

Ask, Then Think: Does Socratic Prompting Help LLMs on Code Tasks?

Antonio Della Porta

`adellaporta@unisa.it`

University of Salerno: Università degli Studi di Salerno <https://orcid.org/0000-0003-1860-8404>

Jonan Richards

`jonan.richards@ru.nl`

Radboud University Nijmegen

Lucageneroso Cammarota

University of Salerno

Stefano Lambiase

Aalborg University Copenhagen

Fabio Palomba

University of Salerno

Mairieli Wessel

Radboud University Nijmegen

Research Article

Keywords: Large Language Models, Prompt Patterns, Cognitive Theories

Posted Date: October 14th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11361240/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Ask, Then Think: Does Socratic Prompting Help LLMs on Code Tasks?

Antonio Della Porta · Jonan Richards ·
Lucageneroso Cammarota · Stefano
Lambiase · Fabio Palomba · Mairieli
Wessel

the date of receipt and acceptance should be inserted later

Abstract Large Language Models (LLMs) are widely used for software engineering tasks such as code generation, repair, and explanation, yet they remain error-prone when instructions are ambiguous, relevant information is missing, or initial assumptions are incorrect. Existing prompting strategies can structure the reasoning process, but they do not explicitly encourage models to question their own assumptions and intermediate conclusions. Motivated by the possibility of operationalizing theories of human reasoning to steer LLM behavior, we introduce MIND, a prompting framework that translates principles of classical Socratic reasoning into a structured self-interrogation process

This work used the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-19214. We thank SURF (www.surf.nl) for the support in using the National Supercomputer Snellius.

A. Della Porta;
University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano (SA), Italy
E-mail: adellaporta@unisa.it

J. Richards;
Radboud University, Houtlaan 4, 6525 XZ Nijmegen, Netherlands
E-mail: jonan.richards@ru.nl

L. Cammarota;
University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano (SA), Italy
E-mail: l.cammarota8@studenti.unisa.it

S. Lambiase
Aalborg University in Copenhagen, A. C. Meyers Vænge 15, 2450 Copenhagen, Denmark
E-mail: stla@cs.aau.dk

F. Palomba
University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano (SA), Italy
E-mail: fpalomba@unisa.it

M. Wessel;
Radboud University, Houtlaan 4, 6525 XZ Nijmegen, Netherlands
E-mail: mairieli.wessel@ru.nl

for LLMs. We evaluated MIND across 12 LLMs from four model families and three size categories on four benchmarks covering code generation, repair, explanation, and efficiency, comparing it with Chain-of-Thought, Self-Ask, Self-Refine, and ReAct. We further conducted an ablation study to disentangle the contribution of the individual Socratic reasoning components. Our results show that MIND does not outperform established prompting strategies. It falls below Chain-of-Thought, which achieves the highest average performance, as well as below Self-Refine and, in most settings, Self-Ask, while introducing substantial inference-time overhead. MIND is ahead of ReAct overall, mostly because of its advantage on code efficiency, but not on every benchmark, and its gap to the strongest baselines narrows only for the largest models. The ablation analysis shows that most Socratic components do not pay off, as removing them often improves performance and only premise seeking seems to impact positively more often than the others. Overall, our findings indicate that transferring a theory of human reasoning into a prompt as a whole does not improve LLM performance on code tasks, and that any benefit depends on selecting the few mechanisms that actually support the model’s reasoning.

Keywords Large Language Models; Prompt Patterns; Cognitive Theories

1 Introduction

Large Language Models (LLMs) have become widely used in software engineering for problem solving [67], code generation [45], and supporting reasoning tasks [40]. As their use spreads, expectations rise, and users increasingly want systems that show clear understanding, cope with unclear instructions, and avoid errors caused by false assumptions [24]. Yet current models still show clear limitations when instructions are vague or when essential context is missing [73]. A substantial amount of work has aimed to improve this through prompting methods. Techniques like Chain-of-Thought [70] and ReAct [75] show that asking a model to spell out its reasoning can help, but these methods mostly guide the model to think in a straight, predictable way. The nature of these prompting strategies leads the model to define and perform steps, but not to question whether those steps make sense in the first place or if they are going in the correct direction to answer the user’s request [54]. As a result, the model can produce detailed step planning that can potentially miss the point when the original problem was misunderstood.

At the same time, whether the internal processes underlying LLM outputs can meaningfully be characterized as reasoning in the human sense is still an open question. Rather than assuming such an equivalence, a potentially fruitful direction is to ask whether theories developed to explain and improve human cognitive processes can be operationalized to steer the observable behavior of LLMs. This possibility is particularly intriguing in light of recent studies showing that LLMs can exhibit behavioral patterns associated with human-like traits, including emotional intelligence [59], self-reflection [57], and metacognitive abilities [65]. These findings do not imply that LLMs possess

human cognitive processes; however, they raise the question of whether principles originally developed for human cognition can provide useful structures for eliciting more deliberate and self-critical model behavior.

From this perspective, human problem solving offers an interesting starting point. When approaching a complex problem, people can formulate an initial plan and subsequently challenge it by searching for flaws, questioning assumptions, or identifying missing conditions [28]. For example, a developer might sketch an algorithm, realize that an edge case invalidates part of the logic, and revise the plan before implementing it. This leads to the central intuition behind our work: *Can a theory designed to stimulate this form of critical inquiry in humans be translated into a prompting strategy that similarly encourages LLMs to question and revise their own reasoning?*

To investigate this question, we introduce MAIEUTIC INFERENCE AND DISCOVERY (*MIND*), a prompting framework that operationalizes principles of classical Socratic reasoning [2][63][27] that is based on the Socratic theory of thinking.¹ Instead of simply asking the model to think step by step, *MIND* leads the model through short, focused self-questions that help it clarify the task, identify missing information, break the problem into parts, and check whether the final answer makes sense. The goal is to bring a small amount of structured self-interrogation into the reasoning process. While existing prompting methods encourage models to show their chain of thought, *MIND* pushes them to examine it by spotting vagueness, checking assumptions, identifying constraints, and challenging intermediate conclusions. In this way, it adds a form of internal quality control that other prompting strategies usually do not have.

We empirically evaluated *MIND* across 12 LLMs from four model families and three size categories, using four code-related benchmarks covering code generation, repair, explanation, and efficiency: BigCodeBench [78], HumanEvalFix and HumanEvalExplain [49], and Mercury [23]. In addition, we compared *MIND* against four established prompting strategies, i.e., Chain-of-Thought, Self-Ask, Self-Refine, and ReAct, and complemented this evaluation with an ablation study to determine the contribution of the individual stages composing the Socratic process.

Our results showed that *MIND* did not outperform established prompting strategies overall: Chain-of-Thought achieved the highest average performance across all benchmarks, with *MIND* trailing it by approximately 6–9 percentage points at $k = 1$ while also introducing substantial inference-time overhead. The ablation analysis revealed that the individual Socratic mechanisms do not contribute uniformly: while some components supported performance, removing others frequently improved it. These findings suggest that directly transferring

¹ This work builds upon the Registered Report accepted at the 23rd International Conference on Mining Software Repositories (MSR 2026) [17], which presented the study design and empirical protocol underlying this investigation. Following the defined protocol we updated the LLMs to their latest versions. For time feasibility reasons, we switched the backend used to execute the models from llama.cpp to vLLM, but this change allowed us to speed up inference time and also use more precise models (FP8 instead of FP4 as originally defined).

a complete human-inspired reasoning process to LLMs is not sufficient per se; rather, its individual cognitive mechanisms need to be understood and selectively operationalized according to how they interact with model behavior. As a consequence, we addressed our motivating question *affirmatively, but with an important qualification*: human reasoning theories can provide a valuable basis for steering LLM behavior, but their effectiveness depends on how their individual mechanisms are translated into prompting strategies rather than on faithfully reproducing the complete human reasoning process. To sum up, our work provides the following contributions:

- We introduce MIND, a prompting framework that operationalizes classical Socratic reasoning as a structured self-interrogation process for LLMs;
- A large-scale empirical evaluation of *MIND*, showing that its effectiveness depends on the underlying model and task, that it does not outperform established prompting strategies overall but becomes more competitive as model size increases, and that this performance comes with a substantial inference-time overhead;
- We characterize the opportunities and limitations of operationalizing human reasoning theories to steer LLM behavior, showing that transferring a human-inspired reasoning process as a whole does not necessarily improve LLM performance, while its individual cognitive mechanisms can have substantially different effects. These findings provide empirical foundations for future research on cognitively inspired prompting strategies;
- A replication package [18] containing the implementation of *MIND*, the experimental infrastructure, prompts, and data produced during our study, enabling the replication of our results and further research on human-inspired reasoning strategies for LLMs.

2 Background and Related Works

2.1 Metacognitive Abilities of LLMs

Metacognition denotes the monitoring and control that a reasoner exerts over its own cognition [25, 50], and a growing body of work suggests that LLMs possess functional analogues of it. Beyond showing human-like content effects in reasoning [39], large models are well calibrated on many question formats and can predict whether they know the answer to a question [35], they can learn to predict their own behavior better than other models trained on that behavior [3], and researchers from Anthropic found that a model internally represents its own knowledge and uncertainty [44]. This self-knowledge can also be put to use, as LLMs can identify the skills that a problem requires and improve when prompted with exemplars of those skills [21], *Metacognitive Prompting* outperforms Chain-of-Thought on language understanding tasks by asking the model to evaluate its preliminary judgment critically and to assess its confidence [69], and reasoning models trained with reinforcement learning develop reflection and self-verification without being taught to [26].

2.2 Prompt Engineering

In software engineering, few-shot, zero-shot, and Chain-of-Thought prompting are the most frequently used techniques [29], and their effect on code has proven mixed. The prompt technique influences the correctness and quality of generated functions, although combining techniques does not necessarily help [36], and prompt patterns do not significantly change the maintainability, security, and reliability of ChatGPT-generated code [14], while they do affect its size, with Chain-of-Thought producing more concise code [16]. Beyond the choice of technique, the wording of a prompt also matters. Semantically equivalent descriptions of the same method lead GitHub Copilot to generate different code [47], the readability of developer prompts predicts the correctness and usefulness of the code returned by ChatGPT, and their structure predicts its consistency with the code eventually committed [15], recurring prompt smells signal quality issues that surface-level text metrics only partly capture [19], and syntactic elements such as constraints and conditions, together with their position in the prompt, affect how often open LLMs generate insecure code [12]. Finally, techniques designed for earlier LLMs bring smaller benefits, or even harm, on more advanced and reasoning models [64].

2.3 Structured Reasoning Prompting

Among prompt engineering techniques, those that structure the reasoning of the model are the closest to our work [54]. Chain-of-Thought, proposed by Wei et al., prompts the model to produce explicit intermediate steps before the final answer [70], also without demonstrations [37] or with a majority vote over several sampled chains [68]. Decomposition methods reduce the problem to simpler sub-problems [77], devise a plan before carrying it out [66], or, as in Self-Ask, have the model pose and answer explicit follow-up questions [55], while Tree of Thoughts searches over alternative partial solutions [74] and ReAct interleaves brief reasoning with actions such as tool calls [75]. These methods improve accuracy by encouraging a more detailed decomposition of the task, but apart from the heuristic search of Tree of Thoughts they generate their steps in a forward manner. The model explains, splits, or operationalizes its reasoning without being required to question whether its assumptions hold or whether its intermediate conclusions are valid, so that Self-Ask never examines whether its sub-questions are well posed and ReAct reasons about the next action rather than about the soundness of the current conclusion.

A second family asks the model to criticize its own output. Self-Refine, proposed by Madaan et al., generates a solution, produces feedback on it, and revises it [46], Reflexion keeps verbal reflections on failed attempts to guide the next one [60], and verification methods check a conclusion by recovering the conditions of the problem from it [71] or by answering verification questions about a draft [20]. Here the critique targets a finished output, so the refinement edits the result rather than the understanding of the problem and

the plan that produced it, and its benefit is limited when the model has no external feedback to rely on [30]. A few methods draw directly on the Socratic tradition, by checking the logical consistency of abductive explanations with an external solver [34], by adapting Socratic techniques to the design of prompts [6], or by recursively raising and answering sub-questions [56]. Together with Metacognitive Prompting [69], they show that questioning can improve the reasoning of LLMs, but they were designed for question answering and language understanding, and they interrogate the content of an answer rather than the framing of the problem and the plan that precede it.

2.4 Structured Reasoning for Code Generation

Structured reasoning has also been adapted to code generation [4]. Jiang et al. separate planning from implementation [32], Li et al. structure the chain of thought with the sequence, branch, and loop constructs of programs [41], Pan and Zhang organize the reasoning as a graph of modules [52], and ClarifyGPT asks clarifying questions when a requirement is ambiguous [48]. Most of the large gains reported for code, however, come from methods that check the reasoning against execution, such as Self-Debug [9], LDB [76], CodeT [7], and multi-stage flows like AlphaCodium and MapCoder [58,31]. When the model has to rely on its own judgment, the gains are more fragile, as self-repair is bottlenecked by the quality of the feedback that the model gives on its own code [51], self-generated tests introduce a bias that limits self-debugging [10], and Self-Debug improves far less when no unit tests are available [9], although execution-free gains have been reported for self-collaboration [22] and for iterative self-critique in refactoring [5].

Independent comparisons are also less favorable than the papers that propose these methods. The benefit of structured chains of thought depends on model capacity, and naive zero-shot reasoning can even degrade performance [33], applying Chain-of-Thought uniformly induces overthinking on simple tasks [43], and the gains of Chain-of-Thought itself concentrate on mathematical and symbolic reasoning [62]. Socratic questioning, finally, has been applied to programming mostly as a dialogue in which an expert guides the model through problems it had failed to solve [11], that is, with a questioner who knows more than the answerer.

2.5 Positioning of *MIND*

Compared to these approaches, *MIND* shifts the target of structure from the surface form of the reasoning to the checks that the model must perform on its own thinking, before, during, and after the construction of a solution. The model is first asked to expose the ambiguities and hidden assumptions of the request, to separate what it knows with certainty from what it does not, and to identify the constraints that a solution must not violate. It then decomposes the problem, searches each sub-problem for failure modes and edge

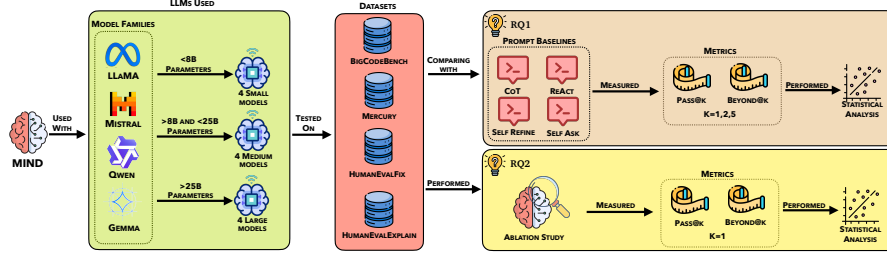


Fig. 1: Overview of the Research Method

cases, and validates the plan through an explicit gate that sends it back to the decomposition when the plan is judged invalid, before a second gate checks the solution against the original intent. *MIND* thus scaffolds both the monitoring and the control of the model’s reasoning, with the same model acting as questioner and answerer and without any execution feedback, which sets it apart from Socratic methods that rely on a more knowledgeable questioner or target question answering, and from code generation methods that either add forward structure or validate the solution by running it.

3 Maieutic INference and Discovery approach

Recent studies in the AI field [13][39][44] show that LLMs display content-dependent reasoning biases and are able to internally represent their own knowledge and uncertainty. Based on these findings on similarities between humans’ and LLMs’ thought processes, we aim to extend the concept in the SE field to further evaluate the extent to which processes defined to improve human reasoning can be applied to improve the performance of LLMs.

MIND is based on the classical *Socratic reasoning* process, a form of philosophical inquiry that examines beliefs through structured dialogue [2]. This process starts from the idea that people often hold views that seem clear to them but contain hidden assumptions or unresolved tensions. By asking focused questions, Socrates leads the interlocutor to inspect the foundations of their thinking rather than receiving answers from an external authority.

The classical Socratic process used to build the proposed approach is composed of the following three main phases [63]:

Elenchos. In this first phase, the *Elenchos* works as a careful check of the initial claim. Socrates pushes the interlocutor to say clearly what the problem is, what assumptions they are relying on, and what follows from taking those assumptions seriously. This removes vagueness and makes hidden premises visible. Once everything is stated plainly, any clash with the interlocutor’s other beliefs becomes obvious, and the initial claim often collapses. This collapse is useful because it shows which parts of the problem are solid, which are unclear, and which pieces of information or constraints are actually missing.

Aporia. When the initial claim falls apart, the interlocutor enters *Aporia*. The tensions exposed during the elenchos point directly to the parts that need to be examined one by one, and these become separate questions. The problem is broken into smaller, clearer components, each of which can be checked independently. This division is guided by the weak spots uncovered earlier, so it is neither random nor superficial. With the problem now decomposed, the interlocutor can explore each part, test where it might fail, and verify whether solving these parts genuinely solves the whole.

Maieutica. Once the key ideas are clarified and the problem is divided into stable parts, the *Maieutica* allows a complete answer to take shape. Socrates helps the interlocutor rebuild the solution by connecting the pieces in a clear and ordered way. He does not provide the answer himself; he encourages the interlocutor to follow the structure that emerged from their own reasoning. Because the earlier steps removed contradictions and validated the plan, the final idea appears as the natural outcome of the process. The conclusion is then checked against the original question to ensure it actually addresses it and does not introduce new issues.

Based on this reasoning process, we propose the MAIEUTIC INFERENCE AND DISCOVERY (*MIND*) approach, which enables language models to reason inspired by the Socratic Maieutic method. We adapt this concept for language models not as a simple, one-shot prompt, but as a structured, multi-phase dialectic that emulates the process Socrates used with his students to teach how to tackle complex questions. In Table 1 we describe the reasoning process by mapping the classical Socratic method onto the same three phases, and we keep the goals of each phase close to how they work in the original practice.

To implement the approach, we started by first building a complete prompt pipeline that encodes each phase of the method as a separate prompt. These prompts were then tested on one model per size category to confirm that the structure is followed. We checked whether the model interprets each step correctly, whether the answers stay within the intended boundaries of the phase, and if there are missing sub-steps or skipped parts of the reasoning.

4 Method

The *goal* of the methodology is to empirically evaluate the performances of the *MIND* approach compared with some baseline approaches in complex, code-related tasks, and also investigate which is the part that contributes the most to the outcomes of the *MIND* approach to improve our understanding of cognitive theories in LLMs and also to have some insights on what to improve. Following established guidelines for empirical study design [72], we now formulate two research questions that guide our investigation and help achieve the study’s objectives.

Table 1: Socratic Prompting Framework: Goals and Self-Interrogation Mechanics

Step	Goal	Questions Posed
Phase 1: Elenchos – Refutation & Definition		
1.1 Initial Refutation	Eliminate ambiguity and define exact problem scope.	Where exactly is this request vague? What hidden assumption is the user making that I must challenge or clarify to avoid failure?
1.2 Premise Seeking	Identify certain knowledge, standard tools, and undeniable truths.	What do I know for certain? What libraries or algorithms can serve as a secure base?
1.3 Aporia Discovery	Identify missing information and hard constraints.	What is the core piece of missing knowledge? What is the constraint that, if violated, invalidates everything?
Phase 2: Aporia – Division & Planning		
2.1 Decomposition	Divide the complex problem into manageable, logical parts.	Can this problem be divided into smaller parts? Is this division exhaustive?
2.2 Sub-Problem Inquiry	Stress-test sub-problems to find breaking points (edge cases).	How could this specific part fail? What input would falsify my current hypothesis for this sub-part?
2.3 Path Validation	Validate logical sequence; ensure parts sum to the whole.	If I solve all the sub-problems, is the whole definitely solved? Is the execution order logically unassailable?
Phase 3: Maieutica – Birthing the Solution		
3.1 Solution Crafting	Synthesize final solution strictly following the validated plan.	Execute the full plan that was defined. Take the sub-problems in sequence, generate the solution for each, and then create the final response.
3.2 Final Validation	Validate final solution against original intent, not just the plan.	Does this satisfy the <i>original</i> request? Have I introduced new, unnecessary contradictions?

 **RQ₁** — To what extent does the *MIND* approach improve the performance of LLMs compared to established prompting baselines across standard benchmarks?

With this question, we aim to assess whether *MIND* indeed improves performance on diverse code-related tasks and whether it can be used in real-world scenarios. To answer this research question, we quantitatively evaluated whether the *MIND* approach outperforms established reasoning-based meth-

ods such as Chain-of-Thought, Self-Refine, Self-Ask, and ReAct across different benchmarks, each targeting different programming activities. To quantify the improvement, we measured the functional correctness of the generated solutions using the task-specific tests of each benchmark and compute the *pass@k* and *Beyond@k* metric. We also measured the inference time duration in all the experiments to weight the accuracy with the time needed to get a solution.

 **RQ₂** — Which components of the *MIND* approach contribute most significantly to its overall performance?

This research question examines the individual stages that compose the *MIND* approach. Although each stage is intended to clarify the problem and strengthen the reasoning process, their actual contribution may vary. The ablation study, therefore, serves two purposes. If RQ₁ reveals weaknesses or uneven gains, the ablations help identify which steps are responsible and where improvements are needed. If RQ₁ shows consistently strong performance, the ablations clarify which components are essential, which are redundant, and whether the approach can be simplified without losing effectiveness. To do this, we evaluated ablated variants of *MIND* on the same benchmarks as RQ₁ and measure the performance changes associated with removing each component.

4.1 Data Extraction

4.1.1 LLMs Selection

To make sure that the performance of *MIND* is not tied to a specific model scale, we evaluated it across LLMs of different sizes and different architectures.

For the size, we follow a simple size-based grouping commonly used in recent multi-model studies[78]: **small** models (fewer than 8B parameters), **medium** models (8B–25B), and **large** models (more than 25B). We cap the upper end at 70B parameters to keep the experiments feasible with our available compute.

For the architectures, our goal is to assemble a representative yet manageable set, so we focus on four widely used open-weight families which together cover the main model designs found in current research: **Mistral**, **Gemma**, **LLaMA**, and **Qwen**.

To build the set of models used throughout our evaluation, we adopted the following criteria: within each size category, we select one model from each family using the most recent publicly available checkpoint. When several candidates are available (e.g. LLaMA 3.2 1B and 3B in the small cluster), we pick the largest one to better represent the family’s capabilities.

The list of selected models is reported in Table 2. Following empirical guidelines for LLM-based studies in software engineering proposed by Baltes et al. [1], all experiments include for each model used, its fingerprint information

Table 2: LLMs used in the study

Small		Medium		Large	
Name	Size	Name	Size	Name	Size
LLaMA 3.2	3B	LLaMA 3.1	8B	LLaMA 3.3	70B
Ministral 3	3B	Ministral 3	8B	Mixtral	42B
Qwen 3.5	4B	Qwen 3.5	9B	Qwen 3.6	35B
Gemma 4	4B	Gemma 4	12B	Gemma 4	31B

to ensure reproducibility and traceability. Moreover, experiments were conducted under consistent conditions, using the same hyperparameters across models: the temperature was fixed at 0.7, the top-p parameter at 0.95. These values were selected based on their adoption in similar settings on code-related tasks[42]. Moreover, we decided to also fix the maximum of amount of tokens per generation at 4096, and the context at 32,768 tokens. All the LLMs used were distributed as checkpoints on HuggingFace, using FP8 quantized variants, and were executed using the vLLM[38] backend to enable parallel executions.

4.1.2 Baseline Selection

To have a clear picture of the *MIND* approach performances, we compared it against prompting techniques that also rely on structured or self-reflective reasoning. We select well-established baselines from the literature[53] that follow this style: **Chain-of-Thought** [70], **Self-Ask** [55], **ReAct** [75], and **Self-Refine** [46]. Chain-of-Thought represents step-by-step deductive reasoning and serves as a minimal structured prompting baseline. Self-Ask decomposes a problem into explicit sub-questions, encouraging a more modular form of reasoning. ReAct integrates reasoning and action by alternating between intermediate reasoning steps and tool or environment interactions, testing how reasoning emerges through feedback. Finally, Self-Refine introduces iterative self-correction, where the model revises its own outputs through a series of feedback loops.

To minimize variability due to prompt writing, we used the prompt templates used in the papers that originally proposed these approaches and on which the original evaluation was performed. The prompts that we used are in the replication package[18]. Since the original paper of ReAct had not specified tools for our context, we gave it the ability to use the static analysers Pyflakes² and mypy³. Whenever the model writes code, their output on syntax errors, undefined names and type issues is returned to it as the next observation, without executing the code.

² <https://github.com/PyCQA/pyflakes>

³ <https://mypy-lang.org>

4.1.3 Prompts Evaluation

Because the results of our analysis depend on the prompts, we treat their design as part of the method. Before running the study, the first two authors drew 10 tasks at random from each of BigCodeBench, HumanEvalFix, HumanEvalExplain and Mercury using a fixed seed (42), and set them aside for prompt evaluation. The prompts of both the baselines and MIND were developed only on these holdout tasks, which we excluded from all reported results. The reported results therefore cover 1130 tasks on BigCodeBench, 154 on each HumanEval variant and 246 on Mercury, and no prompt was tuned on a task that later contributed to a reported score.

On the holdout tasks, the first author ran the full pipeline with one model from each size class, choosing a different family each time, namely Gemma 4 4B (small), Ministral 3 8B (medium) and Qwen 3.6 35B (large). Covering all three size classes and three families was meant to keep the prompts from being fitted to the behaviour of a single model. For each run the first author read the complete conversation and checked, step by step, whether the model followed the instructions of that step, whether its output stayed focused on the task it was given, and whether the final answer had the format expected by the benchmark. Every deviation was recorded in a log, and the second author reviewed the log before the prompts were revised.

The purpose of these runs was to inspect the outputs rather than to measure them, so no holdout score was computed or compared at any point. Once the cycle ended, all prompts were frozen and left unchanged for the rest of the study.

4.1.4 Benchmarks Selection

To conduct our experimentation, we selected four public benchmarks that aim at measuring the capabilities of models in code-related tasks, listed in Table 3. These datasets cover complementary aspects of coding tasks, including code generation, optimization, repair, and explanation, allowing a comprehensive assessment of both the productive and reflective capabilities that the *MIND* approach aims to enhance.

BigCodeBench [78] provides a large set of code generation tasks and serves as the main reference for evaluating how effectively a model can compose functional solutions from natural-language problem descriptions.

Mercury [23] focuses on code efficiency and evaluates whether models can produce code solutions that are not only functionally correct but also efficient.

HumanEvalFix and HumanEvalExplain [49] extend the original HumanEval [8] benchmark to the domains of program repair and code explanation, testing the model’s ability to identify and correct errors or to describe the purpose and behavior of a code snippet.

Table 3: Datasets used in the study

Name	Task Performed	#Problems
BigCodeBench [78]	Code Generation	1140
Mercury [23]	Code Efficiency	256
HumanEvalFix [49]	Code Repair	164
HumanEvalExplain [49]	Code Explanation	164

4.2 Data Collection

To answer the research questions, we began by collecting the performance of the MIND approach and the selected baseline approaches on the four selected benchmarks. For each benchmark, we developed an automated pipeline that performs the task and outputs all relevant data, such as the generated solutions, error codes, and reasoning traces, to an external file for transparency.

For **BigCodeBench** [78], the pipeline prompts models with the natural language problem description and make each of the selected models, with each of the selected baselines, generate the code that represents the solution described. The generated code was then tested with the official test suite for each problem that is provided by the dataset. The metric used to assess the performances in this benchmark is the $pass@k$ with $k \in \{1, 2, 5\}$, that quantifies the functional correctness of the generated code in up to 5 different generations.

For **HumanEvalFix** and **HumanEvalExplain** [49], we followed the procedures defined in the original article that proposed the benchmarks. In **HumanEvalFix**, each model was provided with a buggy implementation and the corresponding problem statement, and the task was to fix the buggy code and make it pass the test cases. The metric used to assess the performances in this benchmark is the $pass@k$ with $k \in \{1, 2, 5\}$, that evaluate whether the model was able to fix the code snippet having up to 5 different attempts. For **HumanEvalExplain**, the process was split into two separate steps. In the first step, each model was provided with a code snippet and was first asked to produce a natural language explanation of the snippet. The second step consists of taking the natural language description and then, in a separate stateless API call, the model was asked to regenerate the code from that explanation. The $pass@k$ metric evaluated whether the model was able to understand the content of the code snippet in up to 5 different attempts.

For **Mercury** [23], models were prompted to generate code solutions from natural language problem statements. The generated solutions were evaluated using the *Beyond* metric, the same that is proposed in the original article to evaluate the performances on the Mercury benchmark. The *Beyond* metric measures runtime efficiency by computing the percentile of the generated solution’s execution time within the runtime distribution of verified human-written solutions defined in the benchmark, assigning higher scores to code that is more computationally efficient. In our case, we used an adapted ver-

sion of the *Beyond* metric that we called *Beyond@k* with $k \in \{1, 2, 5\}$. Since each model produces five candidate solutions as in the other benchmark tasks, we compute *Beyond@k* by selecting, respectively, the best score among the top 1, top 2, and top 5 generated solutions.

To analyze the difference in time execution of the approaches, we timed the first execution of them using Python’s built-in `time` library. Specifically, for each LLM call we measured the delta time (in milliseconds) between the timestamp captured immediately before issuing the inference request and the timestamp captured immediately after the response is fully received.

4.3 Data Analysis

To answer **RQ₁**, the performance of the MIND approach were compared against all selected baselines using the metrics defined by each benchmark, and also took into account the different families of models, their size, and the specific task (e.g., code efficiency, code repair) that the LLM has performed. On all the problems that are in BigCodeBench, HumanEvalFix, and HumanEvalExplain, we computed *pass@k* metric with $k \in \{1, 2, 5\}$, and for the problems in the Mercury dataset we computed the *Beyond@k* metric with the same values of k .

The dependent variable of the analysis was the aforementioned *pass@k* and *Beyond@k* metrics, while for the independent variables, we grouped the observations using the type of **prompting strategy** used (e.g., MIND and ReAct), while also taking into account the **model family**, the **size** cluster it was in, and the **task** that was performed.

The hypotheses that guide the statistical analysis are the following:

- **Null Hypothesis:** There are no significant differences in the evaluation metrics based on
 - ($H1_0$) The prompting strategy used,
 - ($H2_0$) The model family,
 - ($H3_0$) The model size cluster,
 - ($H4_0$) The task type.
- **Alternative Hypothesis:** There are significant differences in the evaluation metrics based on
 - ($H1_a$) The prompting strategy used,
 - ($H2_a$) The model family,
 - ($H3_a$) The model size cluster,
 - ($H4_a$) The task type.

Given that *pass@k* and *Beyond@k* are bounded proportions in $[0, 1]$, and that we have multiple observations of the same problem, we model the analysis using **beta mixed-effects regression**. Before fitting the model, we (1) check the models’ assumptions (e.g., adequacy of the beta distribution and random-effects structure) through standard residual and convergence metrics, and (2) rescale the values to avoid boundary issues, since the outcome must

be restricted to the open interval $(0, 1)$. To this aim, we followed Smithson and Verkuilen [61] to apply a standard transformation that maps values into the open interval $(0, 1)$.

If the assumptions of the model are violated or the models fail to converge, we fall back on non-parametric alternatives for repeated measures, such as the Friedman test or aligned-rank transformation ANOVA applied to the same metrics. For statistically significant effects, we perform post-hoc pairwise comparisons with Holm–Bonferroni correction for multiple testing and report effect sizes based on contrasts of estimated marginal means on the link scale (e.g., odds ratios or standardized differences), together with confidence intervals.

To address efficiency concerns, we additionally track inference duration for every run and report it as a descriptive measure of overhead. The goal is to make the trade-off between effectiveness and efficiency explicit. Concretely, for each model family and size, we summarize the inference-time distributions of MIND and the baseline prompting strategies using quartiles, and we report (i) the relative change in inference time of MIND compared to each baseline and (ii) differences in interquartile range (IQR) to capture changes in dispersion. This comparison of inference times allows us to weigh the potential benefits in accuracy of the MIND approach against the potential efficiency losses.

To answer RQ₂, we investigated the contribution of each component of the MIND framework using a single-step ablation study. Since the approach consists of eight steps (1.1 to 3.2), we constructed ablated variants by removing each step individually, except for Step 3.1 (Solution Crafting), which cannot be disabled without preventing solution generation. Each ablated variant, therefore, removes exactly one of the remaining seven steps while keeping all others unchanged. All ablations were performed under the same conditions as the full MIND approach, using identical inputs, decoding settings, and evaluation pipelines.

Ablations were evaluated with $k = 1$, using either *pass@1* or *Beyond@1* depending on the benchmark. The impact of each component is characterized through the absolute and relative changes in performance with respect to the full MIND system. We summarized the observed trends using descriptive statistics and highlight the configurations that produce the largest changes, providing an interpretable view of how each module influences overall effectiveness.

4.4 Hardware Used

The experiments were conducted on the Dutch national supercomputer Snelius, hosted at SURF. The computational nodes used each had a NVIDIA H100 SXM5 GPU with 94 GiB of HBM2e memory, as well as 16 cores of an AMD EPYC 9334 CPU and 180 GiB of DRAM. Software versions used were Python 3.13.5 and PyTorch 2.11.0 with CUDA 12.9 support, with inference being performed on the GPU using vLLM 0.26.0.

To measure inference time, the first iteration of each experimental run for RQ_1 was executed sequentially using a single worker. The remaining iterations and the experiments for RQ_2 were executed with 64 concurrent workers, allowing vLLM to process requests concurrently and make efficient use of the allocated GPU resources.

For reproducibility and transparency, the bash scripts used to run the experiments are provided alongside this study [18]. The scripts are configured for the Slurm job system and the available hardware. With some modifications, they can also be executed in other environments.

5 Analysis of the Results

5.1 Prompt Evaluation Process

This subsection reports the outcome of the prompt evaluation described in subsubsection 4.1.3. All observations come from the holdout tasks, which take no part in any later result.

For the baselines, the first author ran the four prompts, which follow the templates of their original papers, on the holdout tasks with the same three models, checking that each baseline behaved as its original description prescribes and that its final answer was complete and had the structure needed for automatic extraction and testing.

For MIND, the initial version of the prompt template (0.1) was written directly from the Socratic process outlined in Table 1 and each of the eight steps became a separate prompt whose instructions restated the goal and the questions posed for that step. The full pipeline was then run on the holdout tasks with the three models above and read the intermediate output of every step, recording each case in which a step did not do what its definition prescribes. The observations fall into three groups.

Drift from the stated request. The reasoning moved away from what the task asked for. This was the most frequent problem and appeared with all three models. Initial Refutation discussed implementation choices and libraries instead of the problem itself; Premise Seeking reasoned about libraries even when the task did not ask for code, and proposed dependencies outside the Python standard library; Aporia Discovery raised gaps unrelated to the request; and on HumanEvalExplain the final answer sometimes generated code instead of the explanation that was asked for. In version 0.2, every step is instructed to stay anchored to the stated request, to skip implementation concerns unless the request calls for them, and to keep any dependency within the standard library unless the task states otherwise.

Identification without resolution. Questions and sub-problems were raised but left open. Initial Refutation asked the user for missing specifications instead of deriving them from the task, and Sub-Problem Inquiry listed sub-problems without resolving any. In version 0.2, both steps are instructed to close with an explicit result rather than an open question.

Table 4: Adequacy of the beta mixed-effects models. σ_{problem} is the standard deviation of the problem random intercept, uniformity, dispersion and outliers are the three DHARMA residual tests.

Metric	N	σ_{problem}	Uniformity (KS D)	Dispersion (ratio)	Outliers (count)	Verdict
pass@1	86,280	0.58	0.326	1.06	0	rejected
pass@2	86,280	0.71	0.286	1.14	0	rejected
pass@5	86,280	0.84	0.244	1.17	0	rejected
Beyond@1	14,760	0.55	0.123	0.98	0	rejected
Beyond@2	14,760	0.67	0.173	0.85	0	rejected
Beyond@5	14,760	0.75	0.198	0.70	3	rejected

Unconditional execution and cost. Output was produced even when there was nothing relevant to add. Every step generated text on every task, often at a length that consumed substantial time and tokens without contributing to the solution. In version 0.2, a step may return no output when it identifies nothing relevant, and every step must answer concisely in a fixed structure.

Each issue identified was translated into a revision of the corresponding part of the MIND pipeline, producing version 0.2. A round of inspection on the same tasks and models surfaced no further departure from the step definitions, so refinement stopped after a single round. Version 0.2 was frozen before the first task of the final evaluation was executed and applied unchanged to all 12 models. Both versions of MIND and the detailed notes taken during the process are available in the replication package for verifiability [18].

5.2  **RQ₁** — To what extent does the *MIND* approach improve the performance of LLMs compared to established prompting baselines across standard benchmarks?

To answer this research question we compared *MIND* with Chain-of-Thought, Self-Ask, ReAct and Self-Refine on the 12 models and 4 benchmarks selected in Section 4.1, with 5 independent generations per task, for a total of 505 200 generations (339 000 on BigCodeBench, 46 200 on each HumanEval variant and 73 800 on Mercury). Following the protocol of Section 4.3, we fitted one beta mixed-effects regression per metric and value of k , six in total (pass@ k and Beyond@ k at $k \in \{1, 2, 5\}$), each modeling the per-problem score as a function of prompting strategy, model family, size cluster and task type, with a random intercept per problem, after the rescaling of Smithson and Verkuilen [61]. All six converged, and the problem intercept confirms substantial heterogeneity between problems, but none passed the residual checks as can be seen in Table 4.

Therefore, the confirmatory inference for every metric comes from the non-parametric fallback. Each hypothesis is tested with a Friedman test on complete blocks, followed by Holm-corrected pairwise Wilcoxon signed-rank tests

Table 5: Mean pass@ k and Beyond@ k per prompting strategy, in percent, averaged over the 12 models and all tasks. Best value per row in bold.

Benchmark	k	MIND	CoT	ReAct	Self-Ask	Self-Refine
BigCodeBench	1	28.6	34.5	28.3	30.4	29.9
	2	37.2	42.5	37.6	38.8	38.1
	5	47.4	51.2	47.9	48.8	48.0
HumanEvalFix	1	36.8	42.7	27.6	35.0	40.3
	2	47.1	51.8	36.4	44.3	50.0
	5	57.8	61.9	46.4	56.4	60.7
HumanEvalExplain	1	46.7	55.8	43.4	51.4	49.4
	2	58.6	64.7	57.0	61.9	60.9
	5	70.2	73.9	72.1	73.6	73.0
Mercury	1	57.6	64.8	45.2	60.8	62.9
	2	69.6	74.9	58.3	70.6	73.1
	5	79.7	82.4	71.3	79.0	81.6

for the post-hoc comparisons; on the binary pass@ k outcomes the Friedman test coincides with Cochran’s Q . The beta regressions are not discarded entirely and their contrasts of estimated marginal means, expressed as odds ratios of *MIND* over each baseline with 95% confidence intervals, were kept as descriptive quantities. In what follows, the tests decide significance and the odds ratios describe magnitude.

Table 5 reports the mean score of each strategy per benchmark and k , and Figure 2 the score of every model $\times$ strategy configuration at $k = 1$. Chain-of-Thought obtains the highest mean on every benchmark and k . At $k = 1$, *MIND* ranks fourth of the five strategies on BigCodeBench, HumanEvalExplain and Mercury, and third on HumanEvalFix, where it overtakes Self-Ask. ReAct is last on all four. At $k = 1$ the gap between *MIND* and Chain-of-Thought ranges from 5.9 points on BigCodeBench and HumanEvalFix to 9.1 on HumanEvalExplain. Increasing k lifts every strategy and narrows the gaps. The order changes in a few places: *MIND* falls below ReAct on BigCodeBench at $k = 2$ and $k = 5$ and on HumanEvalExplain at $k = 5$, and it moves above Self-Ask on Mercury at $k = 5$. *MIND* never reaches the Chain-of-Thought mean.

Table 6 reports, for each metric and k , the Friedman statistic of each factor. All four null hypotheses of Section 4.3 are rejected in every case ($p < 0.001$ throughout), which means that the prompting strategy, the model family, the size cluster and the task type each have a significant effect on the scores. The size of the statistics nevertheless indicates that the factors do not weigh equally, even though the tests rest on different blockings and the comparison is therefore only indicative. The model family yields the largest statistic on

		BigCodeBench (pass@1)					HumanEvalFix (pass@1)				
Gemma	gemma4-4b	37.4	32.0	36.6	40.2	39.2	44.8	47.4	46.8	48.0	49.4
	gemma4-12b	44.6	45.3	24.3	46.0	47.3	51.3	50.6	5.8	49.4	52.6
	gemma4-31b	49.6	47.6	48.2	49.2	48.6	55.8	57.1	53.9	56.5	54.5
Llama	llama3.2-3b	10.0	18.3	16.7	17.4	16.5	13.6	26.0	6.5	28.6	27.3
	llama3.1-8b	15.5	29.3	12.8	29.0	9.4	21.4	29.9	10.4	35.1	23.4
	llama3.3-70b	34.1	41.9	41.7	42.3	22.0	42.2	44.8	3.9	28.6	32.5
Mistral	ministral3-3b	15.8	23.9	18.6	22.3	21.9	29.2	36.4	38.3	37.0	36.4
	ministral3-8b	20.1	32.0	29.4	30.2	20.4	35.1	46.1	44.8	41.6	29.9
	mixtral-42b	15.3	25.8	8.4	24.2	18.9	13.6	22.7	9.1	19.5	26.6
Qwen	qwen3.5-4b	27.6	36.9	33.5	14.2	36.2	27.9	46.8	26.6	8.4	47.4
	qwen3.5-9b	36.3	43.6	35.6	17.0	40.4	48.0	48.7	39.6	19.5	48.0
	qwen3.6-35b	37.2	37.7	34.2	33.3	37.9	58.4	55.8	46.1	48.0	55.8
		MIND	CoT	ReAct	Self-Ask	Self-Refine	MIND	CoT	ReAct	Self-Ask	Self-Refine

		HumanEvalExplain (pass@1)					Mercury (Beyond@1)				
Gemma	gemma4-4b	73.4	81.8	70.8	79.9	83.1	71.5	71.4	69.4	74.7	74.5
	gemma4-12b	91.6	90.3	9.1	87.0	93.5	82.0	80.4	15.3	81.3	82.0
	gemma4-31b	94.2	93.5	83.8	90.9	93.5	85.1	83.5	81.3	85.2	84.5
Llama	llama3.2-3b	20.1	29.2	20.1	33.1	36.4	32.8	36.9	17.0	51.7	44.5
	llama3.1-8b	29.2	32.5	18.2	34.4	16.9	46.0	51.6	15.9	60.2	32.2
	llama3.3-70b	48.7	59.1	47.4	45.5	38.3	72.4	72.9	31.6	71.5	58.3
Mistral	ministral3-3b	47.4	61.0	50.0	60.4	46.8	40.6	64.2	55.8	55.4	60.7
	ministral3-8b	44.8	72.1	60.4	62.3	48.0	55.1	68.1	67.7	71.6	65.9
	mixtral-42b	23.4	55.2	17.5	35.7	25.3	26.5	38.1	10.9	46.4	38.4
Qwen	qwen3.5-4b	11.7	14.3	35.7	11.0	28.6	43.6	64.5	49.7	20.8	61.0
	qwen3.5-9b	20.1	20.1	46.8	20.8	26.6	63.8	71.8	55.2	34.5	77.0
	qwen3.6-35b	55.8	60.4	61.0	55.8	55.8	71.8	74.0	72.5	76.3	75.2
		MIND	CoT	ReAct	Self-Ask	Self-Refine	MIND	CoT	ReAct	Self-Ask	Self-Refine

Fig. 2: Pass@1 and Beyond@1 of every model $\times$ strategy configuration, in percent. Models are ordered by size cluster.

every metric, about nine times that of the prompting strategy on pass@ k and about four times on Beyond@ k , while the strategy falls below the size cluster on pass@ k and is of comparable magnitude to it on Beyond@ k . The task type cannot be blocked by problem, because each problem belongs to a single task type, so it is tested on only 60 blocks of mean scores (one per model and strategy) against thousands of problem-level blocks for the other factors. Since the Friedman statistic grows with the number of blocks, its value cannot be set against theirs, and it is significant on all three pass@ k .

Table 7 compares *MIND* with each baseline on all six combinations of metric and k . Each comparison is summarised by an odds ratio, where a value below 1 indicates that *MIND* performs worse than the baseline and a value above 1 that it performs better, and the contrasts that are significant under the Holm-corrected post-hoc test are marked. *MIND* is significantly worse than Chain-of-Thought and than Self-Refine on all six combinations, and worse than Self-Ask on four of them, the exceptions being Beyond@2, where the two do not differ, and Beyond@5, where *MIND* is slightly ahead. ReAct is the only

Table 6: Tests of H_1 – H_4 per metric and k . Each cell reports the statistic of the confirmatory test (Friedman χ_F^2). Task type does not apply to Mercury and has no Beyond@ k test.

	H_1 Strategy (df=4)	H_2 Model family (df=3)	H_3 Model size (df=2)	H_4 Task type (df=2)
pass@1	524.6 $p < 0.001$	4560.7 $p < 0.001$	937.2 $p < 0.001$	58.4 $p < 0.001$
pass@2	388.1 $p < 0.001$	3507.3 $p < 0.001$	776.7 $p < 0.001$	64.1 $p < 0.001$
pass@5	235.8 $p < 0.001$	2182.4 $p < 0.001$	516.9 $p < 0.001$	63.1 $p < 0.001$
Beyond@1	238.8 $p < 0.001$	850.6 $p < 0.001$	254.3 $p < 0.001$	–
Beyond@2	205.5 $p < 0.001$	921.5 $p < 0.001$	200.4 $p < 0.001$	–
Beyond@5	226.1 $p < 0.001$	821.6 $p < 0.001$	131.7 $p < 0.001$	–

baseline that *MIND* outperforms, by a small margin on pass@1 and pass@2 and by a wide one on Mercury, whereas at pass@5 the two are not distinguishable. The size of these differences depends on the benchmark, since on the three correctness benchmarks all odds ratios lie between 0.87 and 1.04, which corresponds to modest differences that are nonetheless consistent enough across problems and models to be significant, whereas on Mercury the contrasts are more pronounced in both directions. Finally, the disadvantage with respect to Chain-of-Thought shrinks as k grows without disappearing, as expected when more candidate solutions are sampled, so that a larger k narrows the gap but does not close it.

We then examined whether the gap between *MIND* and the baselines depends on the size of the model or on the type of task, the two moderators defined in Section 4.3. Both interactions are significant on every metric, and Figure 3 shows how *MIND* compares with each baseline within each size cluster and each task type. Model size has the stronger influence, as with small models (under 8B) *MIND* is clearly behind Chain-of-Thought and Self-Refine on every metric, whereas with medium and large models it is consistently ahead of ReAct and no longer differs from Self-Refine. The gap to Chain-of-Thought shrinks with size as well, since it is significant on every metric for small models, on most of them for medium models and only at pass@1 for large ones. Task type has a weaker influence, because *MIND* trails Chain-of-Thought on all three task types and its advantage over ReAct appears only on repair, the task on which ReAct fails most often. Since these contrasts come from a regression that did not pass its checks, they are indicative rather

Table 7: RQ₁: *MIND* against each baseline. OR is the estimated-marginal-mean contrast of the beta model on the link scale (odds of *MIND* over the baseline) with 95% CI, reported as descriptive since the beta models are rejected; p is the Holm-corrected p -value of the confirmatory test (pairwise Wilcoxon signed-rank post-hoc of the Friedman test).

	CoT	ReAct	Self-Ask	Self-Refine
pass@1	0.87 [0.84, 0.90] $p < 0.001$	1.04 [1.00, 1.07] $p < 0.001$	0.96 [0.93, 1.00] $p < 0.001$	0.96 [0.93, 1.00] $p < 0.001$
pass@2	0.89 [0.86, 0.92] $p < 0.001$	1.02 [0.98, 1.06] $p = 0.004$	0.97 [0.94, 1.01] $p = 0.001$	0.97 [0.94, 1.01] $p = 0.001$
pass@5	0.92 [0.89, 0.95] $p < 0.001$	1.01 [0.97, 1.05] $p = 0.055$	0.97 [0.94, 1.00] $p < 0.001$	0.98 [0.94, 1.01] $p = 0.002$
Beyond@1	0.81 [0.74, 0.88] $p < 0.001$	1.46 [1.34, 1.59] $p < 0.001$	0.92 [0.84, 1.00] $p = 0.022$	0.86 [0.79, 0.94] $p = 0.003$
Beyond@2	0.85 [0.78, 0.93] $p < 0.001$	1.47 [1.35, 1.61] $p < 0.001$	0.97 [0.90, 1.06] $p = 0.777$	0.90 [0.83, 0.98] $p = 0.017$
Beyond@5	0.93 [0.86, 1.01] $p < 0.001$	1.40 [1.29, 1.52] $p < 0.001$	1.04 [0.96, 1.13] $p = 0.002$	0.96 [0.88, 1.04] $p = 0.050$

than confirmatory, but they point in one direction, namely that larger models reduce the disadvantage of *MIND* without ever turning it into an advantage over Chain-of-Thought.

The efficiency side of the trade-off is unambiguous. Table 8 summarises the per-task inference time of iteration 0, the only iteration whose timings are comparable across strategies (iterations 1 to 4 were executed in parallel and ran 2.2 to 3.1 times slower depending on the strategy). The median *MIND* call takes 20.4 s (median over models, interquartile range 17.9 to 24.1 s) against 5.8 s for Chain-of-Thought, 4.8 s for Self-Ask, 13.0 s for ReAct and 21.5 s for Self-Refine. Averaged over the 12 models, *MIND*’s median time is 4.9 times that of Chain-of-Thought (+389%), 4.6 times that of Self-Ask, 2.7 times that of ReAct and 1.5 times that of Self-Refine, and its interquartile range is 12.4 s wider than Chain-of-Thought’s and 13.4 s narrower than Self-Refine’s, whose iterative loop is the more variable of the two. The overhead is largest for medium models (+607% against Chain-of-Thought) and smallest for large ones (+209%), so the cluster in which *MIND* comes closest to the baselines in accuracy is also the one in which its relative cost is lowest. Weighing the two sides together, *MIND* delivers lower accuracy than the simplest baseline at several times its cost for every model size; only for the largest models does it reach parity with Chain-of-Thought on all but one comparison, and there at roughly three times the cost.

The efficiency side of the trade-off is unambiguous. Table 8 reports the per-task inference time of iteration 0, the only iteration with comparable timings

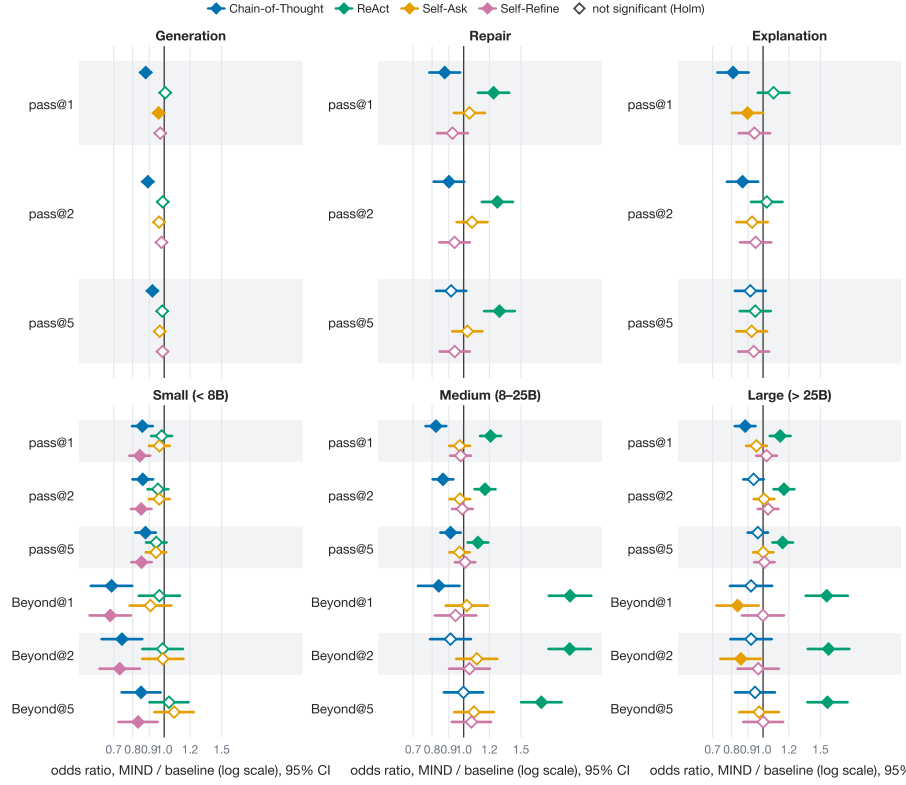


Fig. 3: RQ₁: odds ratios of *MIND* over each baseline within each task type (top) and size cluster (bottom), from the beta model with both interactions, with 95% confidence intervals on a log scale. Filled markers are significant after Holm correction within each set of four contrasts.

across strategies (iterations 1 to 4 ran in parallel and were 2.2 to 3.1 times slower). The median *MIND* call takes 20.4 s against 5.8 s for Chain-of-Thought and 21.5 s for Self-Refine. Since costs vary widely across models, we compute ratios within each model and average them. On this basis *MIND* takes 4.9 times as long as Chain-of-Thought (+389%), 4.6 times as long as Self-Ask, 2.7 times as long as ReAct and 1.5 times as long as Self-Refine. These ratios exceed those between the pooled medians because a few models inflate *MIND*'s cost (almost 16 times Chain-of-Thought on Qwen3.5-9B), while Self-Refine is far slower than *MIND* on Llama3.3-70B and Qwen3.5-9B, which raises its pooled median. The overhead is largest for medium models (+607% against Chain-of-Thought) and smallest for large ones (+209%), the cluster in which *MIND* also comes closest to the baselines in accuracy. Overall, *MIND* is less accurate than the simplest baseline at several times its cost for every model size, and

Table 8: *MIND*’s inference-time overhead against each baseline (iteration 0 only): relative change in the median per-task inference time and difference in interquartile range, mean, minimum and maximum over the 12 models.

Baseline	Median rel. change (mean)	(min, max)	IQR diff (s, mean)
CoT	+389%	(+53%, +1480%)	+12.4
ReAct	+166%	(-50%, +872%)	+5.3
Self-Ask	+357%	(-8%, +899%)	+10.8
Self-Refine	+53%	(-61%, +378%)	-13.4

even for the largest models, where it reaches parity with Chain-of-Thought on all but one comparison, it costs roughly three times as much.

🔍 **Answer to RQ₁.** *MIND* does not improve the performance of LLMs over the established prompting baselines. It is significantly worse than Chain-of-Thought and Self-Refine on every metric, trailing Chain-of-Thought by 6 to 9 points at $k = 1$ while taking a median of about four times its inference time per task. On the positive side, it is better than ReAct overall, mainly on the efficiency of the generated code, although not on every model. Unlike ReAct and Self-Ask, it never drops below half of the best strategy’s score on any model. Its disadvantage also shrinks with model size and becomes marginal above 25B parameters.

5.3  **RQ₂** — Which components of the *MIND* approach contribute most significantly to its overall performance?

To answer this research question, we evaluated seven single-step ablations of *MIND*, one for each step except 3.1 (*Solution Crafting*), which cannot be disabled without preventing solution generation. Each variant was run on all the 12 models and 4 benchmarks selected in Section 4.1, producing 336 distinct comparisons, 48 per removed step. For every model $\times$ benchmark cell we report $\Delta = \text{score}(\text{ablated}) - \text{score}(\text{full MIND})$, expressed in percentage points (pp)⁴, using pass@1 for BigCodeBench, HumanEvalFix and HumanEvalExplain and Beyond@1 for Mercury. Under this convention $\Delta > 0$ means that *removing* the step improved performance, that is, the step made a *negative* net contribution to the full pipeline. We characterize the contribution of each component through absolute and relative changes summarized with descriptive statistics, and we highlight the configurations that produce the largest changes.

⁴ The four benchmarks resolve a change with different granularity, since a single task accounts for at most 0.09 pp on BigCodeBench (1,130 tasks), 0.41 pp on Mercury (246 tasks) and 0.65 pp on the two HumanEval variants (154 tasks each).

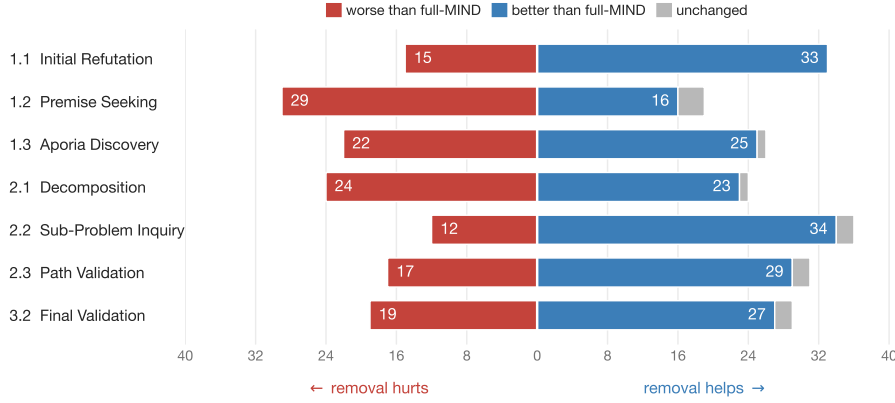


Fig. 4: Direction of the change in performance when each MIND step is removed. Bars extending to the right count the configurations in which removing the step improved performance relative to the full pipeline, bars extending to the left those in which it degraded performance, and the grey segment those in which the score was unchanged.

Table 9: Effect of removing each MIND step, pooled over 12 models $\times$ 4 benchmarks ($n = 48$ per step). Δ is the ablated minus the full-MIND score in percentage points. $\bar{\Delta}$, $\tilde{\Delta}$ and s are the mean, median and standard deviation of the 48 changes; $\tilde{\Delta}_{\text{rel}}$ is the median relative change; “i/d/t” counts the configurations that improved, degraded, or tied.

Removed step	$\bar{\Delta}$ (pp)	$\tilde{\Delta}$ (pp)	s (pp)	Min (pp)	Max (pp)	$\tilde{\Delta}_{\text{rel}}$ (%)	i/d/t
1.1 Initial Refutation	+1.25	+1.36	3.08	-6.49	+9.09	+3.44	33/15/0
1.2 Premise Seeking	-0.70	-0.57	3.31	-9.09	+7.14	-0.99	16/29/3
1.3 Aporia Discovery	+0.59	+0.22	2.98	-5.19	+9.74	+0.66	25/22/1
2.1 Decomposition	+0.08	-0.06	3.07	-5.84	+6.49	-0.07	23/24/1
2.2 Sub-Problem Inquiry	+2.01	+1.60	3.33	-5.84	+8.78	+5.05	34/12/2
2.3 Path Validation	+0.54	+0.65	2.50	-5.84	+7.14	+1.78	29/17/2
3.2 Final Validation	+0.52	+0.28	2.92	-7.79	+6.49	+1.18	27/19/2
All ablations pooled	+0.61	+0.53	3.11	-9.09	+9.74	+1.30	187/138/11

Of the 336 comparisons, 187 favored the ablated variant and 138 favored the full pipeline, with the remaining ones ending in an exact tie, so that removing a step improved performance somewhat more often than it degraded it.

This aggregate balance is not produced evenly by the seven steps, as Table 9 reports in full. The largest mean change came from removing the step 2.2 that performs the sub-problem inquiry (+2.01 pp), which improved 34 of the 48 configurations, followed by the step 1.1 that performed the initial refutation (+1.25 pp) with 33 improved. Moving in the opposite direction, step 1.2 that

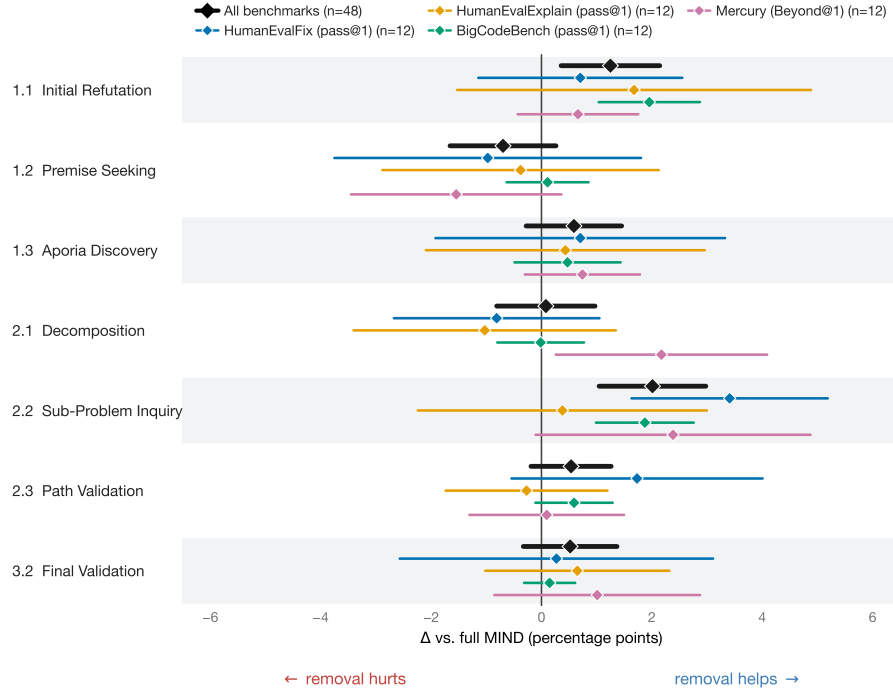


Fig. 5: Change in performance when each MIND step is removed, relative to the full pipeline. Diamonds are means, white ticks medians, bars the interquartile range, and grey dots the 48 individual model $\times$ benchmark configurations. The dark row of each band pools all benchmarks, and the four coloured rows below it show each benchmark separately.

did the premise seeking was the only step whose removal lowered performance on average (-0.70 pp), degrading 29 configurations against 16 improved. The four remaining steps occupy the space between these two poles, with means no larger than $+0.6$ pp and improved-to-degraded counts that stay close to even. These averages should be read against the dispersion of the underlying measurements, since every per-step standard deviation exceeds every per-step mean. The same ranking appears in Figure 5, where 2.2 is the only step whose interquartile range lies entirely above zero, with 1.1 the next closest, while the other five straddle it more evenly.

Considering the benchmarks separately, reported in Table 10 and visible in Figure 5, the two steps that stood out in aggregate behave consistently. Removing 2.2 produced a positive mean on all four benchmarks, at its strongest on HumanEvalFix ($+3.41$ pp), where 11 of 12 models improved, and at its weakest on HumanEvalExplain, where the models split almost evenly with 5 improved, 6 degraded and 1 tied. Removing 1.1 was likewise positive everywhere, ranging from $+1.95$ pp on BigCodeBench, again with 11 of 12 models

Table 10: RQ₂: change in performance when removing each MIND step, relative to the full framework (mean over the 12 models, absolute change in percentage points $\pm$ std). Mercury is evaluated with Beyond@1, the remaining benchmarks with pass@1.

Removed step	HumanEvalFix	HumanEvalExplain	BigCodeBench	Mercury
1.1	$+0.7 \pm 2.9$	$+1.7 \pm 5.0$	$+2.0 \pm 1.4$	$+0.7 \pm 1.7$
1.2	-1.0 ± 4.4	-0.4 ± 3.9	$+0.1 \pm 1.2$	-1.5 ± 3.0
1.3	$+0.7 \pm 4.1$	$+0.4 \pm 4.0$	$+0.5 \pm 1.5$	$+0.7 \pm 1.6$
2.1	-0.8 ± 2.9	-1.0 ± 3.7	-0.0 ± 1.2	$+2.2 \pm 3.0$
2.2	$+3.4 \pm 2.8$	$+0.4 \pm 4.1$	$+1.9 \pm 1.4$	$+2.4 \pm 3.9$
2.3	$+1.7 \pm 3.6$	-0.3 ± 2.3	$+0.6 \pm 1.1$	$+0.1 \pm 2.2$
3.2	$+0.3 \pm 4.5$	$+0.6 \pm 2.6$	$+0.1 \pm 0.7$	$+1.0 \pm 2.9$

improving, down to $+0.66$ pp on Mercury. The step that behaves differently across benchmarks is 2.1 that performs the decomposition of the problem, whose removal lowered the mean on HumanEvalFix, HumanEvalExplain, and BigCodeBench, but raised it by $+2.17$ pp on Mercury, where 9 of 12 models improved. A weaker mirror of this pattern appears for 1.2, which degraded three of the four benchmarks and reached its most negative value on Mercury, its only positive value being a tenth of a point on BigCodeBench. Aggregated over all steps the benchmarks differ little from one another, since their means lie within 0.58 pp of each other, and the dispersion within each cell remains large, with the standard deviations in Table 10 exceeding the corresponding means in all but three of the 28 cells.

Variation across models turns out to be at least as pronounced as variation across steps. Figure 6 reports every model $\times$ step cell, one panel per benchmark. Averaged over the seven ablations and the four benchmarks, the per-model means span 3.75 pp, from -1.55 pp for qwen3.5-9b up to $+2.20$ pp for ministral3-3b. This range is wider than the 2.71 pp separating the per-step means in Table 9 and considerably wider than the 0.58 pp separating the per-benchmark means. Within the matrix, gemma4-31b is the only model whose mean change is negative for all seven steps, and ministral3-3b the only one for which it is positive throughout.

Part of this variation follows model scale. Pooling all ablations within each size cluster, the mean change decreases from $+1.01$ pp for small models (< 8 B) to $+0.50$ pp for medium models (8–25B) and $+0.32$ pp for large models (> 25 B). The step-by-step breakdown in Table 11 shows that 2.2 follows this trend, with its largest value in the small cluster and nearly the same value in the two larger ones, whereas 1.1 does not and instead peaks in the medium cluster. Step 1.2 reaches its most negative value in the medium cluster as well.

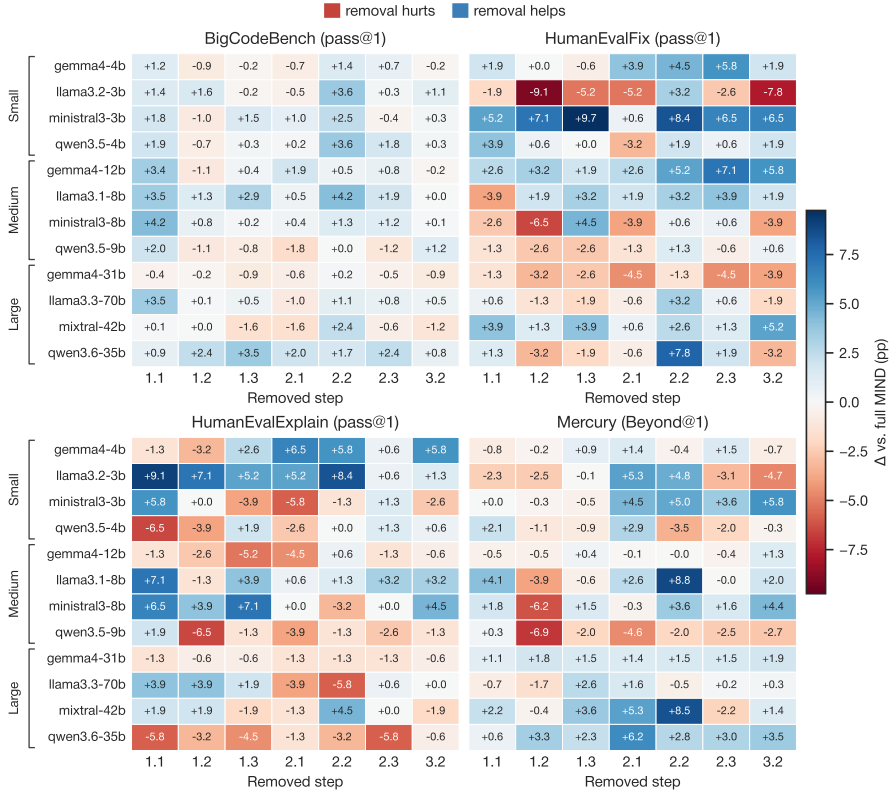


Fig. 6: Change in performance for every model and removed step, relative to the full pipeline, in percentage points. Blue cells indicate that removal improved performance and red cells that it degraded performance. One panel per benchmark.

Table 11: Mean change in performance (Δ vs. full MIND, percentage points) when each MIND step is removed, by model size cluster ($n = 16$ per cell).

Removed step	Small (< 8B)	Medium (8–25B)	Large (> 25B)
1.1	+1.35	+1.74	+0.65
1.2	−0.40	−1.75	+0.06
1.3	+0.67	+0.85	+0.24
2.1	+0.83	−0.62	+0.02
2.2	+3.02	+1.51	+1.51
2.3	+1.03	+0.73	−0.16
3.2	+0.58	+1.03	−0.05
All ablations	+1.01	+0.50	+0.32

🔗 **Answer to RQ₂.** The ablation study shows that some components individually shift the performance of the approach. Removing steps 2.2 and 1.1 raised the score most, by +2.01 and +1.25 pp, so their net contribution was negative. Step 1.2 was the only component whose removal lowered performance (−0.70 pp), and step 2.1 helped on the three correctness-oriented benchmarks but not on Mercury. Most changes stayed within two percentage points and varied more across models than across steps.

6 Further Analysis

The results of the two research questions leave two points open. RQ₁ shows that *MIND* scores lower than the baselines without explaining why, and RQ₂ shows that some of its steps reduce the score without telling whether a version of *MIND* without them would close the gap. We addressed both points with two additional experimental analyses.

6.1 Why does MIND fail?

To understand why *MIND* fails, we examined the solutions it produced and compared them with those of the baselines. We restricted the analysis to the first of the five attempts, because in this way every strategy contributes exactly one solution for each of the 20 208 combinations of model and task (1684 tasks for 12 models), and a failure of *MIND* can be set against what the other strategies did with the same model on the same problem. As the term of comparison we used Chain-of-Thought, which is the strongest baseline and the simplest one, since it answers in a single call and any difference can therefore be traced to the additional steps of *MIND*.

We began by asking whether *MIND* fails in a different way from Chain-of-Thought, because a strategy that mostly fails to deliver code would call for a different explanation than one that delivers wrong code. On the first attempt *MIND* fails in 12 758 of the 20 208 combinations and Chain-of-Thought in 11 425. We grouped these failures according to the outcome of their tests, depending on whether the code runs and returns a wrong result, raises an exception while running, does not run at all because of a syntax or naming error, is not usable because what was extracted from the answer contains no function, or exceeds the time or context limit. Table 12 shows that the two profiles are almost the same, with wrong results accounting for more than half of the failures of both strategies and runtime exceptions for about a third, so that *MIND* does not make a different kind of mistake but makes the same mistakes more often.

Most of these failures say little about *MIND*, because in 7878 of them no strategy solves the tasks in the benchmarks. We therefore isolated the cases in which a strategy fails while the other four succeed, where the model has

Table 12: Classes of failure on the first attempt, in percent of the failures of each group. The last two columns cover the cases in which the strategy failed while the other four strategies passed.

Failure class	<i>MIND</i>	CoT	<i>MIND</i> only	CoT only
Wrong result	53.8%	58.8%	47.2%	35.9%
Runtime exception	32.4%	29.3%	24.9%	19.0%
Code does not run	7.2%	7.3%	13.5%	34.4%
No usable code	5.4%	4.0%	13.3%	10.6%
Timeout or context overflow	1.2%	0.7%	1.2%	0.0%
Failures	12 758	11 425	511	273

shown that it can solve the specific task, and found 511 of them for *MIND* against 273 for Chain-of-Thought. The last two columns of Table 12 show that for both strategies these cases contain a larger share of solutions that do not run or cannot be used than the failures in general. The code is extracted from the answer by a separate call to the same model, and this call sometimes returned a placeholder in place of the code, almost always with one of the three Qwen models. This happened 68 times for *MIND* and 29 times for Chain-of-Thought, so that the handover of the answer accounts for only a small part of the difference between the two strategies.

To find the reasons behind the other failures, we read the complete reasoning of *MIND* for 24 of the 511 cases, two on average for each combination of benchmark and size cluster, drawn at random in proportion to the failures of each combination. All 24 cases are available in the replication package with their traces, solutions and test logs, and we refer to them by their identifier. In nine cases the answer could not be used, because the extraction returned no code (e.g., **case_18**) or because the explanation requested by HumanEvalExplain was cut by the length limit before reaching its content (e.g., **case_11**). In eight cases the code contains an ordinary programming error that no step introduced, such as a wrong comparison (**case_17**). The remaining seven cases depend on the reasoning itself and share the same pattern, in which a step that precedes the solution takes a decision that the final code then follows. In four of them the step settles a situation that the task leaves open in a way that the tests do not accept, as when Sub-Problem Inquiry decides that archiving the files of a missing directory must be reported as a failure, whereas the tests expect a success with nothing to move (**case_08**). In the other three the decision contradicts something that the task states, as when Initial Refutation assumes that the output is a sequence of bytes although the task asks for a string (**case_02**). These decisions first appear in five different steps, and in all seven cases both validation steps approve the solution.

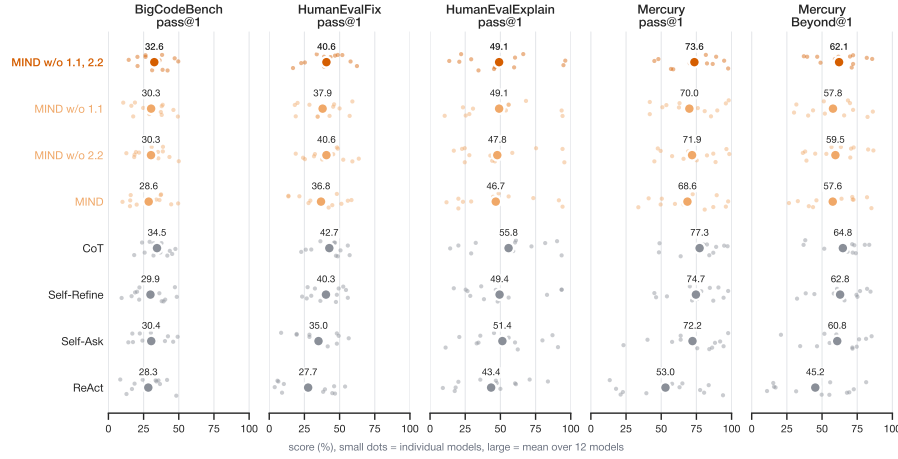


Fig. 7: Pass@1 on BigCodeBench, HumanEvalFix, and HumanEvalExplain, and pass@1 and Beyond@1 on Mercury, for MIND without steps 1.1 (*Initial Refutation*) and 2.2 (*Sub-Problem Inquiry*), the single-step ablations that remove 1.1 or 2.2 alone, full MIND, and the four baseline prompting strategies. Small dots are the 12 models; large dots are the mean.

👉 **Take-away.** *MIND* fails in the same way as Chain-of-Thought but more often. When it fails on a task that every baseline solves, the cause is an answer that cannot be used, an ordinary programming error, or a decision taken during the questioning steps that the task did not ask for and that the validation steps do not catch.

6.2 Removing both step 1.1 and 2.2

Observing the mean improvements in performance on several configurations with a single step removed (Section 5.3) led us to wonder whether *MIND* could be improved by further reducing its configuration. We selected the two steps whose removal led to the largest mean performance gains: 1.1 (*Initial Refutation*) and 2.2 (*Sub-Problem Inquiry*). This variant was, again, run on all the 12 models and 4 benchmarks selected in Section 4.1. Figure 7 shows the performance across benchmarks of removing both steps at once, compared to removing a single step, the full *MIND* approach, and the baselines.

Removing both steps raises *MIND*’s overall performance more than removing either step individually. Additionally, it narrows the gap with Chain-of-Thought, the best performing prompting approach, though the gap remains largest on HumanEvalExplain. Against full MIND the gains are +3.99 pp on BigCodeBench, +3.84 on HumanEvalFix, +2.38 on HumanEvalExplain, +4.95 on Mercury pass@1, and +4.51 on Mercury Beyond@1. Still, this re-

duced *MIND* configuration is 1.92, 2.06, 6.71, 3.69, and 2.68 pp, respectively, behind Chain-of-Thought.

Averaged over the five benchmark scores, the gain over full MIND and the gap that remains with Chain-of-Thought both depend on model size. Small models gain +3.17 pp and close 31% of the gap with Chain-of-Thought, remaining 6.98 pp behind. Medium models gain +5.10 pp and close 76%, remaining 1.63 pp behind. Large models gain +3.53 pp and close 68%, also remaining 1.63 pp behind.

This joint removal variant of *MIND* also changes the comparison with the other baselines. It beats ReAct on every model, by +4.27 to +20.56 pp. It beats Self-Ask on BigCodeBench (+2.18), HumanEvalFix (+5.63), and Mercury (+1.36 pass, +1.31 Beyond), but is still behind it on HumanEvalExplain (-2.33). Against Self-Refine it is ahead on BigCodeBench (+2.74), level on HumanEvalFix (+0.32), and just behind on HumanEvalExplain (-0.32) and Mercury (-1.08 pass, -0.74 Beyond).

The gains from removing both steps stack only on some benchmarks. This matches the per-benchmark pattern in Table 10. On HumanEvalFix, the joint removal leads to the same mean as removing 2.2 alone (both 40.64) and is +2.71 pp above removing 1.1. On HumanEvalExplain, it leads to the same mean as removing 1.1 alone (both 49.08) and is +1.3 pp above removing 2.2. On BigCodeBench the joint removal is +2.26 pp above removing 1.1 and +2.35 pp above removing 2.2. The same occurs for Mercury pass@1, where the joint removal is +3.59 pp above removing 1.1 and +1.66 pp above removing 2.2. This reinforces our previous observation from the ablation analysis (Section 5.3) that aggregating can hide that the contribution of a step depends on the benchmark. Interestingly, on HumanEvalFix and HumanEvalExplain the second removal neither improves nor hurts the mean: the joint removal ties with the best of the two single removals.

✚ **Take-away.** Removing steps 1.1 and 2.2 from MIND reduces the gap with Chain-of-Thought, and puts it level with the other baselines. The gain depends on the model and on the benchmark. Medium and large models close most of the gap with Chain-of-Thought, and small models close less than a third. On HumanEvalFix and HumanEvalExplain, removing a second step leaves overall performance unchanged.

7 Implications of the Study

MIND did not improve on the established baselines, but the way it fell short holds lessons that go beyond it. We first discuss the implications for practitioners who write prompts or build coding assistants, and then those for researchers who design and evaluate prompting techniques.

7.1 Implications for Practitioners

For anyone writing a multi-step prompt, the ablations give an indication of which Socratic steps are worth keeping, with one caution: most changes were small, and every per-step standard deviation exceeded the corresponding mean. The clearest signal concerns Initial Refutation and Sub-Problem Inquiry, the two steps that make the model doubt the request or the plan before any solution exists. Removing either of them improved about 70% of the configurations, and removing both raised the mean score on every benchmark (Section 6.2). Premise Seeking, which asks the model to recall what it knows, was the only step whose removal lowered the mean, but the difference is small and we read it as a weaker signal. The remaining steps, including Decomposition, changed the score by about half a point or less on average. The Socratic method is a dialogue in which one person holds a belief and another tests it, while in *MIND* the same model plays both roles. A question such as “where is this request vague?” assumes that something is vague, and with no new information the model can only answer with an assumption of its own, which the code then follows (Section 6.1). In all seven such cases we traced, both validation steps approved the result, in line with evidence that LLMs rarely correct their own reasoning without external feedback [30]. These are seven of the 24 cases we read, so the pattern is an indication to be tested on a larger sample. Self-critique in general is not the problem: Self-Refine, which asks the model to criticize a finished solution, was second or third on every benchmark and significantly better than *MIND* on every metric. The difference is in what is questioned. Self-Refine criticizes a concrete output, while the *MIND* steps with a negative contribution question the request and the plan before any solution exists.

 **Implication 1.** Questioning before a solution exists hurt, critique of a concrete output did not. Be careful with steps that make the model doubt the request or the plan with no new information: in the cases we traced, the model answered such doubts with assumptions and then approved them. The evidence for keeping the recall and planning steps is weaker.

The questions are still worth asking if someone other than the model answers them. Our first version left them open, with Initial Refutation asking the user for missing specifications and Sub-Problem Inquiry leaving its sub-problems unresolved. We treated this as a defect and told both steps to close with an explicit result (Section 5.1), although the Socratic method treats aporia, the admission that something is unknown, as progress. Forcing a closure may have encouraged the assumptions of Section 6.1. Questions about ambiguity can instead go to the developer, as in work that asks clarifying questions before generating code [48], and questions about counterexamples to the interpreter, which runs the code instead of imagining the result [9]. Execution could also catch the crashing or non-running code behind more than a third of the failures unique to *MIND*. Benchmarks with hidden tests are a poor judge

of such designs, because they reward the most common reading of a task, and a step that questions it has more to lose than to gain.

👉 **Implication 2.** Questioning needs someone other than the model to answer. Let questioning steps end with an open question and send it out, ambiguity to the developer and counterexamples to code execution. Judge such designs with real or simulated users, not with hidden tests alone.

7.2 Implications for Researchers

The model mattered more than the prompt (Section 5.2), and the same pipeline could even push two models in opposite directions. Averaged over the benchmarks, every step of *MIND* helped gemma4-31b and every step hurt ministral3-3b, so conclusions about which steps to keep should not be carried from one model size to another. Averages also hide collapses, such as ReAct and Self-Ask solving almost no repair tasks with some models while the best strategy for the same model solved close to half. Time was a fragile measure of cost as well, because iterations run in parallel were two to three times slower and only the first could be used.

👉 **Implication 3.** Averages hide the models that a technique helps, hurts or breaks. Report results per model with their worst cases, compare techniques at equal cost in tokens and calls rather than wall-clock time, and count answers lost to extraction apart from reasoning failures.

Our prompt evaluation (Section 5.1) addressed every defect we saw, and a second round found each step doing what its definition asked. Yet removing most steps from the frozen version raised the mean score. Reading outputs shows that a step does its job, not that the job helps, and effects of one or two percentage points cannot be seen on ten tasks per benchmark, where one task is worth ten percentage points. The instruction fixes also did not fully hold at scale, since the drift away from the request came back in Section 6.1.

👉 **Implication 4.** Inspecting outputs cannot tell whether a step helps. Before freezing a multi-step prompt, measure what each step adds on a development set large enough to detect effects of some percentage points, and check at scale that fixes made through instructions hold.

8 Threats to Validity

Internal validity. A primary concern lies in the variability introduced by stochastic generation in LLM outputs. To mitigate this, all models are evaluated under fixed decoding parameters and each prompting condition is applied

independently across all models and benchmarks. Because pass@ k and Beyond@ k aggregate multiple samples, they reduce but do not eliminate stochastic noise. Another source of internal threat is the correctness of the benchmark execution pipelines. We address this by using the official test suites provided by the datasets and by exporting raw model outputs and execution logs to allow post-hoc inspection. The noise matters most for the ablation study, which uses a single generation per task ($k = 1$): most of the changes we observed are within two percentage points and every per-step standard deviation exceeds the corresponding mean, so we treat the per-step differences as indications and not as confirmed effects. A second threat concerns the prompts. The prompts of both the baselines and *MIND* were checked on the same holdout tasks, but the baselines started from the templates of their original papers while *MIND* went through one round of revision, and a different amount of tuning could favor one side or the other. A third threat concerns the handling of the answers. The code is extracted from each answer by a separate call to the same model, and long answers can reach the length or context limit. Both problems affect every strategy, but *MIND* is more exposed to them than Chain-of-Thought, as the shares of unusable answers and context overflows in Table 12 show. Finally, the size clusters are not a clean comparison, because within a family the models of different sizes also belong to different releases (e.g., LLaMA 3.2, 3.1 and 3.3), so the effect of size cannot be fully separated from the effect of the model generation.

Construct validity. *MIND* is one possible operationalization of the Socratic method. The mapping of its three phases onto eight prompts involves choices of our own, such as having the same model act as questioner and answerer and requiring the questioning steps to close with an explicit result, so our findings concern this operationalization and not the Socratic method in general. On the measurement side, pass@ k is computed on the first k of the five generations of each task, so pass@1 depends on a single sample per task, and Beyond@ k is our adaptation of the Beyond metric of Mercury. Inference time is measured as wall-clock time on a single worker, which depends on the hardware and on the serving backend and is only a proxy of the cost in tokens and calls.

External validity. Our selection of four benchmark families offers broad coverage of code-related tasks but cannot represent all programming domains or prompt scenarios. Likewise, we include multiple LLM families and sizes to improve generalizability, yet the results may not transfer to proprietary models, multimodal models, or models trained with different architectural priors. Similarly, the upper bound of 70B parameters is dictated by resource constraints, and larger-scale models could exhibit qualitatively different behaviour. Ablation effects might also depend on model scale in ways not fully captured by the three size categories we employ.

Conclusion validity. The beta mixed-effects models planned in the protocol did not pass the residual checks, so the confirmatory inference comes from the non-parametric fallback, as the protocol prescribes. Type I and Type II errors remain possible given the large number of comparisons across models,

benchmarks, baselines, and ablations. We address this through Holm–Bonferroni correction. The odds ratios and confidence intervals we report come from the rejected beta models and are descriptive only, as are the contrasts by size cluster and task type. The ablation study and the analyses of Section 6 are also descriptive and were not tested statistically. In particular, the variant without steps 1.1 and 2.2 was selected and evaluated on the same tasks, so its gain may be optimistic. Finally, given the size of the dataset, even small differences reach statistical significance, and statistical significance may not always align with practical significance, especially in tasks with small absolute performance differences or high variance in model outputs.

9 Conclusions

In this paper we introduced *MIND*, a prompting framework that leads a model through the three phases of the Socratic method. We compared it with Chain-of-Thought, Self-Ask, ReAct and Self-Refine on four code benchmarks and twelve open-weight models of three sizes, removed its steps one at a time, and examined the tasks that only *MIND* failed. *MIND* did not improve on the baselines. Among them it outperformed only ReAct, and it trailed Chain-of-Thought, the simplest one, by six to nine percentage points on the first attempt while taking several times longer, a gap that shrank with model size without closing. The steps that ask the model to recall what it knows or to plan the work helped, while those that ask it to doubt or judge its own reasoning slightly. *MIND* failed in the same ways as Chain-of-Thought but more often, and in the cases we read, the failures that depended on its reasoning came from decisions taken during questioning that the validation steps let through. The main lesson is that the Socratic method does not survive being turned into a monologue, because its questions need someone other than the model, such as the developer or the interpreter, to answer them. More broadly, the model mattered more than the prompt, several samples of a simple prompt can beat one long pipeline at equal cost, inspecting outputs cannot show whether a step helps, and a pre-registered analysis must fit how its metric is produced. Whether the questioning steps help once someone else answers them is the question we leave for future work.

Data Availability

To support the reproducibility of our study and enable future research advancements, we provide all materials, including scripts and datasets, through an online appendix [18].

References

1. BALTES, S., ANGERMEIR, F., ARORA, C., BARÓN, M. M., CHEN, C., BÖHME, L., CALEFATO, F., ERNST, N., FALESSI, D., FITZGERALD, B., ET AL. Guidelines for em-

- pirical studies in software engineering involving large language models. *arXiv preprint arXiv:2508.15503* (2025).
2. BENSON, H. H. *Socratic wisdom: the model of knowledge in Plato's early dialogues*. Oxford University Press, 2000.
 3. BINDER, F. J., CHUA, J., KORBAK, T., SLEIGHT, H., HUGHES, J., LONG, R., PEREZ, E., TURPIN, M., AND EVANS, O. Looking inward: Language models can learn about themselves by introspection. In *International Conference on Learning Representations* (2025), vol. 2025, pp. 3710–3756.
 4. CEKA, I., PUJAR, S., MANOTAS, I., KAISER, G., RAY, B., AND RAMJI, S. How does llm reasoning work for code? a survey and a call to action. *arXiv preprint arXiv:2506.13932* (2025).
 5. CHAND, S., KILIC, M., WÜRSCHING, R., PANDEY, S. K., AND PRETSCHNER, A. Automated extract method refactoring with open-source llms: A comparative study. *arXiv preprint arXiv:2510.26480* (2025).
 6. CHANG, E. Y. Prompting large language models with the socratic method. In *2023 IEEE 13th annual computing and communication workshop and conference (CCWC)* (2023), IEEE, pp. 0351–0360.
 7. CHEN, B., ZHANG, F., NGUYEN, A., ZAN, D., LIN, Z., LOU, J.-G., AND CHEN, W. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397* (2022).
 8. CHEN, M., TWOREK, J., JUN, H., YUAN, Q., DE OLIVEIRA PINTO, H. P., KAPLAN, J., EDWARDS, H., BURDA, Y., JOSEPH, N., BROCKMAN, G., RAY, A., PURI, R., KRUEGER, G., PETROV, M., KHLAUF, H., SASTRY, G., MISHKIN, P., CHAN, B., GRAY, S., RYDER, N., PAVLOV, M., POWER, A., KAISER, L., BAVARIAN, M., WINTER, C., TILLET, P., SUCH, F. P., CUMMINGS, D., PLAPPERT, M., CHANTZIS, F., BARNES, E., HERBERT-VOSS, A., GUSS, W. H., NICHOL, A., PAINO, A., TEZAK, N., TANG, J., BABUSCHKIN, I., BALAJI, S., JAIN, S., SAUNDERS, W., HESSE, C., CARR, A. N., LEIKE, J., ACHIAM, J., MISRA, V., MORIKAWA, E., RADFORD, A., KNIGHT, M., BRUNDAGE, M., MURATI, M., MAYER, K., WELINDER, P., MCGREW, B., AMODEI, D., MCCANDLISH, S., SUTSKEVER, I., AND ZAREMBA, W. Evaluating large language models trained on code, 2021.
 9. CHEN, X., LIN, M., SCHÄRLI, N., AND ZHOU, D. Teaching large language models to self-debug. In *International Conference on Learning Representations* (2024), vol. 2024, pp. 8746–8825.
 10. CHEN, X., TAO, Z., ZHANG, K., ZHOU, C., ZHANG, X., GU, W., HE, Y., ZHANG, M., CAI, X., ZHAO, H., ET AL. Revisit self-debugging with self-generated tests for code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2025), pp. 18003–18023.
 11. CHIDAMBARAM, S., LI, L. E., BAI, M., LI, X., LIN, K., ZHOU, X., AND WILLIAMS, A. C. Socratic human feedback (sohf): Expert steering strategies for llm code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024* (2024), pp. 15491–15502.
 12. CICALESE, M., DELLA PORTA, A., LAMBIASE, S., IANNONE, E., HINRICHS, T., SCANDARIATO, R., AND PALOMBA, F. The language of security: How prompt syntax shapes secure code generation in open llms. *arXiv preprint arXiv:2607.15937* (2026).
 13. DASGUPTA, I., LAMPINEN, A. K., CHAN, S. C., CRESWELL, A., KUMARAN, D., MCCLELLAND, J. L., AND HILL, F. Language models show human-like content effects on reasoning. *arXiv preprint arXiv:2207.07051* 2, 3 (2022).
 14. DELLA PORTA, A., LAMBIASE, S., AND PALOMBA, F. Do prompt patterns affect code quality? a first empirical assessment of chatgpt-generated code. *arXiv preprint arXiv:2504.13656* (2025).
 15. DELLA PORTA, A., LAMBIASE, S., PONTILLO, V., FERREIRA, J. F., RAHMAN, A. A., RAGKHITWETSAGUL, C., AND PALOMBA, F. Understanding prompt quality and its relation to llm-based code generation. *IEEE Transactions on Software Engineering* (2026).
 16. DELLA PORTA, A., RECUPITO, G., LAMBIASE, S., DI NUCCI, D., AND PALOMBA, F. Unlocking code simplicity: The role of prompt patterns in managing llm code complexity. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering-Companion (SANER-C)* (2025), IEEE, pp. 140–143.
 17. DELLA PORTA, A., RICHARDS, J., CAMMAROTA, L., LAMBIASE, S., PALOMBA, F., AND WESSEL, M. Ask, then think: Enhancing llm performance with socratic reasoning. Available at SSRN 6167610 (2026).

18. DELLA PORTA, A., RICHARDS, J., CAMMAROTA, L., LAMBIASE, S., PALOMBA, F., AND WESSEL, M. Replication package for "ask, then think: Does socratic prompting help llms on code tasks?". <https://doi.org/10.5281/zenodo.23322894>, Oct. 2026.
19. DELLA PORTA, A., VORIA, G., ABBATE, A., SULIPANO, R., LAMBIASE, S., CATOLINO, G., AND PALOMBA, F. Toward measuring prompt quality: A preliminary investigation on prompt smells. In *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering-Companion (SANER-C)* (2026), IEEE, pp. 293–300.
20. DHULIAWALA, S., KOMEILI, M., XU, J., RAILEANU, R., LI, X., CELIKYILMAZ, A., AND WESTON, J. Chain-of-verification reduces hallucination in large language models. In *Findings of the association for computational linguistics: ACL 2024* (2024), pp. 3563–3578.
21. DIDOLKAR, A., GOYAL, A., KE, N. R., GUO, S., VALKO, M., LILLICRAP, T., REZENDE, D., BENGIO, Y., MOZER, M., AND ARORA, S. Metacognitive capabilities of llms: An exploration in mathematical problem solving. *Advances in Neural Information Processing Systems* 37 (2024), 19783–19812.
22. DONG, Y., JIANG, X., JIN, Z., AND LI, G. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–38.
23. DU, M., TUAN, L. A., JI, B., LIU, Q., AND NG, S.-K. Mercury: A code efficiency benchmark for code large language models. *Advances in Neural Information Processing Systems* 37 (2024).
24. FAN, A., GOKKAYA, B., HARMAN, M., LYUBARSKIY, M., SENGUPTA, S., YOO, S., AND ZHANG, J. M. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)* (2023), IEEE, pp. 31–53.
25. FLAVELL, J. H. Metacognition and cognitive monitoring: A new area of cognitive-developmental inquiry. *American psychologist* 34, 10 (1979), 906.
26. GUO, D., YANG, D., ZHANG, H., SONG, J., WANG, P., ZHU, Q., XU, R., ZHANG, R., MA, S., BI, X., ET AL. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature* 645, 8081 (2025), 633–638.
27. HACKFORTH, R. *Plato: phaedrus*. No. 119. Cambridge University Press, 1972.
28. HAYES-ROTH, B., AND HAYES-ROTH, F. A cognitive model of planning. *Cognitive science* 3, 4 (1979), 275–310.
29. HOU, X., ZHAO, Y., LIU, Y., YANG, Z., WANG, K., LI, L., LUO, X., LO, D., GRUNDY, J., AND WANG, H. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.
30. HUANG, J., CHEN, X., MISHRA, S., ZHENG, H. S., YU, A., SONG, X., AND ZHOU, D. Large language models cannot self-correct reasoning yet. In *International conference on learning representations* (2024), vol. 2024, pp. 32808–32824.
31. ISLAM, M. A., ALI, M. E., AND PARVEZ, M. R. Mapcoder: Multi-agent code generation for competitive problem solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2024), pp. 4912–4944.
32. JIANG, X., DONG, Y., WANG, L., FANG, Z., SHANG, Q., LI, G., JIN, Z., AND JIAO, W. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–30.
33. JIN, N., LI, Z., YANG, G., ZHANG, T., AND ZENG, Q. Understanding chain-of-thought effectiveness in code generation: an empirical and information-theoretic analysis. *Empirical Software Engineering* 32, 1 (2027), 22.
34. JUNG, J., QIN, L., WELLECK, S., BRAHMAN, F., BHAGAVATULA, C., LE BRAS, R., AND CHOI, Y. Maieutic prompting: Logically consistent reasoning with recursive explanations. In *Proceedings of the 2022 conference on empirical methods in natural language processing* (2022), pp. 1266–1279.
35. KADAVATH, S., CONERLY, T., ASKELL, A., HENIGHAN, T., DRAIN, D., PEREZ, E., SCHIEFER, N., HATFIELD-DODDS, Z., DASARMA, N., TRAN-JOHNSON, E., ET AL. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221* (2022).
36. KHOJAH, R., DE OLIVEIRA NETO, F. G., MOHAMAD, M., AND LEITNER, P. The impact of prompt programming on function-level code generation. *IEEE Transactions on Software Engineering* (2025).

37. KOJIMA, T., GU, S. S., REID, M., MATSUO, Y., AND IWASAWA, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.
38. KWON, W., LI, Z., ZHUANG, S., SHENG, Y., ZHENG, L., YU, C. H., GONZALEZ, J., ZHANG, H., AND STOICA, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles* (2023), pp. 611–626.
39. LAMPINEN, A. K., DASGUPTA, I., CHAN, S. C., SHEAHAN, H. R., CRESWELL, A., KUMARAN, D., MCCLELLAND, J. L., AND HILL, F. Language models, like humans, show content effects on reasoning tasks. *PNAS nexus* 3, 7 (2024).
40. LEUNG, S. C. The cognitive impacts of large language model interactions on problem solving and decision making using eeg analysis. *Frontiers in Computational Neuroscience* 19 (2025), 1556483.
41. LI, J., LI, G., LI, Y., AND JIN, Z. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 1–23.
42. LI, J., TAO, C., LI, J., LI, G., JIN, Z., ZHANG, H., FANG, Z., AND LIU, F. Large language model-aware in-context learning for code generation. *ACM Transactions on Software Engineering and Methodology* 34, 7 (2025), 1–33.
43. LI, S., HUANG, L., ZHAN, S., SUN, W., YIN, T., LIU, Z., AND YAN, M. Intention chain-of-thought prompting with dynamic routing for code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2026), vol. 40, pp. 650–658.
44. LINDSEY, J., GURNEE, W., AMEISEN, E., CHEN, B., PEARCE, A., TURNER, N. L., CITRO, C., ABRAHAMS, D., CARTER, S., HOSMER, B., MARCUS, J., SKLAR, M., TEMPLETON, A., BRICKEN, T., MCDUGALL, C., CUNNINGHAM, H., HENIGHAN, T., JERMYN, A., JONES, A., PERSIC, A., QI, Z., THOMPSON, T. B., ZIMMERMAN, S., RIVOIRE, K., CONERLY, T., OLAH, C., AND BATSON, J. On the biology of a large language model. *Transformer Circuits Thread*, 2025.
45. LYU, M. R., RAY, B., ROYCHOUDHURY, A., TAN, S. H., AND THONGTANUNAM, P. Automatic programming: Large language models and beyond. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–33.
46. MADAAN, A., TANDON, N., GUPTA, P., HALLINAN, S., GAO, L., WIEGREFFE, S., ALON, U., DZIRI, N., PRABHUMOYE, S., YANG, Y., ET AL. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2023), 46534–46594.
47. MASTROPAOLO, A., PASCARELLA, L., GUGLIELMI, E., CINISELLI, M., SCALABRINO, S., OLIVETO, R., AND BAVOTA, G. On the robustness of code generation techniques: An empirical study on github copilot. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)* (2023), IEEE, pp. 2149–2160.
48. MU, F., SHI, L., WANG, S., YU, Z., ZHANG, B., WANG, C., LIU, S., AND WANG, Q. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2332–2354.
49. MUENNIGHOFF, N., LIU, Q., ZEBAZE, A., ZHENG, Q., HUI, B., ZHUO, T. Y., SINGH, S., TANG, X., VON WERRA, L., AND LONGPRE, S. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124* (2023).
50. NELSON, T. O. Metamemory: A theoretical framework and new findings. In *Psychology of learning and motivation*, vol. 26. Elsevier, 1990, pp. 125–173.
51. OLAUSSON, T. X., INALA, J. P., WANG, C., GAO, J., AND SOLAR-LEZAMA, A. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations* (2024), vol. 2024, pp. 36545–36593.
52. PAN, R., AND ZHANG, H. Modularization is better: Effective code generation with modular prompting. *arXiv preprint arXiv:2503.12483* (2025).
53. PLAAT, A., WONG, A., VERBERNE, S., BROEKENS, J., AND VAN STEIN, N. Multi-step reasoning with large language models, a survey. *ACM Computing Surveys* (2025).
54. PLAAT, A., WONG, A., VERBERNE, S., BROEKENS, J., VAN STEIN, N., AND BACK, T. Reasoning with large language models, a survey. *arXiv preprint arXiv:2407.11511* (2024).
55. PRESS, O., ZHANG, M., MIN, S., SCHMIDT, L., SMITH, N. A., AND LEWIS, M. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023* (2023), pp. 5687–5711.

56. QI, J., XU, Z., SHEN, Y., LIU, M., JIN, D., WANG, Q., AND HUANG, L. The art of socratic questioning: Recursive thinking with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (2023), pp. 4177–4199.
57. RENZE, M., AND GUVEN, E. Self-reflection in large language model agents: Effects on problem-solving performance. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)* (2024), IEEE, pp. 516–525.
58. RIDNIK, T., KREDO, D., AND FRIEDMAN, I. Code generation with alphacodium: From prompt engineering to flow engineering. *arXiv preprint arXiv:2401.08500* (2024).
59. SABOUR, S., LIU, S., ZHANG, Z., LIU, J., ZHOU, J., SUNARYO, A., LEE, T., MIHALCEA, R., AND HUANG, M. Emobench: Evaluating the emotional intelligence of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2024), pp. 5986–6004.
60. SHINN, N., CASSANO, F., GOPINATH, A., NARASIMHAN, K., AND YAO, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems* 36 (2023), 8634–8652.
61. SMITHSON, M., AND VERKUILEN, J. A better lemon squeezer? maximum-likelihood regression with beta-distributed dependent variables. *Psychological methods* 11, 1 (2006), 54.
62. SPRAGUE, Z., YIN, F., RODRIGUEZ, J., JIANG, D., WADHWA, M., SINGHAL, P., ZHAO, X., YE, X., MAHOWALD, K., AND DURRETT, G. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *International Conference on Learning Representations* (2025), vol. 2025, pp. 94118–94162.
63. VLASTOS, G. The socratic elenchus. *The Journal of Philosophy* 79, 11 (1982), 711–714.
64. WANG, G., SUN, Z., YE, S., GONG, Z., CHEN, Y., ZHAO, Y., LIANG, Q., AND HAO, D. Do advanced language models eliminate the need for prompt engineering in software engineering? *ACM Transactions on Software Engineering and Methodology* 35, 8 (2026), 1–33.
65. WANG, G., WU, W., YE, G., CHENG, Z., CHEN, X., AND ZHENG, H. Decoupling metacognition from cognition: a framework for quantifying metacognitive ability in llms. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2025), vol. 39, pp. 25353–25361.
66. WANG, L., XU, W., LAN, Y., HU, Z., LAN, Y., LEE, R. K.-W., AND LIM, E.-P. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (2023), pp. 2609–2634.
67. WANG, X., HU, Z., LU, P., ZHU, Y., ZHANG, J., SUBRAMANIAM, S., LOOMBA, A. R., ZHANG, S., SUN, Y., AND WANG, W. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635* (2023).
68. WANG, X., WEI, J., SCHUURMANS, D., LE, Q., CHI, E., NARANG, S., CHOWDHURY, A., AND ZHOU, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).
69. WANG, Y., AND ZHAO, Y. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (2024), pp. 1914–1926.
70. WEI, J., WANG, X., SCHUURMANS, D., BOSMA, M., XIA, F., CHI, E., LE, Q. V., ZHOU, D., ET AL. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
71. WENG, Y., ZHU, M., XIA, F., LI, B., HE, S., LIU, S., SUN, B., LIU, K., AND ZHAO, J. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023* (2023), pp. 2550–2575.
72. WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M. C., REGNELL, B., WESSLÉN, A., ET AL. *Experimentation in software engineering*, vol. 236. Springer, 2012.
73. XU, F., LIN, Q., HAN, J., ZHAO, T., LIU, J., AND CAMBRIA, E. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering* (2025).

74. YAO, S., YU, D., ZHAO, J., SHAFRAN, I., GRIFFITHS, T., CAO, Y., AND NARASIMHAN, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems* 36 (2023), 11809–11822.
75. YAO, S., ZHAO, J., YU, D., DU, N., SHAFRAN, I., NARASIMHAN, K. R., AND CAO, Y. Re-act: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations* (2022).
76. ZHONG, L., WANG, Z., AND SHANG, J. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics: ACL 2024* (2024), pp. 851–870.
77. ZHOU, D., SCHÄRLI, N., HOU, L., WEI, J., SCALES, N., WANG, X., SCHUURMANS, D., CUI, C., BOUSQUET, O., LE, Q., ET AL. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022).
78. ZHUO, T. Y., VU, M. C., CHIM, J., HU, H., YU, W., WIDYASARI, R., YUSUF, I. N. B., ZHAN, H., HE, J., PAUL, I., ET AL. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).