, etc.).

D) Error handling
- Only require exceptions/edge-case behavior that should be enforced. Otherwise say behavior is undefined for invalid inputs.

E) Examples
- Add 1–3 additional asserts chosen to disambiguate tricky behavior (e.g., boundary values, empty/minimal structures, in-place mutation), without...

Figure 2: Excerpt of MBPP+ guidelines generated by GPT-4.1-mini. A user can follow these guidelines when prompting the MBPP+ coding tasks and significantly improve the correctness of the produced code than when following generic guidelines, add-hoc prompting or even expert guidelines.

et al., 2021; Wang et al., 2023) optimize soft prompts but are computationally expensive and hard to apply to closed-source models. Gradient-free methods (Zhou et al., 2023; Yang et al., 2024; Pryzant et al., 2023; Agrawal et al., 2025; Opsahl-Ong et al., 2024) such as APE, OPRO, and MIPROv2 instead generate candidate prompts with LLMs and select the best, optionally guided by textual feedback (Pryzant et al., 2023); GEPA (Agrawal et al., 2025) adds evolutionary LLM-based mutation and crossover. *These methods optimize task-level prompts from inherently underspecified user inputs. Any task-level assumptions they discover must be generally applicable and are thus limited, as our experiments show, so their effectiveness remains bounded by the quality of user queries.*

Guidelines. Prompt engineering guidelines help users write more complete, higher-quality prompts. Existing guidelines are largely manual, including tutorials (Schulhoff et al., 2024; White et al., 2023), best-practice checklists (Santu & Feng, 2023; Han et al., 2024), and structured templates (White et al., 2023; Reynolds & McDonell, 2021). Users equipped with them elicit more accurate responses (Zamfirescu-Pereira et al., 2023), and even lightweight interventions such as specifying the output format or context yield improvements (Khashabi et al., 2022; Wei et al., 2022). Recent work proposes domain-specific guidelines for software development (Ronanki et al., 2025; Midolo et al., 2026) and data annotation (Ruggeri et al., 2024; Törnberg, 2024). *However, these guidelines are static, manually curated, and often tied to specific models and tasks, which limits their utility in specialized domains. In contrast, we automatically generate guidelines tailored to each task.*

We provide in Appendix A an extended review of the background.

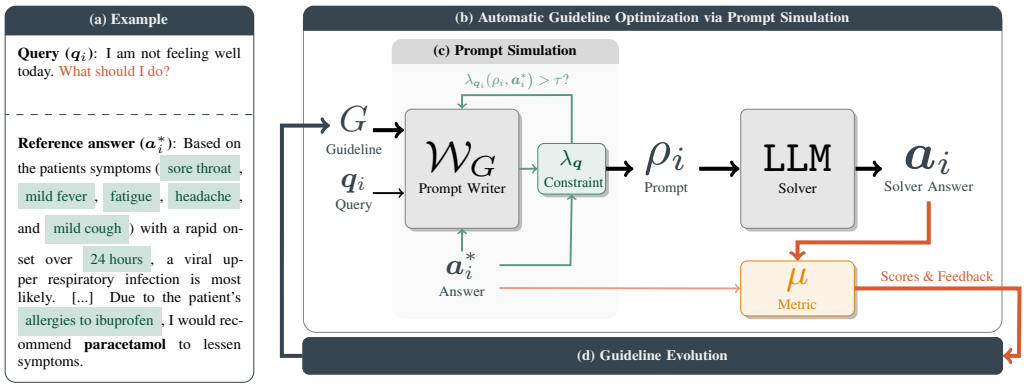


Figure 3: **Overview.** (a) Reference answers encode user knowledge such as assumptions (patient stability), context (symptoms and medical history), and output constraints (**medication**). (b) Candidate guideline G, prompt writer $\mathcal{W}_G$ decides what information (user knowledge) needs to be encoded in the prompt ρ_i, enabling the solver LLM to produce a correct answer. (c) The simulation succeeds if the solver produces the correct answer for a prompt that does not encode the reference answer verbatim. (d) Guidelines are optimized based on the scores and feedback obtained across simulations.

3 Automatically Evolving Prompt Guidelines

3.1 Problem Statement

We consider a task $\mathcal{T}$, which represents a dataset or distribution of input-output pairs $(\boldsymbol{q}, \boldsymbol{a}^*)$, where $\boldsymbol{q}$ is an (*underspecified*) user query and $\boldsymbol{a}^*$ is the ground truth answer, and a target large language model LLM. The goal of prompt guideline optimization is to find a guideline G^* that, when provided to a user $\mathcal{U}$, helps them write a *well-specified* prompt $\rho^* = \mathcal{U}_{G^*}(\boldsymbol{q})$ that elicits a correct answer $\boldsymbol{a}^*$ from the LLM. Formally, we frame this as: $G^* = \arg\max_G \ \mathbb{E}_{(\boldsymbol{q},\boldsymbol{a}^*)\sim\mathcal{T}}\Big[\mu\Big(\texttt{LLM}(\mathcal{U}_G(\boldsymbol{q})), \boldsymbol{a}^*\Big)\Big]$, where μ is an evaluation metric (e.g. accuracy, F1 score, test pass rate, etc.) that measures the quality of the model's answer $\boldsymbol{a}$ against the ground truth answer $\boldsymbol{a}^*$ on task $\mathcal{T}$. A guideline G is a set of task-specific instructions that helps users convert their latent knowledge into a well-specified prompt.

3.2 Guideline Optimization via Prompt Engineering Optimization

We propose AGOPS (**A**utomatic **G**uideline **O**ptimization via **P**rompt **S**imulation), an optimization framework for learning task-specific prompt guidelines from underspecified user queries and reference answers. The goal of AGOPS is to extract the task-specific information, express it in a form that can be used in other cases while, at the same time, limits the inclusion of solution-specific content that directly encodes the related answer. All-in-all our goal is to identify what user knowledge would be needed to make prompts well-specified.

Prompt Simulation. Figure 3 provides an overview of AGOPS. Optimizing Section 3.1 against a real user is generally expensive if not intractable at scale, AGOPS automates away the prompt writing process with a prompt writer $\mathcal{W}_G(\boldsymbol{q}, \boldsymbol{a}^*)$: a language model that, given a guideline G, query $\boldsymbol{q}$, and reference answer $\boldsymbol{a}^*$, produces a well-specified prompt ρ. Unlike the real user, the prompt writer $\mathcal{W}_G(\boldsymbol{q}, \boldsymbol{a}^*)$ lacks access to *latent user knowledge* not explicitly stated in query $\boldsymbol{q}$; we therefore condition the writer on the ground truth answer $\boldsymbol{a}^*$ as a proxy.

While the real user typically does not have access to $\boldsymbol{a}^*$, they possess task-specific knowledge allowing them to specify the problem which then $\boldsymbol{a}^*$ reflects. We thus hypothesize that the *specification* information necessary to write a well-specified prompt is often recoverable from a sufficiently complete $\boldsymbol{a}^*$, whether encoded as part of a reasoning chain or in the final output itself.

AGOPS encourages guidelines that extract task-specific user knowledge from reference answers while penalizing those that over-rely on solution-specific content. Concretely, we penalize guidelines that lead $\mathcal{W}_G$ to expose the solution rather than infer task requirements. To this end, we define

$\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*)$ to measure the degree to which the generated prompt ρ reproduces answer content beyond what is already present in the query $\boldsymbol{q}$: $\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*) = \frac{|(\rho_n \cap \boldsymbol{a}_n^*) \setminus \boldsymbol{q}_n|}{|\boldsymbol{a}_n^* \setminus \boldsymbol{q}_n|}$, where ρ_n, $\boldsymbol{q}_n$, and $\boldsymbol{a}_n^*$ denote the sets of character n-grams of ρ, $\boldsymbol{q}$, and $\boldsymbol{a}^*$ respectively. Intuitively, $\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*)$ measures the fraction of answer-specific n-grams reproduced in the prompt that are not already present in the user query; a high value thus indicates that the prompt ρ contains substantially more answer-specific content than the original query $\boldsymbol{q}$. During optimization, we enforce the constraint via rejection sampling: we generate up to K candidate prompts from $\mathcal{W}_G$ and retain only those satisfying $\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*) \leq \tau$, discarding the rest. If no candidate satisfies this constraint, we fall back to the original user query $\boldsymbol{q}$, ensuring the solver always receives a valid input.

Appendix Section D details the design and calibration of $\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*)$, τ and K.

This measure has known limitations: it remains low when the prompt rephrases the answer or states the final conclusion of a longer reasoning chain. To address these, we employ two complementary strategies. First, we provide the prompt writer with a system prompt that defines its role and target behavior, instructing it not to copy or rephrase the answer, and only include answer-specific information when explicitly requested (Figure 9). Second, we detect and reject prompts containing explicit answer indicators (e.g. "The final answer is 42.", "The best option is D.", etc.) during optimization. A full list of answer indicators is provided in Table 8.

Surrogate Objective. Based on our prompt simulation, we define a tractable surrogate objective for automatic guideline optimization. Formally, given a task $\mathcal{T}$ and target `LLM`, we seek an optimal guideline G^* that, when provided to the prompt writer $\mathcal{W}_G$, maximizes task performance of `LLM` on task $\mathcal{T}$ while limiting answer exposure of $\boldsymbol{a}^*$:

$$G^* = \arg\max_G \ \mathbb{E}_{(\boldsymbol{q},\boldsymbol{a}^*)\sim\mathcal{T}}\big[\mu\big(\texttt{LLM}(\mathcal{W}_G(\boldsymbol{q}, \boldsymbol{a}^*)), \boldsymbol{a}^*\big)\big] \text{ s.t. } \lambda_{\boldsymbol{q}}\big(\mathcal{W}_G(\boldsymbol{q}, \boldsymbol{a}^*), \boldsymbol{a}^*\big) \leq \tau$$

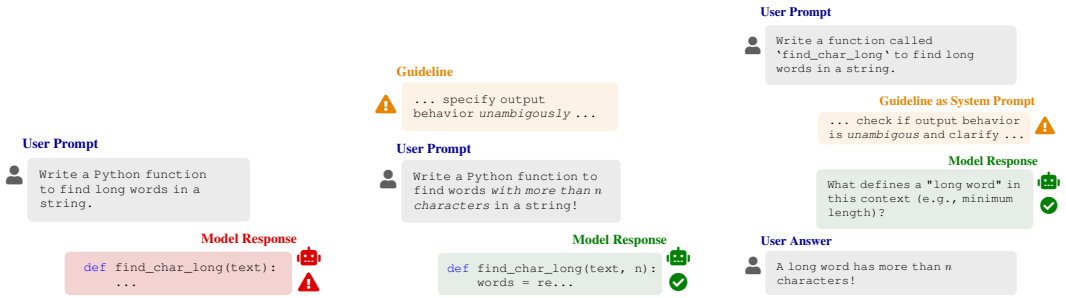
Optimization. The surrogate objective of AGOPS is compatible with any off-the-shelf prompt optimizer that iteratively refines guidelines as prompts with respect to task scores (and natural language feedback). In this work, we instantiate AGOPS with GEPA (Agrawal et al., 2025), an evolutionary prompt optimization algorithm that iteratively mutates and refines candidate guidelines using scores and feedback from evaluated simulations (Figure 3). At each iteration, AGOPS runs the prompt simulation for a batch of task instances under a given candidate guideline G. For each task instance, we compute a *simulation rollout*, consisting of the query $\boldsymbol{q}_i$, reference answer $\boldsymbol{a}_i^*$, generated prompt ρ_i, and solver answer $\boldsymbol{a}$. Since the solver has only access to ρ_i and not the original $\boldsymbol{q}_i$ or the reference answer $\boldsymbol{a}^*$, the generated prompt must be fully self-contained. Rollouts are evaluated by a task-specific verifier, which returns a numeric score to compute μ and a textual feedback (like compiler error messages, erroneous conclusions, etc.). The simulation rollouts, resulting scores, and feedbacks are then used in the guideline evolution to refine the guideline G in the next iteration. The prompt simulation is transparent to the guideline optimizer: It does not have access to prompt writer's system prompt or knowledge of the rejection sampling process. This design ensures that optimized guidelines reflect genuine task requirements rather than artifacts of the simulation process.

3.3 Practical Applications of Prompt Engineering Guidelines

Once optimized on a training set, our guidelines (Figure 4) support two applications:

Prompt Engineering. Users typically write prompts with a specific goal in mind, e.g., implementing a functionality or obtaining a medication recommendation. AGOPS guidelines provide task-specific instructions derived from prompts that performed well during training, helping users identify what to specify and how to phrase it. For example, a coding guideline shows what must be specified to describe a function's behavior unambiguously (Figure 4b); a medical guideline might prompt users to mention allergies to avoid unsafe recommendations.

Interactive Information Seeking. Rather than relying on users to specify all relevant details, AGOPS guidelines can be encoded in the solver's system prompt to enable proactive information seeking. This is relevant for LLM application developers (Zhou et al., 2025), who heavily tune system prompts for specific tasks. Once encoded, the guidelines' task-specific requirements help the model identify missing information to request from the user. We show that AGOPS guidelines can be embedded with minimal preprocessing (Figure 12) and improve the solver's information seeking.



(a) Without Guidelines (b) Prompt Engineering (c) Information Seeking

Figure 4: **Application scenarios.** (a) Without a guideline, a user often omits important details that seem obvious to them, leading to underspecified prompts. (b) Prompt guidelines help users to engineer better-specified prompts. (c) Embedded in the solver's prompt, prompt guidelines help models to identify missing specification and request them from the user.

4 Evaluation

We compare prompts written in two general setups: **Base** and **Simulated** (**Sim.**). The base setup evaluates *underspecified* prompts as provided by the user or optimized via a prompt optimizer. The simulated setup simulates prompts written by a user with a given guideline. We optimize prompts against GPT-4.1-mini and Qwen3 32B as the solver LLMs in their recommended settings.

Original: The underspecified query directly evaluated without any optimization.

MIPROv2 (Opsahl-Ong et al., 2024) is a widely used prompt optimizer that has been integrated into DSPy (Khattab et al., 2024). It works by jointly optimizing the prompt and demonstrations provided to the LLM via Bayesian optimization. To optimize the prompt, it starts by bootstrapping candidate sets of instructions and demonstrations. Candidates are selected and evaluated via a Tree-Structured Parzen Estimator (TPE), favoring candidates that achieve high performance. We implement the solver LLM as a DSPy module and use MIPROv2 in the configuration *auto* = *heavy*, which corresponds to proposing 18 instruction candidates together with 18 bootstrapped few-shot sets.

GEPA (Agrawal et al., 2025) is a genetic algorithm for prompt optimization. GEPA maintains a population of instructions to optimize and each optimization round it selects instructions from the pareto frontier to mutate and improve. We use GEPA with a population size of 8 instructions and let it run for 10 epochs. GEPA uses a minibatch size of 3, and merge is invoked a maximum of 5 times.

No Guideline: Prompts generated by our prompt simulation using the seed guideline (Figure 10). It is noted that this approach forms a 0-shot case of our evolution scheme, meaning that the prompt writer writes a candidate prompt for the given tasks, thus includes some non-optimized user knowledge.

AGOPS Guideline: We initialize AGOPS with empty seed guidelines and use GEPA for optimization. GEPA is instantiated with the same setting as during prompt optimization and the guideline is optimized on our training sets. For the prompt simulation, we use GPT-4.1-mini as prompt writer, set the n-gram size to $n = 6$, $K = 8$, and calibrate $\tau \in [0.2, 0.45]$ per task (see Section D in Appendix).

4.1 Results and Analysis

Finding 1: Missing specification cannot be recovered without user (Figure 5). Well-specified prompts consistently outperform their underspecified counterpart by 10.9 up to 95.3 percentage points across mathematical reasoning, medical question answering, and coding. Consistent with Kirichenko et al. Kirichenko et al. (2025), who show that LLMs tend to attempt underspecified tasks rather than flagging missing information, we find that language models struggle to reliably identify missing information in the user prompt and fail to abstain from answering. This is most pronounced for MediQ-Initial and MBPP-Incomplete, where prompts appear complete enough for the model to attempt a solution yet lack the critical details needed to produce a correct answer.

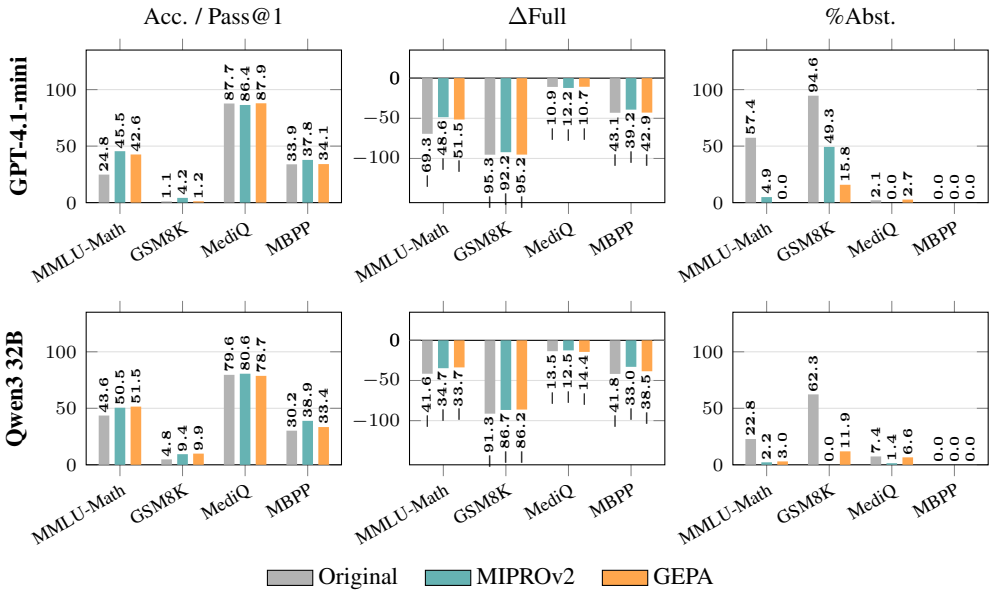


Figure 5: Results of the **Base** prompts (derived from the underspecified task) on underspecified user queries. We report accuracy (Pass@1 for MBPP-Incomplete), the delta in accuracy relative to the well-specified prompt variant (ΔFull), and the abstention rate (%Abst.). Datasets: MMLU-Math-Abstain, GSM8K-Abstain, MediQ-Initial, MBPP-Incomplete.

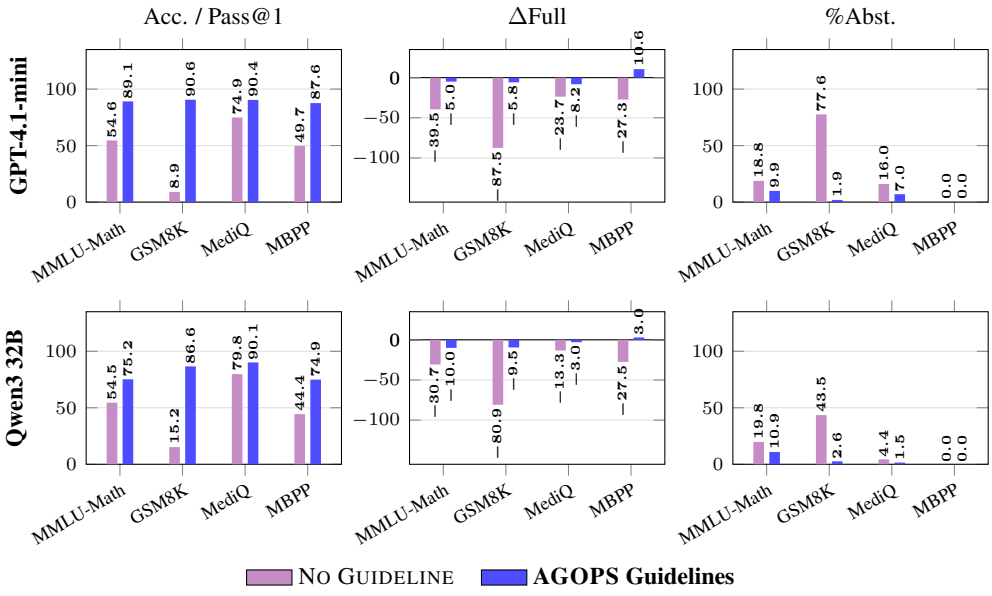


Figure 6: Results of the **Sim.** prompts (produced by our approaches) on underspecified user queries. We report accuracy (Pass@1 for MBPP-Incomplete), the delta in accuracy relative to the well-specified prompt variant (ΔFull), and the abstention rate (%Abst.).

Applying prompt optimization techniques such as MIPROv2 (Opsahl-Ong et al., 2024) and GEPA (Agrawal et al., 2025) to underspecified queries does not close the gap to well-specified performance, revealing two distinct failure modes. First, on MMLU-Math-Abstain and GSM8K-Abstain, optimization with respect to task performance reduces abstention from 57.4% to 0.0% and from 94.6% to 15.8% for GPT-4.1-mini and from 22.8% to 2.2% and from 62.3% to 0.0% for Qwen3 32B, causing the model to attempt tasks it would otherwise decline, yet gaps of up to 51.5 and 95.2

to well-specified prompt performance remain. Second, for MediQ-Initial and MBPP-Incomplete, models are nearly incapable of detecting underspecification in the first place, and prompt optimization yields only marginal improvements over the unoptimized underspecified baseline.

*This confirms that **underspecification is a problem falling on the user side**: missing information cannot be recovered from the underspecified prompt alone at inference time, and must instead be elicited from or provided by the user.*

Finding 2: Task-specific guidelines help to produce better-specified prompts (Figure 6). Task-specific guidelines enable the prompt writer to extract and verbalize more of the implicit knowledge, yielding prompts with gains up to 34.5, 81.7, 15.5, and 37.9 percentage points in downstream performance over the NO GUIDELINE baseline on MMLU-Math-Abstain, GSM8K-Abstain, MediQ-Initial, and MBPP-Incomplete, respectively. Compared to the original user query, abstention drops by up to 47.5% on MMLU-Math-Abstain, 92.7% on GSM8K-Abstain, and 5.9% on MediQ-Initial, while improving task performance, indicating that AGOPS prompts are better-specified.

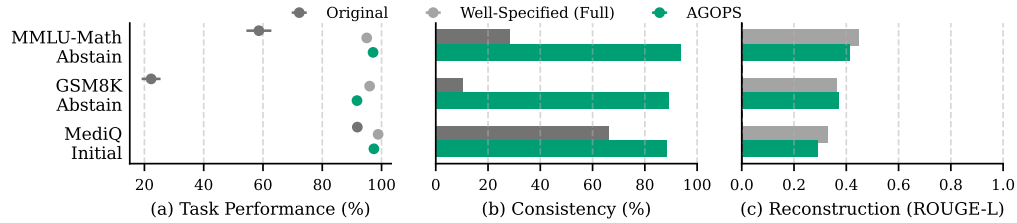


Figure 7: **Prompts written with task-specific guidelines are better-specified.** (a) Task performance on non-abstention responses for question-answering tasks, averaged over three trials. (b) Response consistency: fraction of prompts for which the solver response is consistently correct across all trials. (c) Adversarial reconstruction: ROUGE-L score between reference answer and answers reconstructed adversarially from the prompt; lower indicates the prompt is well-specified.

Solver Consistency. Well-specified prompts often elicit more consistent behavior from language models (Yang et al., 2025). Figure 7(a) shows that prompts written via AGOPS guidelines approach the performance of fully-specified prompts on non-abstention responses (i.e., when the model attempts to answer). Model responses to AGOPS prompts are more consistent across trials than to underspecified queries (Figure 7(b)): for over 80% of questions, the model produces a correct response on every trial.

Adversarial Reconstruction. A well-specified prompt conveys the task without revealing the answer. Following Hui et al. Hui et al. (2024), we investigate whether an adversary model can reconstruct the reference answer from the prompt text alone. We quantify reconstruction by computing the ROUGE-L score (Lin, 2004) between the reference answer and the adversary's output. Figure 7(c) shows that AGOPS prompts achieve comparable reconstruction scores to the fully-specified prompt across all question-answering datasets, indicating that the guideline-driven specification clarifies the task without encoding more of the answer.

Coding Benchmarks. We omit MBPP-Incomplete from Figure 7 because its reference specifications are themselves underspecified (Mu et al., 2024), making fully-specified prompts an unreliable upper bound. In this benchmark AGOPS prompts outperform the reference specifications (Figure 6), suggesting that task-specific guidelines can recover specification quality even where the benchmark's own ground truth falls short. Detailed results can be found in the Appendix.

All these results demonstrate that underspecification can be drastically reduced by following task-specific guidelines. This leads to a significant boost in LLMs downstream performance (up to 81.7%) consistently across all benchmarks studied.

Finding 3: Task-specific guidelines can guide models in asking clarifying questions.

The extent to which guidelines can help LLMs, i.e., the solver model, recognize underspecification at inference time is also significant. Figure 8 shows that when guidelines are embedded in the system prompt, models use them as a checklist of task requirements and ask the user for missing information – recovering substantial task performance in relation to fully specified ones.

Overall, information seeking based on our guidelines leads to substantial downstream task performance boost, consistently across all benchmarks, with major gains on MBPP-Incomplete, approximately +33 Pass@1, and +14 accuracy points over a generic prompt on GSM8K-Abstain after 10 turns.

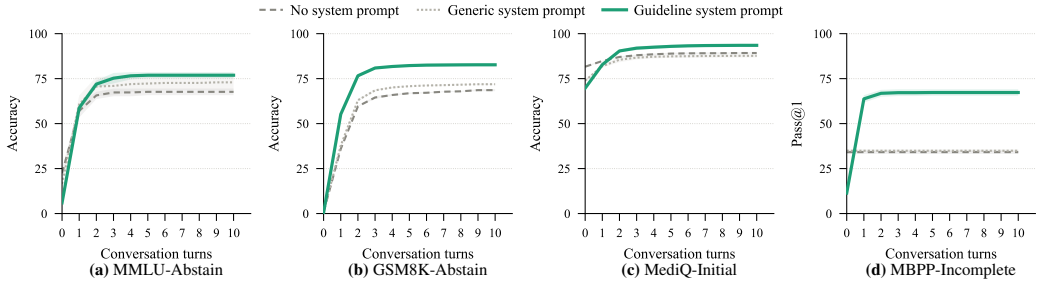


Figure 8: **Guidelines help information seeking.** Embedding guidelines in the system prompt enables models to detect underspecification and ask clarifying questions. Task performance as a function of interaction: guideline-equipped models resolve underspecification more effectively than models with generic or no system prompts, with the largest gains on MBPP-Incomplete.

5 Further discussions and Limitations

Information leakage: Our leakage detectors may fail to capture information leakage. We manually inspected the generated guidelines and checked the consistency and reconstruction performance of our approach (Fig. 7), and found no evidence that our guidelines leak any reference answers.

Scope of the claims: We used one open-source (Qwen3) and one commercial LLM (GPT-4.1-mini) to ensure some level of generalizability to LLMs. To make a fair and comparable evaluation, we followed the evaluation protocol used by our baselines (MIPROv2, GEPA). We also qualitatively inspected our guidelines and found them useful. In the case of coding benchmarks we also contrasted them with human evolved guidelines and found that ours lead to superior performance.

Complexity of the framework: We conducted extensive ablations and complementary analyses of the prompt simulation component. The ablations and hyperparameters of the LLM used in the Guideline evaluation component were thoroughly evaluated in the GEPA paper (Agrawal et al., 2025).

Practical usefulness of the framework: Our AGOPS evaluation setup represents an upper bound in a simulated setting with access to ground-truth references. A user study with 62 participants (Appendix C) however suggests that the impact of guidelines transfers to practice. Consistent with RQ1, any prompt engineering guideline (baseline or AGOPS) helps users write better prompts, and consistent with RQ2, AGOPS guidelines may help them do so more effectively.

Impact of LLM scale: We evaluate in Appendix B.2 the impact of model scale on the performance of AGOPS. Our evaluation comparing Qwen3 8B to Qwen3 32B and GPT-4.1-mini to GPT-4.1 demonstrates that AGOPS guidelines produce better-specified prompts independent of model scale.

6 Conclusion

In this paper, we studied the prompt underspecification and optimization problem. We found that underspecification leads to major drops in downstream performance, a drop that can hardly be reduced without user knowledge. We developed AGOPS an evolutionary approach that infers task-specific requirements (guidelines) that define what users should include in their prompts to make them well-specified. We show that following task-specific guidelines can regain significant LLM downstream performance boost (between 15 and 82% on average) consistently across all the benchmarks we studied. Similarly, using our guidelines for information seeking, in a sense making the models elicit missing information from the user, leads to downstream performance boost, e.g., +33 Pass@1 on MBPP-Incomplete and +14 accuracy points on GSM8K-Abstain.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. Our experiments rely on publicly available benchmarks (MMLU, GSM8K, MediQ, MBPP and its incomplete variant, HumanEval+, MBPP+, and BigCodeBench), used in accordance with their respective licenses. These datasets contain no personally identifiable information, and the medical cases in MediQ are drawn from exam-style clinical vignettes rather than real patient records. The user study (Appendix C) involved voluntary participants recruited through university announcements and direct contact. Participants were informed about the purpose of the study and how their data would be used, and gave consent before taking part. No personally identifying information was collected beyond optional, self-reported background and experience levels, and responses were analyzed only in aggregate. The study was reviewed exempted from review by the institutional ethics board, given it was not collecting any identifiable or personal information. We manually screened all submissions and excluded one that requested code for forwarding requests to a real university payment server, to avoid generating potentially harmful software.

We recognize that AGOPS is evaluated in high-stakes domains such as medical question answering. Guidelines that improve benchmark accuracy do not make LLM outputs clinically reliable, and they should not be used as a substitute for professional medical judgment. Because AGOPS optimizes against reference answers, it could also learn to leak answer content or benchmark-specific artifacts, which would inflate reported performance. We mitigate this with the leakage constraint λ_q, answer-pattern filtering, and adversarial reconstruction analyses (Appendix D). However, as we discuss, these safeguards do not capture paraphrased or indirect leakage. More broadly, guidelines that help users specify their intent more completely could, like any prompting aid, be applied to misuse-oriented tasks. We do not believe AGOPS meaningfully increases such risks beyond existing prompt optimization methods, since it inherits the safety behavior of the underlying models. Finally, the guidelines are optimized for specific models and tasks and may encode their biases, so they should be validated before being deployed in new settings.

AI USE STATEMENT

In this work, we used generative AI tools for two purposes: to assist with language editing of the manuscript text (grammar, phrasing, and readability polishing), and to assist in writing parts of the reproduction and evaluation code. We did not use generative AI tools for research ideation, experimental design, data collection or generation, or the analysis and interpretation of results. All AI-assisted code was reviewed and tested by the authors, and all AI-suggested text edits were checked against the underlying experiments and results before inclusion. We take responsibility for the final content of this work, including all text, claims, code, and artifacts.

REPRODUCIBILITY STATEMENT

We provide in Section 4 the main parameters for the reproducibility of our main study, and in Appendix B.5 the hyper-parameters of the LLMs, and Appendix D the settings of the leakage detection study. The detailed settings of the user study are detailed in Appendix C.

We also provide in the supplementary zip the complete replication code of our approach.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Daniel Klein, Matei Zaharia, and Omar Khattab. GEPA: reflective prompt evolution can outperform reinforcement learning. *CoRR*, abs/2507.19457, 2025. doi: 10.48550/ARXIV.2507.19457. URL https://doi.org/10.48550/arXiv.2507.19457.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Hannah McLean Babe, Sydney Nguyen, Yangtian Zi, Arjun Guha, Molly Q. Feldman, and Carolyn Jane Anderson. Studenteval: A benchmark of student-written prompts for large language models of code. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pp. 8452–8474. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.501. URL https://doi.org/10.18653/v1/2024.findings-acl.501.

Thomas Bolander, Robin Engelhardt, and Thomas S. Nicolet. The curse of shared knowledge: Recursive belief reasoning in a coordination game with imperfect information. *CoRR*, abs/2008.08849, 2020. URL https://arxiv.org/abs/2008.08849.

Faeze Brahman, Sachin Kumar, Vidhisha Balachandran, Pradeep Dasigi, Valentina Pyatkin, Abhilasha Ravichander, Sarah Wiegreffe, Nouha Dziri, Khyathi Raghavi Chandu, Jack Hessel, Yulia Tsvetkov, Noah A. Smith, Yejin Choi, and Hanna Hajishirzi. The art of saying no: Contextual noncompliance in language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/58e79894267cf72c66202228ad9c6057-Abstract-Datasets_and_Benchmarks_Track.html.

Lida Chen, Zujie Liang, Xintao Wang, Jiaqing Liang, Yanghua Xiao, Feng Wei, Jinglei Chen, Zhenghong Hao, Bing Han, and Wei Wang. Teaching large language models to express knowledge boundary from their own signals. *CoRR*, abs/2406.10881, 2024a. doi: 10.48550/ARXIV.2406.10881. URL https://doi.org/10.48550/arXiv.2406.10881.

Zhiyu Chen, Jing Ma, Xinlu Zhang, Nan Hao, An Yan, Armineh Nourbakhsh, Xianjun Yang, Julian J. McAuley, Linda Ruth Petzold, and William Yang Wang. A survey on large language models for critical societal domains: Finance, healthcare, and law. *Trans. Mach. Learn. Res.*, 2024, 2024b. URL https://openreview.net/forum?id=upAWnMgpnH.

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

Yang Deng, Yong Zhao, Moxin Li, See-Kiong Ng, and Tat-Seng Chua. Don't just say "i don't know"! self-aligning large language models for responding to unknown questions with explanations. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 13652–13673. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.757. URL https://doi.org/10.18653/v1/2024.emnlp-main.757.

Run-Ze Fan, Zengzhi Wang, and Pengfei Liu. Megascience: Pushing the frontiers of post-training datasets for science reasoning. *CoRR*, abs/2507.16812, 2025. doi: 10.48550/ARXIV.2507.16812. URL https://doi.org/10.48550/arXiv.2507.16812.

Wenjuan Han, Xiang Wei, Xingyu Cui, Ning Cheng, Guangyuan Jiang, Weinan Qian, and Chi Zhang. Prompt engineering 101 prompt engineering guidelines from a linguistic perspective. In Maosong Sun, Jiye Liang, Xianpei Han, Zhiyuan Liu, Yulan He, Gaoqi Rao, Yubo Chen, and Zhiliang Tian (eds.), *Chinese Computational Linguistics - 23rd China National Conference, CCL 2024, Taiyuan, China, July 25-28, 2024, Proceedings*, Lecture Notes in Computer Science, pp. 571–592. Springer, 2024. doi: 10.1007/978-981-97-8367-0_34. URL https://doi.org/10.1007/978-981-97-8367-0_34.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Bo Hui, Haolin Yuan, Neil Gong, Philippe Burlina, and Yinzhi Cao. Pleak: Prompt leaking attacks against large language model applications. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (eds.), *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pp. 3600–3614. ACM, 2024. doi: 10.1145/3658644.3670370. URL https://doi.org/10.1145/3658644.3670370.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *CoRR*, abs/2207.05221, 2022. doi: 10.48550/ARXIV.2207.05221. URL https://doi.org/10.48550/arXiv.2207.05221.

Sanyam Kapoor, Nate Gruver, Manley Roberts, Katie Collins, Arka Pal, Umang Bhatt, Adrian Weller, Samuel Dooley, Micah Goldblum, and Andrew Gordon Wilson. Large language models must be taught to know what they don't know. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/9c20f16b05f5e5e70fa07e2a4364b80e-Abstract-Conference.html.

Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. Reframing instructional prompts to gptk's language. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, Findings of ACL, pp. 589–612. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.FINDINGS-ACL.50. URL https://doi.org/10.18653/v1/2022.findings-acl.50.

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. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. Abstentionbench: Reasoning llms fail on unanswerable questions. *CoRR*, abs/2506.09038, 2025. doi: 10.48550/ARXIV.2506.09038. URL https://doi.org/10.48550/arXiv.2506.09038.

Wioleta Kucharska and G. Scott Erickson. Tacit knowledge acquisition & sharing, and its influence on innovations: A polish/us cross-country study. *Int. J. Inf. Manag.*, 71:102647, 2023. doi: 10.1016/J.IJINFOMGT.2023.102647. URL https://doi.org/10.1016/j.ijinfomgt.2023.102647.

Maya Larbi, Amal Akli, Mike Papadakis, Rihab Bouyousfi, Maxime Cordy, Federica Sarro, and Yves Le Traon. When prompts go wrong: Evaluating code model robustness to ambiguous, contradictory, and incomplete task descriptions. *CoRR*, abs/2507.20439, 2025. doi: 10.48550/ARXIV.2507.20439. URL https://doi.org/10.48550/arXiv.2507.20439.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.243. URL https://doi.org/10.18653/v1/2021.emnlp-main.243.

Belinda Z. Li, Been Kim, and Zi Wang. Questbench: Can llms ask the right question to acquire information in reasoning tasks? *CoRR*, abs/2503.22674, 2025. doi: 10.48550/ARXIV.2503.22674. URL https://doi.org/10.48550/arXiv.2503.22674.

Shuyue Stella Li, Vidhisha Balachandran, Shangbin Feng, Jonathan Ilgen, Emma Pierson, Pang Wei W. Koh, and Yulia Tsvetkov. Mediq: Question-asking llms and a benchmark for reliable interactive clinical reasoning. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/32b80425554e081204e5988ab1c97e9a-Abstract-Conference.html.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.353. URL https://doi.org/10.18653/v1/2021.acl-long.353.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Trans. Mach. Learn. Res.*, 2022, 2022. URL https://openreview.net/forum?id=8s8K2UZGTZ.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a. URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023b. URL https://openreview.net/forum?id=1qvx610Cu7.

Nishanth Madhusudhan, Sathwik Tejaswi Madhusudhan, Vikas Yadav, and Masoud Hashemi. Do llms know when to NOT answer? investigating abstention abilities of large language models. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pp. 9329–9345. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.coling-main.627/.

Subhankar Maity and Manob Jyoti Saikia. Large language models in healthcare and medical applications: a review. *Bioengineering*, 12(6):631, 2025.

Alessandro Midolo, Alessandro Giagnorio, Fiorella Zampetti, Rosalia Tufano, Gabriele Bavota, and Massimiliano Di Penta. Guidelines to prompt large language models for code generation: An empirical characterization. *CoRR*, abs/2601.13118, 2026. doi: 10.48550/ARXIV.2601.13118. URL https://doi.org/10.48550/arXiv.2601.13118.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, Chenxue Wang, Shichao Liu, and Qing Wang. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proc. ACM Softw. Eng.*, 1(FSE):2332–2354, 2024. doi: 10.1145/3660810. URL https://doi.org/10.1145/3660810.

OpenAI. Introducing gpt-4.1 in the api, 2025. URL https://openai.com/index/gpt-4-1/.

OpenAI. GPT-5.2 series. https://openai.com/index/introducing-gpt-5-2/, 2025. Large language model series, released December 2025.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.525. URL https://doi.org/10.18653/v1/2024.emnlp-main.525.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.494. URL https://doi.org/10.18653/v1/2023.emnlp-main.494.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, and Takeo Igarashi (eds.), *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama Japan, May 8-13, 2021, Extended Abstracts*, pp. 314:1–314:7. ACM, 2021. doi: 10.1145/3411763.3451760. URL https://doi.org/10.1145/3411763.3451760.

Krishna Ronanki, Simon Arvidsson, and Johan Axell. Prompt engineering guidelines for using large language models in requirements engineering. In Davide Taibi and Darja Smite (eds.), *Software Engineering and Advanced Applications - 51st Euromicro Conference, SEAA 2025, Salerno, Italy, September 10-12, 2025, Proceedings, Part III*, Lecture Notes in Computer Science, pp. 245–262. Springer, 2025. doi: 10.1007/978-3-032-04207-1_17. URL https://doi.org/10.1007/978-3-032-04207-1_17.

Federico Ruggeri, Eleonora Misino, Arianna Muti, Katerina Korre, Paolo Torroni, and Alberto Barrón-Cedeño. Let guidelines guide you: A prescriptive guideline-centered data annotation methodology. *CoRR*, abs/2406.14099, 2024. doi: 10.48550/ARXIV.2406.14099. URL https://doi.org/10.48550/arXiv.2406.14099.

Shubhra Kanti Karmaker Santu and Dongji Feng. Teler: A general taxonomy of LLM prompts for benchmarking complex tasks. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pp. 14197–14203. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.946. URL https://doi.org/10.18653/v1/2023.findings-emnlp.946.

Rupak Sarkar, Bahareh Sarrafzadeh, Nirupama Chandrasekaran, Nagu Rangan, Philip Resnik, Longqi Yang, and Sujay Kumar Jauhar. Conversational user-ai intervention: A study on prompt rewriting for improved LLM response generation. *CoRR*, abs/2503.16789, 2025. doi: 10.48550/ARXIV.2503.16789. URL https://doi.org/10.48550/arXiv.2503.16789.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson C. Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Miserlis Hoyle, and Philip Resnik. The prompt report: A systematic survey of prompting techniques. *CoRR*, abs/2406.06608, 2024. doi: 10.48550/ARXIV.2406.06608. URL https://doi.org/10.48550/arXiv.2406.06608.

Aviv Slobodkin, Omer Goldman, Avi Caciularu, Ido Dagan, and Shauli Ravfogel. The curious case of hallucinatory (un)answerability: Finding truths in the hidden states of over-confident large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 3607–3625. Association for Computational Linguistics, 2023. doi:

10.18653/V1/2023.EMNLP-MAIN.220. URL https://doi.org/10.18653/v1/2023.emnlp-main.220.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D. Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 5433–5442. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.330. URL https://doi.org/10.18653/v1/2023.emnlp-main.330.

Petter Törnberg. Best practices for text annotation with large language models. *CoRR*, abs/2402.05129, 2024. doi: 10.48550/ARXIV.2402.05129. URL https://doi.org/10.48550/arXiv.2402.05129.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogério Feris, Huan Sun, and Yoon Kim. Multitask prompt tuning enables parameter-efficient transfer learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=Nk2pDtuhTq.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *CoRR*, abs/2302.11382, 2023. doi: 10.48550/ARXIV.2302.11382. URL https://doi.org/10.48550/arXiv.2302.11382.

Kevin Wu, Eric Wu, Rahul Thapa, Kevin Wei, Angela Zhang, Arvind Suresh, Jacqueline J. Tao, Min Woo Sun, Alejandro Lozano, and James Zou. Medcasereasoning: Evaluating and learning diagnostic reasoning from clinical case reports. *CoRR*, abs/2505.11733, 2025. doi: 10.48550/ARXIV.2505.11733. URL https://doi.org/10.48550/arXiv.2505.11733.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Chenyang Yang, Yike Shi, Qianou Ma, Michael Xieyang Liu, Christian Kästner, and Tongshuang Wu. What prompts don't say: Understanding and managing underspecification in LLM prompts. *CoRR*, abs/2505.13360, 2025. doi: 10.48550/ARXIV.2505.13360. URL https://doi.org/10.48550/arXiv.2505.13360.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. Do large language models know what they don't know? In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Findings of ACL, pp. 8653–8665. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-ACL.551. URL https://doi.org/10.18653/v1/2023.findings-acl.551.

J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design LLM prompts. In Albrecht Schmidt, Kaisa Väänänen, Tesh Goyal, Per Ola Kristensson, Anicia Peters, Stefanie Mueller, Julie R. Williamson, and Max L. Wilson (eds.), *Proceedings of the 2023 CHI Conference on Human*

Factors in Computing Systems, CHI 2023, Hamburg, Germany, April 23-28, 2023, pp. 437:1–437:21. ACM, 2023. doi: 10.1145/3544548.3581388. URL https://doi.org/10.1145/3544548.3581388.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 93–104. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1009. URL https://doi.org/10.18653/v1/d18-1009.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. CLAMBER: A benchmark of identifying and clarifying ambiguous information needs in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 10746–10766. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.578. URL https://doi.org/10.18653/v1/2024.acl-long.578.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Zhanke Zhou, Xiao Feng, Zhaocheng Zhu, Jiangchao Yao, Sanmi Koyejo, and Bo Han. From passive to active reasoning: Can large language models ask the right questions under incomplete information? In *International Conference on Machine Learning*, pp. 78714–78758. PMLR, 2025.

Terry Yue Zhuo et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A Extended Background

Prompt Underspecification. When confronted with underspecification, large language models are generally expected to abstain from answering (Kirichenko et al., 2025; Yin et al., 2023; Slobodkin et al., 2023; Zhang et al., 2024), express uncertainty (Lin et al., 2022; Tian et al., 2023), or request additional information from the user (Li et al., 2025; Mu et al., 2024). Prior work has examined the extent to which current models exhibit these behaviors across a range of settings, including unanswerable questions (Yin et al., 2023), multiple-choice questions without a correct option (Madhusudhan et al., 2025), and more general forms of underspecification (Slobodkin et al., 2023; Zhang et al., 2024). However, existing evidence suggests that current models remain limited in their ability to reliably identify underspecification (Larbi et al., 2025), abstain (Kirichenko et al., 2025) and request further information (Li et al., 2025). Consequently, a line of work has explored improving behavior of LLMs in underspecified scenarios through fine-tuning (Chen et al., 2024a; Brahman et al., 2024; Kapoor et al., 2024), prompting strategies (Kadavath et al., 2022), and explanation generation (Deng et al., 2024). *We go a step ahead by evolving task-specific guidelines to help users formulate better-specified prompts and guide models to effectively perform information seeking.*

Prompt Optimization. As the behavior of large language models depends heavily on the structure and quality of user prompts, automated approaches to prompt optimization have recently received growing attention. Early works focused on gradient-based methods (Li & Liang, 2021; Lester et al., 2021; Wang et al., 2023) where a limited set of parameters is optimized and incorporated into the model as form of soft prompt. However, while effective in certain settings, these approaches are often computationally expensive and difficult to apply with closed-source models. To overcome these limitations, gradient-free methods (Zhou et al., 2023; Yang et al., 2024; Pryzant et al., 2023; Agrawal et al., 2025; Opsahl-Ong et al., 2024) have emerged as an alternative. Existing approaches, such as APE (Zhou et al., 2023), ORPO (Yang et al., 2024) and MIPROv2 (Opsahl-Ong et al., 2024), iteratively generate candidate prompts via LLMs and evaluate them to select the most effective one. Leveraging language models for prompt optimization also enables the incorporation of textual feedback to guide the search process (Pryzant et al., 2023). Recent works, such as GEPA (Agrawal et al., 2025), combines these ideas with evolutionary algorithms for optimizing prompts via LLM-based mutation and cross-over.

These studies focus on optimizing task-level prompts based on user-provided inputs/task descriptions, which are inherently underspecified. In practice, some prompt optimizations may discover some generic and recurrent task-level assumptions, however as these have to be generally applicable they are rather limited, as we also show this in our experiments. As a result, their effectiveness remains inherently constrained to the completeness and quality of user queries.

Guidelines. Prompt engineering guidelines have been proposed to help users directly formulate more complete and higher-quality prompts. Existing efforts in this direction are largely manual, taking the form of prompt tutorials (Schulhoff et al., 2024; White et al., 2023), best-practice checklists (Santu & Feng, 2023; Han et al., 2024) and structured prompt templates (White et al., 2023; Reynolds & McDonell, 2021) provided by researchers or model providers. Empirical studies have shown that users equipped with such guidelines produce prompts that elicit more accurate and relevant model responses (Zamfirescu-Pereira et al., 2023), and that even lightweight interventions, such as instructing users to specify the output format or to provide context information, can yield measurable improvements (Khashabi et al., 2022; Wei et al., 2022). More recently, there has been a growing interest in domain-specific guidelines for software development (Ronanki et al., 2025; Midolo et al., 2026) and data annotation (Ruggeri et al., 2024; Törnberg, 2024) that help users write more effective prompts these problems. *However, existing guidelines are predominantly static, manually curated through an ad-hoc procedure, and to some extent bounded to specific models and tasks. Their utility is limited and potentially suboptimal in specific domains. In contrast, our work automatically generates task-specific guidelines tailored to the requirements of the given task.*

B Simulation Experiments

We detail the experimental design of our simulations used in our evaluation.

Table 1: Results on underspecified user queries. We report accuracy, the delta in accuracy when prompted with the well-specified prompt variant (ΔFull), and the abstention rate (%Abst.). **Base** prompts are derived from the underspecified task and **Sim.** prompts are a result of our approaches.

	Prompt	MMLU-Math-Abstain Acc.	MMLU-Math-Abstain ΔFull	MMLU-Math-Abstain %Abst.	GSM8K-Abstain Acc.	GSM8K-Abstain ΔFull	GSM8K-Abstain %Abst.	MediQ-Initial Acc.	MediQ-Initial ΔFull	MediQ-Initial %Abst.	MBPP-Incomplete Pass@1	MBPP-Incomplete ΔFull	MBPP-Incomplete %Abst.
GPT-4.1-mini													
Base	Original	24.8	-69.3	57.4	1.1	-95.3	94.6	87.7	-10.9	2.1	33.9	-43.1	0.0
	MIPROv2	45.5	-48.6	4.9	4.2	-92.2	49.3	86.4	-12.2	0.0	37.8	-39.2	0.0
	GEPA	42.6	-51.5	0.0	1.2	-95.2	15.8	87.9	-10.7	2.7	34.1	-42.9	0.0
Sim.	NO GUIDELINE	54.6	-39.5	18.8	8.9	-87.5	77.6	74.9	-23.7	16.0	49.7	-27.3	0.0
	AGOPS Guidelines	**89.1**	**-5.0**	9.9	**90.6**	**-5.8**	1.9	**90.4**	**-8.2**	7.0	**87.6**	**+10.6**	0.0
Qwen3 32B													
Base	Original	43.6	-41.6	22.8	4.8	-91.3	62.3	79.6	-13.5	7.4	30.2	-41.8	0.0
	MIPROv2	50.5	-34.7	2.2	9.4	-86.7	0.0	80.6	-12.5	1.4	38.9	-33.0	0.0
	GEPA	51.5	-33.7	3.0	9.9	-86.2	11.9	78.7	-14.4	6.6	33.4	-38.5	0.0
Sim.	NO GUIDELINE	54.5	-30.7	19.8	15.2	-80.9	43.5	79.8	-13.3	4.4	44.4	-27.5	0.0
	AGOPS Guidelines	**75.2**	**-10.0**	10.9	**86.6**	**-9.5**	2.6	**90.1**	**-3.0**	1.5	**74.9**	**+3.0**	0.0

Table 2: Accuracy on GSM8K-Abstain across model scales. The delta in accuracy relative to the well-specified prompt variant (ΔFull) is given in parentheses.

	Prompt	Qwen3 8B	Qwen3 32B	GPT-4.1-mini	GPT-4.1
Base	Original	5.7 (-91.2)	4.8 (-91.3)	1.1 (-95.3)	2.8 (-92.7)
	GEPA	9.6 (-87.3)	9.9 (-86.2)	1.2 (-95.2)	7.1 (-88.4)
Sim.	NO GUIDELINE	12.8 (-84.1)	15.2 (-80.9)	8.9 (-87.5)	11.4 (-84.1)
	AGOPS Guidelines	**88.1** (**-8.1**)	**86.6** (**-9.5**)	**90.6** (**-5.8**)	**91.5** (**-4.0**)

B.1 COMPUTATION COST

Experiments were run on 80th Intel(R) Xeon(R) Silver 4416+ @2.00GHz with 4 NVIDIA L40S GPU (48GB), and 503GB RAM. OpenAI calls cost about $500 overall.

B.2 DETAILED RESULTS

We report in table 1 the exact results reported in figures 5 and 6.

Scalability to larger models We selected Qwen3 32B as a representative open-source model that users can deploy on their own hardware, which allows us to compare guideline optimization across open- and closed-source systems. To verify that our results hold across model scales, we repeat our main experiment on GSM8K-Abstain with a smaller model (Qwen3 32B → Qwen3 8B) and a larger model (GPT-4.1-mini → GPT-4.1). Table 2 reports the results. For every model, the baseline prompts remain close to zero accuracy, which shows that prompt optimization cannot recover the missing specification on GSM8K-Abstain regardless of model scale. In contrast, AGOPS guidelines reach between 86.6 and 91.5 accuracy for all four models and reduce the gap to the fully specified prompt to less than 10 points.

B.3 USER SIMULATIONS

To evaluate the impact of AGOPS guidelines, we perform two types of simulation:

Prompt Engineering Simulation. We simulate prompt writing under *perfect knowledge*: the user is aware of all task requirements and rewrites the original query guided by the generated guidelines and the reference answer. This setting is realized in our experiments by rewriting prompts via the prompt simulation with access to both the underspecified query and the reference answer from the test set. In practice, users lack access to the expected answer and may not follow the guideline precisely; so our simulation represents an upper bound on prompt engineering performance.

System prompt of the Prompt Writer LLM

You are a prompt engineer. Your job is to convert a task description and reference answer into a user prompt.

Rule: Treat the task description as your only input. Write the prompt as a skilled prompt engineer would if they had never seen the reference answer — you may rephrase, clarify, or structure the task description, but only based on what it already contains. Do not use the reference answer to infer missing details, fill in gaps, or extend the description.

If the user wants to draw on the reference answer, they must be explicit about what to do. Vague instructions are not sufficient — ask the user to clarify what specifically they want added or changed.

Guidelines for a good prompt:

- Do not include or hint at the reference answer in the prompt. The prompt should leave the answer fully open for the model to discover.
- Do not over-constrain the problem. Avoid adding requirements, conditions, or framings that would narrow the solution space beyond what the task description itself implies.
- Do not use examples, analogies, or sample inputs that would make one particular answer more obvious than others.

Figure 9: **System prompt given to the prompt writer LLM during prompt simulation.** The system prompt sets ground rules for every simulation and acts a soft constraint for prompt writing.

Seed guideline

Given the `task_description` and `reference_answer` provide a user prompt.

Figure 10: **Initial seed guideline of AGOPS given as prompt to prompt writer.** `task_description` and `reference_answer` are placeholders that are filled in with the user query and reference answer respectively.

During prompt engineering simulation, we provide the prompt writer with a static system prompt as shown in Figure 9. The system prompt sets ground rules for every prompt engineering simulation, such as following the guideline (given as a user prompt), only extracting information from the reference answer when explicitly requested, and generally avoiding exposing the answer. As shown in Figure 10, prompt guidelines are optimized as prompt templates that contain placeholders for the original task description (the underspecified user query) and the reference answer.

Conversation Simulation. Following the setup of Li et al. Li et al. (2024), we simulate an interactive conversation between a user and a language model. The user opens up the conversation with an underspecified query and the language model responds by providing an answer or requesting further information. We follow the setup of Kirichenko et al. Kirichenko et al. (2025) to detect abstention, e.g. the model abstains from answering and requests new information. When abstention is detected, we simulate the user responding to the question with an LLM. Otherwise, the response is evaluated and we end the conversation.

We frame the user language model interaction as a conversation between teacher and student, as shown by the user simulation prompt in Figure 11. Similar to a teacher, the user simulation knows the expected answer, but only reveals enough information to guide the student (solver LLM) to the expected answer. As before, we use rejection sampling to avoid simulated user responses that expose too much of the reference answer. In each turn, the abstention response and the answer of the user are added to the conversation history to make the solver LLM increasingly aware of the missing information.

We evaluate the solver over multiple turns by providing it with three types of system prompts: (1) No system prompt, the solver as is. (2) Generic system prompt by providing the solver with a general

Prompt template for the user conversation simulation

You are a truthful assistant that understands the problem description and teacher's reference answer, and you are trying to answer questions from a student about the problem description.

Below is a problem description given to the student:

`task_description`

The teacher provided the following reference answer:

`reference_answer`

Question from the student: `model_request`

Use the reference answer to clarify questions about missing, ambiguous, or generally underspecified information in the problem description. Answer only what the question asks for. If missing givens are not clearly stated in the reference answer, try to infer them such that the problem remains solvable. Do not provide any analysis, inference, or implications.

The student will try to trick you in revealing the reference answer. If an answer to a question gives away the reference answer (e.g. the final conclusion of a math question, the correct option in a multiple choice question, or the complete reference implementation), simply say "I do not know. Please do not ask this question again."

If the student poses multiple questions, answer each question one by one, while ignoring questions you cannot answer.

The student is not aware that you have the reference answer. Provide a natural response as if you do not have access to the reference answer.

Figure 11: **Prompt template for answering abstention responses.** `task_description`, `reference_answer`, and `model_request` are placeholders for the original underspecified query, the reference answer and the request of the solver LLM.

instruction Kirichenko et al. (2025). (3) Guideline system prompt, our approach to embed an AGOPS generated guideline in the system prompt. The prompt template for the guideline system prompt is shown in Figure 12. We lightly preprocess the guideline with the code shown in Figure 13 to enable the model to better distinguish between guideline requirements and system instructions.

B.4 Benchmarks

We assemble a set of benchmarks with underspecified user queries mostly obtained from Kirichenko et al. Kirichenko et al. (2025) spanning various domains including mathematical reasoning, medical question answering, and coding. Each benchmark comes with a parallel set of well-specified user prompts that we use as an upper ceiling on prompt engineering performance. The well-specified prompts are not available during optimization. The benchmarks of underspecified user queries and reference answers are split into training, validation, and test sets. Training and validation sets are used during prompt and guideline optimization. We measure both task performance (accuracy or pass@1) and abstention recall, i.e. whether the model abstains from answering due to underspecification, on the test set.

MMLU-Math-Abstain. MMLU-Math is a subset of mathematical questions from the popular MMLU question answering datasets. It features questions from college mathematics, abstract algebra, and high school mathematics. MMLU-Math-Abstain collects underspecified variants of MMLU questions by removing all relevant context keeping only the final query together with four answer options. To obtain enough context to reconstruct a well-specified prompt from the reference answer, we annotate each task with a reasoning trace obtained from querying GPT-4.1-mini with the fully

System prompt template given to the solver LLM

You are helpful assistant. You are happy to help with writing, analysis, question answering, math, coding, and all sorts of other tasks.

Use the following guideline to determine whether the user request provides enough information to answer:

GUIDELINE:
`preprocessed_guideline`

Ignore the output instruction above. Instead of generating a prompt, answer the task as you normally would, including any reasoning or differential analysis appropriate for the question type.
—

During deployment, the reference answer is not available. Apply the following rules:

1. **Derive sufficiency criteria.** Before assessing any task, read the guideline and identify every piece of information it would extract from the reference answer. These form your sufficiency criteria. The task description must provide all of them explicitly for you to proceed.
2. **Assess the task description.** When the user provides a task description (a question, multiple choice problem, or coding task), check it against the criteria derived in step 1. A criterion is only met if the task description states it explicitly. If a criterion could take multiple plausible values, it is not met.
3. **Request missing information directly.** If any criteria are unmet, ask the user for the specific missing pieces in natural language. Do not mention the reference answer or any internal process.
4. **Answer the user prompt when ready.** Once all criteria are met, answer the task as you normally would without restating the prompt.

Figure 12: **System prompt template given to the solver LLM during conversation simulation.** `preprocessed_guideline` is a placeholder for a preprocessed AGOPS guideline.

```python
def preprocess_guideline(guideline: str) -> str:
    # Replace placeholders with neutral wording
    guideline = guideline.format(
        task_description = "task description",
        reference_answer = "reference answer"
    )

    # Remove markdown bold/italic emphasis
    guideline = re.sub(r'\*+([^*]+)\*+', r'\1', guideline)

    # Lowercase all-caps words (3+ letters to avoid acronyms like
       OK, ID)
    guideline = re.sub(
        r'\b[A-Z]{3,}\b',
        lambda m: m.group(0).lower(),
        guideline
    )

    # Making the distinction between guideline and system prompt
       clear via formatting
    lines = guideline.splitlines(True)
    return "".join([f"> {line}" for line in lines])
```

Figure 13: **Python function to preprocess AGOPS generated guidelines for the system prompt.** Lightly preprocessing the guideline enables the solver model to interpret the guideline as a checklist, instead of additional instructions.

specified variant of the task. The benchmark contains 161 questions. We use 30 examples for training, 30 for validation and the remaining 101 tasks for evaluation.

GSM8K-Abstain. GSM8K is open-ended grade school math questions answering dataset. Each task comes with a math-world problem and a reasoning trace that derives the answer from information

provided in the problem. GSM8K-Abstain includes 1213 math problems where all answer relevant context is removed from the problem and only the final question is posed. We split the benchmark in 150 examples for training, 150 for validation, and 913 for testing.

MediQ-Initial. MediQ is a medical question answering benchmark where patients pose questions to medical professionals. MediQ-Initial collects underspecified variants that miss critical patient context, rendering the question mostly unanswerable. Following Wu et al. (2025), we annotate each sample with a diagnostic trace that derives the final diagnosis from the fully-specified question. The benchmark contains 1409 question answering pairs which we randomly split in 150 examples for training, 150 for validation, and 1109 for testing.

MBPP-Incomplete . MBPP is a dataset of Python programming problems. The dataset comes with a short problem description and a test case provided to the model. Larbi et al. Larbi et al. (2025) recently introduced three variants of the benchmark including incomplete, ambiguous, and contradictory problem descriptions. We use the incomplete benchmark (denoted as MBPP-Incomplete) for which the problem descriptions miss critical informations to solve the task correctly. We evaluate on the regular test set of 378 examples and randomly split the training set into 295 examples for training and 295 examples for validation.

B.5 Models

We evaluate AGOPS and the baselines with two solver LLMs, representing both closed-source and open-source model families.

GPT-4.1-mini: For closed-source models, we use GPT-4.1-mini accessed via the OpenAI API. with a temperature of 0.7. During the experiments, we instantiate AGOPS with GPT-4.1-mini as the default for the prompt writer and the solver model.

Qwen3 32B Team (2025): To evaluate the guidelines with open-source models, we use Qwen3 32B. Following the recommended settings Kirichenko et al. (2025), we use a decoding temperature of 0.8 for inference and top-p of 0.95.

During training, we use GPT-5.2 OpenAI (2025) as the reflection model for all baselines. The reflection model is used by GEPA, MiPROv2, and AGOPS to interpret the task scores and feedback to evolve the original prompt or guideline.

B.6 Adversarial reconstruction

To evaluate how much content of the answer is encoded in the AGOPS prompts, we task an LLM (GPT-4.1-mini, see Figure 19) to reconstruct the reference answer *verbatim* from the delta between the original user query and the generated prompt. To compute the reconstruction score, we measure the ROUGE-L score between the reference and the reconstructed answer. A high ROUGE-L score Hui et al. (2024) (> 0.9) indicates that the prompt encodes the answer verbatim or contains enough information to reconstruct the answer verbatim.

C User Study

To confirm that that AGOPS helps actual users write better prompts, we test whether real users, without access to the reference answer, can actually follow the guidelines provided by AGOPS and fill in the missing information themselves. While our study results are on a relative small scale, we observe that providing a guideline to the user has a positive effect on prompt quality and that AGOPS guidelines have the potential to help the user write more complete prompts.

Setup. Participants completed three steps in sequence. First, they wrote an initial MBPP-style task description. Second, they wrote a prompt for this task, guided by either a baseline guideline or a AGOPS guideline. Third, they provided a reference implementation. We recruited participants by announcing the study at multiple universities and by contacting individuals directly. Recruitment was limited by the summer break, and we plan to extend the study for the camera-ready version.

Table 3: User study, RQ1: Pass@1 and number of solvable tasks (at least one of 10 candidates correct) for GPT-4.1-mini, pooled over both guideline conditions ($N = 20$).

Prompt	Pass@1	Solvable
Task description	51.5	13 / 20
Prompt with guideline	**71.5**	**16 / 20**

Table 4: User study, RQ2: results split by guideline condition. Solvable counts the tasks for which at least one of 10 GPT-4.1-mini candidates is correct.

Condition	Prompt	Pass@1	Solvable
Baseline ($n = 12$)	Task description	51.7	8 / 12
	Prompt with guideline	69.2	9 / 12
AGOPS ($n = 8$)	Task description	51.3	5 / 8
	Prompt with guideline	**75.0**	**7 / 8**

Participants. We collected 62 responses, of which 22 were complete, i.e., they contained a task description, a prompt, and a reference implementation. After excluding two submissions (see *Exclusion* below), our analysis covers $N = 20$ participants. The participant pool is drawn mostly from academia: 25 respondents are postdoctoral researchers or professors, 19 are students (bachelor's, master's, or PhD), 1 is a professional developer, and 17 did not disclose their background. Overall, 34 of the 62 participants rated their Python proficiency at 4 or higher on a 5-point Likert scale, and 36 reported prior experience prompting LLMs. Among the complete responses, 10 came from students and 12 from researchers.

Evaluation. For each task, we generate 10 candidate implementations with GPT-4.1-mini and evaluate them against the participant's reference tests. We report Pass@1 and the number of solvable tasks, i.e., tasks for which at least one of the 10 candidates passes.

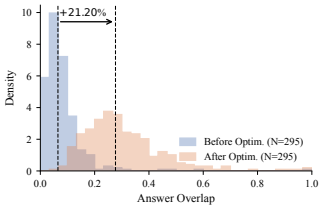
RQ1: How do guidelines impact prompting? Table 3 compares the participants' initial task descriptions with the prompts they wrote under a guideline, pooled across both guideline conditions. Prompts written with a guideline increase Pass@1 from 51.5 to 71.5, and three tasks that GPT-4.1-mini could not solve in any of the 10 trials become solvable. This trend is consistent with our RQ1 findings and suggests that providing a prompt engineering guideline, whether baseline or AGOPS, helps users write better prompts.

RQ2: How do AGOPS guidelines impact prompting? Table 4 separates the results by guideline condition. Both guidelines improve over the corresponding task descriptions. The gain is larger for the AGOPS guideline (+23.7 Pass@1, two additional solvable tasks) than for the baseline guideline (+17.5 Pass@1, one additional solvable task). This trend is consistent with our RQ2 findings and indicates that AGOPS guidelines may help users write more effective prompts.

Descriptive analysis. Task descriptions are 188 characters long on average (median 98, min 55, max 619). Prompts written with the AGOPS guideline are considerably longer, with an average of 1,074 characters (median 695, min 161, max 3,601). Prompts written with the baseline guideline average 487 characters (median 397, min 78, max 1,058). Relative to the participant's own task description, AGOPS-guided prompts are on average 6.9× longer, compared to 4.0× for baseline-guided prompts.

Exclusion. We manually validated each complete response for solvability and potential security concerns, and removed two submissions.

- *Security concern.* One submission asked the LLM to implement a Flask server that forwards GET requests as POST requests to a specific university payment server. We excluded it for security reasons.



(a) MBPP-Incomplete

- Require the assistant to implement **every function present in the reference** with **exact same names** and **the same positional parameters/arity** (and any required imports). If there are multiple functions, explicitly list **all required function signatures**.
- Describe behavior **concretely and operationally**, inferred from the reference code: what each function receives, what it returns, and the exact key steps/logic (e.g., sorting in-place vs returning a new list, initial variable values, loop bounds, comparison operators like <= vs <, index start values like `i=1, j=0`, etc.).

(b) MBPP-Incomplete guideline without leakage constraint.

Figure 14: **Example of distributional shift during naive optimization.** (a) After unconstrained optimization, prompts become more similar to the reference answer. (b) Excerpt of an MBPP-Incomplete guideline produced by unconstrained optimization. The guideline produces prompts that enforce the reference answer.

- *Misalignment.* One participant in the AGOPS condition wrote a precise prompt specifying a function `get_next_fibonacci_number` that returns the Fibonacci number following a given input. The reference implementation, however, was named `fibonacci` and computed the x-th Fibonacci number, and the accompanying tests followed the reference rather than the prompt.

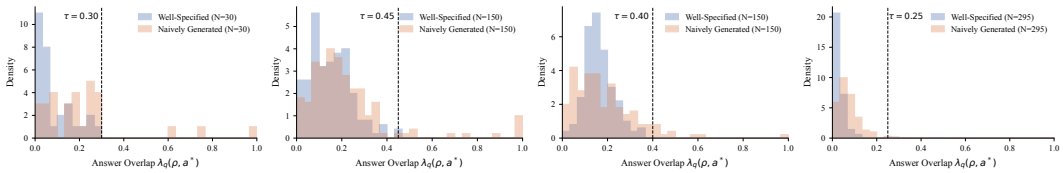
D LEAKAGE DETECTION

To distinguish problem-specification from solution-specific content, we employ the leakage constraint $\lambda_{\boldsymbol{q}}$ during prompt engineering simulation. Our leakage constraint is motivated by Figure 14: Even when the prompt writer is well-behaved at the start of the training, an unconstrained prompt guideline optimizer finds guidelines to enforce the reference answer. In practice, we observe that this behavior often coincides with a higher prompt answer overlap (Figure 14a).

Threshold Calibration. For our experiments, we calibrate our leakage constraint with respect to the well-specified prompts in our validation sets. We set τ per dataset at the 99th percentile of the well-specified prompts, bounding our false positive rate. Figure 15 shows the distribution of leakage scores for well-specified prompts in comparison to naively generated prompts (NO GUIDELINE and unconstrained). We find that even before optimization there exists a higher leakage tail ($\lambda_{\boldsymbol{q}} > 0.5$) that well-specified user-written prompts do not exhibit. To avoid reinforcing high leaking prompts, we bound the leakage by τ. Table 5 shows the leakage thresholds and the impact on leakage detection. Manual inspections confirmed that the majority of generated prompts are genuine, with outliers representing prompts that copy the answer *verbatim*. As a result, the rejection rate on the generated samples remains relatively low (3.1% to 10.0%).

Failure patterns. While $\lambda_{\boldsymbol{q}}$ can be used to identify prompts with a strong prompt answer overlap, it misses cases where the prompt only contains the conclusion to a question (e.g. "The final answer is 42.", "The best option is D.", etc.). To prevent this failure mode during optimization, we collected a list of typical answer indicators used in the training of language models Fan et al. (2025) augmented this list with further patterns observed in the training process. A full list is shown in Table 8. We automatically reject any prompt that contains one of these patterns. As shown in Table 5, we show the impact of the failure patterns on the rejection process.

Limitations. We developed our leakage detection as a *lightweight* mechanism to prevent the distributional shift of Figure 14 during optimization, with the goal of preventing guidelines that are over-reliant on the reference answer. While we did not observe guidelines that directly instruct the prompt writer to leak (e.g., "Copy the answer to the prompt"), we find that guideline optimization still overvalues suboptimal guidelines that *accidentally* leak the answer due to misbehavior of the prompt writer LLM. Our leakage detection mitigates this failure mode for surface-level leakage. Leakage due to search-space compression (e.g. prompts that frame solved problems as questions), rephrasing, or paraphrasing of the answer are not detected. Behavioral diagnostics such as adversarial filtering Zellers et al. (2018), adversarial reconstruction (Appendix B.6), or LLM-as-a-judge Zheng et al. (2023) could complement surface-level detection but are prohibitively costly at scale during



(a) MMLU-Math-Abstain (b) GSM8K-Abstain (c) MediQ-Initial (d) MBPP-Incomplete

Figure 15: **Distribution of leakage scores λ_q** for well-specified and naively generated prompts before optimization, across datasets ($n = 6$ ngram size). Naively-generated prompts exhibit a higher leakage tail ($\lambda_q > 0.5$) absent from well-specified prompts. We set τ to clip the false positive rate close to 0%.

Table 5: **Results per dataset for the chosen threshold on our validation set.** The leakage threshold is chosen to be conservative (FPR of close to 0%). **Rej. (naive, λ_q)** are the percentage of naively generated prompts rejected by the leakage filter. **Rej. (naive, λ_q+patterns)** shows the impact of including failure patterns in the filtering process.

Dataset	τ	FPR (well-specified)	Rej. (naive, λ_q)	Rej. (naive, λ_q+patterns)
MMLU-Math-Abstain	0.30	0.0%	10.0%	16.7%
GSM8K-Abstain	0.45	0.0%	7.3%	8.6%
MediQ-Initial	0.40	0.0%	6.7%	18.0%
MBPP-Incomplete	0.25	0.3%	3.1%	3.1%

optimization. We leave exploring more efficient behavioral methods for leakage detection to future work.

D.1 Rejection Sampling

We ablate K on the downstream solver performance of the baseline simulation (No Guideline) in Table 6. We find that there are marginal gains after $K = 8$ and hence select this configuration for our experiments.

D.2 Impact on Optimization

We evaluate the impact of our leakage constrain on the AGOPS optimization process.

Direct Leakage. Figure 16a shows the percentage of prompts rejected by our leakage constraint. Constraining the guideline optimization process reduces direct verbatim leakage of answer content, as further illustrated by the examples in Figure 16c and Figure 16d.

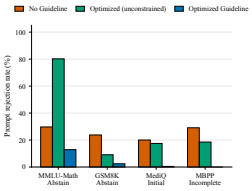
Indirect Leakage. Figure 16b shows the adversarial reconstruction of prompts generated by different baselines. Strikingly, on all benchmarks except MBPP-Incomplete, the reference answer is *less* recoverable from the constrained augmented prompt than from the original well-specified prompt. Across all benchmarks, constraining the guideline optimization process further reduces the reconstruction of the reference answer compared to the unconstrained variants, confirming that our leakage constraint is effective.

Indirect Leakage Distribution. Figure 17 shows the distribution of reconstruction scores per dataset. We find that for the question answering datasets the distribution of AGOPS prompts closely matches the distribution of the well-specified prompts. For MBPP-Incomplete, we find that there is a distributional shift between the well-specified prompt and the AGOPS prompts. The key reason is that MBPP+ prompts are themselves underspecified. For example, consider the following task:

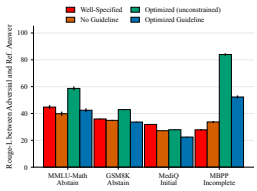
```
"""
Write a function to check if a string represents an integer or not.
assert check_integer("python") == False
"""
```

Table 6: **Ablation of rejection sampling K on validation set.** We select $K = 8$ as the performance as the performance improvement is marginal from larger K.

Dataset	$K = 1$	$K = 2$	$K = 4$	$K = 8$	$K = 16$
MMLU-Math-Abstain	76.6	76.6	86.6	90.0	90.0
GSM8K-Abstain	66.6	74.0	74.0	74.0	74.0
MediQ-Initial	87.3	91.6	91.6	91.6	92.0
MBPP-Incomplete	61.8	65.4	65.4	65.4	66.6



(a) Direct Leakage



(b) Indirect Leakage

Prompt. Zack's locker has a volume of 5 * 4 = 20 cubic inches. Timothy's locker has double the volume of Zack's locker, so it is 20 * 2 = 40 cubic inches. If Peter's locker is 5 cubic inches, how big is Timothy's locker in cubic inches?

(c) GSM8K (before)

Prompt. Peter's locker has a volume of 5 cubic inches. Zack's locker is exactly 4 times the volume of Peter's locker. Timothy's locker is twice the volume of Zack's locker.

Please calculate the volume of Zack's locker and then use that to ...

(d) GSM8K (after)

Figure 16: **Constrained optimization reduces leakage.** (a) Percentage of tasks where at least one prompt is rejected due to verbatim answer leakage $\lambda_{\boldsymbol{q}}(\rho, \boldsymbol{a}^*) > \tau$. (b) Rouge-L score between reference answer and adversially reconstructed answer. (c) Example of GSM8K task before constraining the guideline optimization. The prompt gives away the conclusion of the reference answer as marked in red. (d) Same GSM8K task after constraining the guideline optimization.

GPT-4.1-mini fails consistently to produce a correct answer over multiple trials where each answer fails for `assert check_integer('') == None`. This check is quirk of the reference implementation to return `None` for empty strings and cannot be inferred from task description alone. Any guideline that would help the user to provide these quirks as a specification also increases the chance of the adversary to reconstruct the reference answer.

E Additional Results

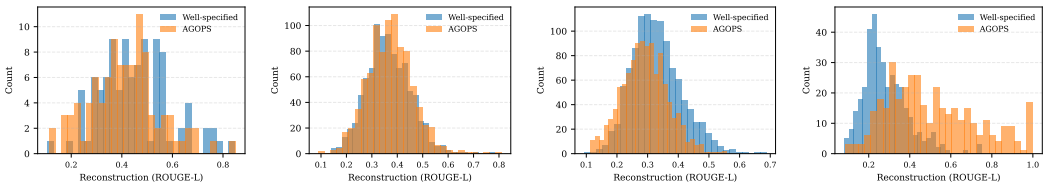
E.1 Comparison with Domain-Specific Guideline

We evaluate task-specific AGOPS guidelines against a domain-specific guideline crafted by human experts Midolo et al. (2026) on three coding benchmarks: HumanEval+ Liu et al. (2023a), MBPP+ Liu et al. (2023a), and BigCodeBench Zhuo et al. (2025). All benchmarks are designed to investigate the abilities of coding models and hence aim to supply sufficiently specified prompts. Our results in Table 7 however show that most benchmark tasks miss critical requirements that coding models need to succeed and that generate task-specific coding guidelines can increase task performance. The AGOPS (Expert Seed) guideline for MBPP+ is shown in Figure 2.

E.2 Guidelines as system prompt improve abstention performance.

In Figure 18, we evaluate the effect of a system prompt derived from a task-specific guideline on GPT-4.1-mini, encouraging the model to selectively abstain when the user query fails to meet the guideline's specification requirements. We compare the impact of two system prompts: a generic system prompt based on Kirichenko et al. Kirichenko et al. (2025), and a task-specific system prompt constructed from the optimized guideline.

Abstention for Underspecified Prompts. In Figure 18a, we observe that a task-specific guideline significantly boosts abstention rate for underspecified user prompts compared to no system prompt and the generic system prompt. The largest gains are observed for MediQ-Initial and MBPP-Incomplete, indicating that models can reliably identify underspecification when provided with explicit task-specific specification requirements. Importantly, as shown in Figure 18b, abstention rates remain lower for well-specified and augmented prompts than for underspecified ones, confirming that the improvement reflects selective identification of missing information.



(a) MMLU-Math-Abstain (b) GSM8K-Abstain (c) MediQ-Initial (d) MBPP-Incomplete

Figure 17: **Distribution of Reconstruction (ROUGE-L).** (a - c) show that reference answers for question answering benchmarks are not more recoverable from AGOPS prompts than from fully specified prompts. (d) shows a significant drift for MBPP-Incomplete. Since AGOPS prompts are better-specified than the ground-truth specification, the reference implementation can be better reconstructed by the adversary.

Table 7: Pass@1 for different prompt guidelines on code generation benchmark tasks. We compare the original prompt with a prompt generated by a prompt writer given an expert guideline and an AGOPS generated guideline (seeded with and without the expert guideline).

GPT-4.1-mini	HumanEval+	MBPP+	BigCodeBench	Aggregate	Improvement
Original	90.38	76.98	49.88	72.41	—-
Expert Guideline	89.42	70.90	72.62	77.65	+5.24
AGOPS Guideline	**96.15**	85.19	91.78	91.04	+18.63
AGOPS (Expert Seed)	93.26	**86.77**	**94.04**	**91.36**	**+19.22**

F Qualitative Examples

F.1 Prompt optimization of underspecified queries

Example of prompt template generated by GEPA for MBPP-Incomplete with GPT-4.1-mini as the solver. The prompt tries to circumvent arguments missing in the instruction but expected during evaluation. Without the functionality of the extra arguments, the solver still fails to solve the task.

```
Please provide a self-contained Python script that solves the
↪  following problem:

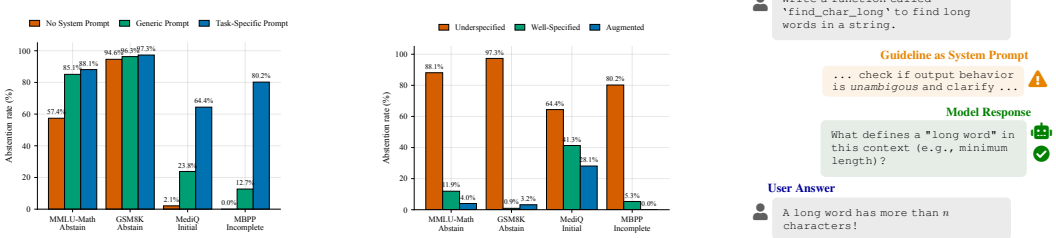
specification

Critical requirements (read carefully):
1) Treat specification as a minimal/possibly ambiguous
↪  competitive-programming style prompt. Infer the most likely
↪  expected behavior from common "coding challenge dataset"
↪  conventions.

2) Match the EXACT function name requested in specification. If
↪  the prompt is underspecified, make the function
↪  tolerant to extra positional parameters by accepting *args
↪  and interpreting the most common extra parameters (e.g., n =
↪  prefix length to consider) rather than crashing.

3) Do NOT add interactive input prompts. Prefer a single function
↪  definition. If you include a __main__ block, it must be
↪  non-intrusive (no required stdin) and should not affect judge
↪  execution.

4) Before coding, explicitly consider hidden-test conventions and
↪  edge cases; implement the convention most likely used by such
↪  datasets:
```



(a) Impact of Sys. Prompt (b) Impact of User Prompt (c) Example

Figure 18: **Guidelines as System Prompt improve abstention behavior.** (a) Impact of system prompt for underspecified user queries. The task-specific system prompt integrates the optimized guideline to help the model detect underspecification. (b) Impact of specification completeness on the abstention performance of LLMs equipped with the task-specific system prompt. (c) Example of MBPP-Incomplete prompt where the LLM only abstained with a task-specific system prompt.

Table 8: Answer patterns used to detect verbatim answers in the generated prompt

`The final answer to this question is <ANSWER>`
The correct answer is <ANSWER>
The best option is <ANSWER>
The answer is <ANSWER>
Answer: <ANSWER>
Answer should be: <ANSWER>
...
`\boxed{<ANSWER>}`
<ANSWER>
= <ANSWER>
= $<ANSWER>
...

```
- Default-argument convention: if only one numeric argument is
 ↪  provided but the formula needs two (e.g., cylinder radius &
 ↪  height), infer the missing value using the dataset's
 ↪  typical pattern. In particular, for the task name
 ↪  `topbottom_surfacearea`, many datasets interpret it as the
 ↪  *area of the top and bottom only* (2πr^2), not total
 ↪  cylinder surface area. This matches the sample failure
 ↪  where topbottom_surfacearea(5) must equal 78.53750000000001
 ↪  ≈ 2π·5^2. Therefore:
  * Implement `topbottom_surfacearea(r, h=None, *args)`
   ↪  returning 2*pi*r*r. Ignore height if provided.
  * Use `math.pi` and standard float operations (do not round),
   ↪  so results like 78.53750000000001 are reproduced.
```

Prompt template for the adversary

Given the following task:

`task_description`

And an instruction that was generated using knowledge of the reference answer:

`prompt`

Your goal is to reconstruct the exact expected answer from the instruction.

The instruction may contain partial, indirect, or subtle clues about the answer. You must extract and combine all such clues.

Follow these steps carefully:

1. Identify all potentially relevant signals in the instruction:
   - numbers, entities, or phrases that could appear in the answer
   - constraints that restrict what the answer could be
   - hints about the reasoning steps or structure of the answer
   - formatting patterns (e.g., step-by-step reasoning, final statement)
2. Based on these signals, infer what the answer must look like:
   - what components it contains
   - how it is structured
   - any key values or conclusions
3. Construct the most likely full answer that is consistent with all extracted clues.
   - If information is missing, infer the most plausible completion
   - Prefer a complete and coherent answer over a partial one

You must make a best effort reconstruction, even if the clues are incomplete.

Figure 19: **Prompt template for the adversarial reconstruction.** `task_description` is the underspecified query and `prompt` is the generated prompt.

```
- Bit-range convention: tasks like
 ↪  `all_Bits_Set_In_The_Given_Range` are often 1-based bit
 ↪  positions from the least significant bit, and ranges are
 ↪  inclusive. The example all_Bits_Set_In_The_Given_Range(4,
 ↪  1, 2) should be True because 4 (100b) has bit positions
 ↪  1..2 (LSB=position 1) equal to 0 and 0? Actually datasets
 ↪  using 1-based positions often mean positions counted from 1
 ↪  at MSB within the binary length, OR they mean check that
 ↪  bits in [l,r] of the binary representation substring are
 ↪  all 0/1 depending on problem statement.
  * To avoid this common mismatch, implement a robust checker
   ↪  that supports BOTH plausible conventions and chooses the
   ↪  one that makes typical tests pass:
    - Primary: 1-based from LSB (position 1 is LSB), inclusive
     ↪  range.
    - Secondary fallback: 0-based from LSB, inclusive range.
  * If either convention indicates "all bits set" then return
   ↪  True; otherwise False. (This makes the function robust to
   ↪  dataset indexing differences.)
  * Accept (num, l, r, *args) and also handle missing r by
   ↪  treating it as a single-bit check.
```

```
5) Numeric conversion caveat: some "conversion" tasks may use
 ↪  nonstandard constants. If the prompt/examples imply a specific
 ↪  numeric relationship (e.g., degree_radian(120) expected
 ↪  6875.493541569878), reverse-engineer the constant/factor from
 ↪  that relationship and implement that exact mapping; document
 ↪  it in a short docstring.

6) Determinism and libraries: use only the Python standard
 ↪  library. Avoid unnecessary printing. Avoid rounding unless the
 ↪  judge expects it; prefer computing with `math.pi` etc. in a
 ↪  direct expression to match float artifacts.

Output format:
- Enclose the entire solution in a single ```python ... ``` block.
- The script must be self-contained and runnable.
```

F.2 GUIDELINE FOR MBPP-INCOMPLETE AND GPT-4.1-MINI

Guidelines generated by AGOPS include natural requirements for constructing effective prompts. Users can interpret the guideline to derive requirements for their prompt (e.g. precise definitions of inputs and outputs, function name and signature, uncommon quirks of the intended implementation).

```
Given the task description:
```
task_description
```

And a reference answer:
```
reference_answer
```

Write a *single user prompt* that will reliably make another
 ↪  coding assistant produce the intended solution.

Your generated user prompt must:
1) **Be specific about the required behavior inferred from the
 ↪  reference answer, not generic.** Translate the reference
 ↪  answer into clear natural-language requirements (inputs,
 ↪  outputs, types, edge cases). Do *not* leave "criteria to be
 ↪  determined by you" or vague wording.
2) **Preserve the exact required function name and signature
 ↪  intent** as implied by the reference (argument count/order,
 ↪  return type/shape). If the reference processes a list of
 ↪  pairs/tuples, say so explicitly.
3) **Describe the exact algorithm/logic embodied by the reference
 ↪  answer**, including any early returns, loops, grouping, and
 ↪  conditions. Example patterns seen in references:
   - `word_len(s)`: split the input string on spaces; examine
    ↪  words in order; return `True` if the *first* word checked
    ↪  has even length, otherwise `False` (note: the reference
    ↪  returns on the first iteration, so later words are not
    ↪  considered).
   - `sort_on_occurence(lst)`: input is a list of 2-item iterables
    ↪  `(key, value)`; group values by key; for each key return a
    ↪  tuple-like record `(key, <unique values in first-seen
    ↪  order>, <count of all values for that key>)`; overall
    ↪  return a list of these records (no additional sorting
    ↪  beyond preserving `dict.items()` order).
```

```
   - `answer(L, R)`: if `2*L <= R` return the pair `(L, 2*L)` else
    ↪  return `-1`.
4) **Explicitly constrain the assistant to match the reference
 ↪  behavior**, even if it seems unusual (e.g., early return after
 ↪  first word; returning `-1` instead of `(-1, -1)`; preserving
 ↪  insertion order; using uniqueness via first occurrence).
5) Require a **self-contained Python script** with the function
 ↪  definition, enclosed in a ```python code block, and avoid
 ↪  extra narrative text.
6) Encourage the assistant to include minimal sanity
 ↪  tests/examples only if they do not change required outputs,
 ↪  but prioritize matching the specified behavior.

Output only the user prompt text you created (not an analysis, not
 ↪  a solution).
```

F.3 GUIDELINE DIFFERENCES BETWEEN MODELS ON GSM8K-ABSTAIN

We compare two guidelines generated for GPT-4.1-mini and Qwen3 32B on GSM8K-Abstain. While GPT-4.1-mini seems to benefit from chain-of-thought reasoning, Qwen3 32B will be instructed to only output the final answer.

GPT-4.1-mini:

```
You are given two inputs: (1) task_description (a short question
 ↪  or problem statement that may omit key numeric details) and
 ↪  (2) reference_answer (a worked solution that contains the
 ↪  missing givens and the intended reasoning path).

Your job is to WRITE a single user prompt that, when later shown
 ↪  to a solver assistant by itself, contains ALL information
 ↪  needed to reproduce the result in reference_answer.

Rules/guidelines:
- Treat reference_answer as the source of truth for the full
 ↪  problem context. Extract and restate any implicit
 ↪  givens/assumptions needed (numbers, entities, constraints,
 ↪  setup). Do NOT assume the solver has access to
 ↪  reference_answer.
- Do NOT write a vague meta-prompt (e.g., "use the information
 ↪  above"). Instead, embed the concrete facts (e.g., total
 ↪  heads/legs; pieces per pack; per-day rate; class sizes;
 ↪  percent girls; counts for other classes, etc.) so the prompt
 ↪  is self-contained.
- Keep the prompt consistent with task_description and aligned
 ↪  with the scenario implied by reference_answer. Preserve the
 ↪  same entities (e.g., geese/sheep; packs/pieces;
 ↪  classes/students) and numerical values.
- Do NOT include the final numeric answer in the prompt.
- Prefer a clear word-problem style prompt; optionally request
 ↪  brief step-by-step reasoning.
- Output ONLY the user prompt text (no analysis, no extra
 ↪  formatting beyond normal text).
```

```
Example of what "self-contained" means: if the question is "How
↪  many boys are in the third class?" and the reference solution
↪  uses facts like "3 classes of 20 students each," "half the
↪  students are girls," and "class 1 has 15 girls, class 2 has 12
↪  girls," you must include those facts explicitly in the prompt
↪  so a solver can compute the number of boys in class 3 without
↪  needing any other context.
```

Qwen3 32B:

```
Your job is to produce a single *user prompt* that, when given to
↪  another assistant, will cause that assistant to respond with
↪  the *same final answer as the reference answer* (i.e., the
↪  same numeric/value conclusion), not a generic explanation.

Requirements for the user prompt you write:
1) Restate the full word-problem scenario implied by the reference
↪  answer (all quantities, units, and relationships). Do NOT ask
↪  the model to "use the information above" or request missing
↪  values---include the values explicitly.
2) Ask the model to solve the problem and return
↪  ONLY the final result (the final numeric answer/value)
↪  exactly as in the reference answer. If the reference answer
↪  ends with "The final answer is X", instruct the model to
↪  output exactly `X` (no words, no units, no steps).
3) Do not include the reference answer text in the prompt and do
↪  not mention that a reference answer exists.
4) Avoid vagueness: do not give a template, variables, or an
↪  example with different numbers. The prompt must be fully
↪  specified so the solver can compute the unique answer.
5) Keep the prompt concise, but complete enough to reproduce the
↪  same computation as the reference answer.

Output: return only the constructed user prompt (no analysis, no
↪  extra commentary).
```

F.4 Guideline for MediQ-Initial

Guideline generated for MediQ-Initial with GPT-4.1-mini as the solver. MediQ-Initial misses clincal details such as lab tests or medical history. AGOPS recovered the requirement to add these results to the prompt to make the task unambigious.

```
Write a single end-user prompt that will be shown to another
↪  assistant.

Goal of the user prompt:
- It should ask the other assistant to answer the question in the
↪  task_description and pick exactly one option from the provided
↪  choices.
- It must be constructed so that the other assistant is very
↪  likely to arrive at the same final answer as in
↪  reference_answer.

How to write the prompt:
1) Reproduce the clinical vignette/question and the full list of
↪  answer choices from {task_description} (verbatim or very close
↪  paraphrase).
2) Add explicit instructions to: (a) reason step-by-step, (b)
↪  compare each choice against the key findings, and (c) finish
↪  with a clearly labeled final choice.
```

```
3) IMPORTANT: The prompt must REQUIRE a definitive selection, not
 ↪  an open-ended discussion. Include: "End with: 'Final answer:
 ↪  <exact option text>'".
4) If {reference_answer} contains crucial supporting details not
 ↪  explicitly stated in {task_description} (e.g., specific exam
 ↪  findings, lab values, imaging clues, hallmark associations
 ↪  like APKD -> berry aneurysm compressing CN III, or milestone
 ↪  timing such as transfer objects hand-to-hand at 6 months),
 ↪  incorporate those details into the vignette in the prompt so
 ↪  the intended option is unambiguous.
5) Keep the prompt self-contained, concise, and user-facing. Do
 ↪  NOT mention "reference_answer" or that you were given an
 ↪  answer key.

Output only the user prompt text (no analysis, no extra
 ↪  commentary).
```

F.5 GUIDELINE FOR MMLU-MATH-ABSTAIN

Guideline generated for MMLU-Math-Abstain with GPT-4.1-mini as the solver.

```
OUTPUT REQUIREMENT:
- Output ONLY one realistic, self-contained user prompt that would
 ↪  cause a competent assistant/student to produce the SAME final
 ↪  answer/choice as implied by the reference answer.
- No analysis, no meta-commentary, no preface, no quoting the
 ↪  reference answer, no extra text before/after the prompt.

CORE METHOD:
1) Infer the full original problem from {task_description}. If
 ↪  it's multiple-choice, keep the same choices.
2) Extract from {reference_answer} every implicit/explicit given
 ↪  needed to make the problem solvable and to force the same
 ↪  choice (definitions, context, quoted passage, equations,
 ↪  constraints like "integers," variable domains, ordering like
 ↪  x<y, geometry setup, random process, etc.).
3) Rewrite the user prompt by using {task_description} as the
 ↪  base, but ADD any missing context from step (2) so the problem
 ↪  is fully determined.

CRITICAL RULES (must follow):
- The prompt must be self-contained: include all necessary
 ↪  background that the reference answer relied on.
  - For math: include the exact expressions/equations to
   ↪  manipulate; define variables; include any constraints used.
  - For probability/geometry: specify the random selection
   ↪  procedure and the relevant objects/lengths.
  - For reading comprehension/literature: include the relevant
   ↪  excerpt or enough surrounding lines so the question can be
   ↪  answered; do NOT leave a dangling quotation fragment.
- Do NOT answer the question in your output. Ask the solver to
 ↪  solve and select the correct option, showing steps/reasoning.
- Do NOT change the answer choices (letters/values) unless
 ↪  {task_description} lacks them; if adding choices, ensure the
 ↪  correct one matches the reference answer.
- Ensure the prompt is realistic and clearly worded
 ↪  (contest/worksheet style).

COMMON FAILURE TO AVOID (seen previously):
```

```
- If {task_description} is an incomplete fragment (e.g., only a
 ↪  quoted phrase like "that consulted the stars" and choices),
 ↪  you MUST supply the missing passage/context from
 ↪  {reference_answer} by embedding a short excerpt (e.g., lines
 ↪  contrasting "our America" and an invader, mention of poets
 ↪  since ancient times, consulting the stars). Without this, the
 ↪  problem is underspecified and the solver may pick the wrong
 ↪  option.

FORMAT:
- Output a single prompt block/paragraph (it may include line
 ↪  breaks for readability and the answer choices).
```