

Can LLMs Reason in the Wild with Programs?

Yuan Yang^{1*}, Siheng Xiong¹, Ali Payani², Ehsan Shareghi³ & Faramarz Fekri¹

¹Georgia Institute of Technology, ²Cisco, ³Monash University
mblackout@hotmail.com* {sxiong45@, faramarz.fekri@ece.}gatech.edu
apayani@cisco.com ehsan.shareghi@monash.edu

Abstract

Large Language Models (LLMs) have shown superior capability to solve reasoning problems with programs. While being a promising direction, most of such frameworks are trained and evaluated in settings with a prior knowledge of task requirements. However, as LLMs become more capable, it is necessary to assess their reasoning abilities in more realistic scenarios where many real-world problems are open-ended with ambiguous scope, and often require multiple formalisms to solve. To investigate this, we introduce the *reasoning in the wild* task, where an LLM is tasked to solve a reasoning problem of unknown type by identifying the sub-problems and their corresponding formalisms, and writing a program to solve each sub-problem, guided by a *tactic*. We create a large tactic-guided trajectory dataset containing detailed solutions to a diverse set of reasoning problems, ranging from well-defined single-form reasoning (e.g., math, logic), to ambiguous and hybrid ones (e.g., commonsense, combined math and logic). This allows us to test various aspects of LLMs reasoning at the fine-grained level such as the selection and execution of tactics, and the tendency to take undesired shortcuts. In experiments, we highlight that existing LLMs fail significantly on problems with ambiguous and mixed scope, revealing critical limitations and overfitting issues (e.g. accuracy on GSM8K drops by at least 50%). We further show fine-tuning a local LLM on the trajectories data leads to better performance. Project repo is available [here](#).

1 Introduction

Large language models (LLMs) demonstrate strong capabilities in solving a wide variety of reasoning problems that involve different formalisms such as formal logic, math, graph, and commonsense reasoning. As an example, recent LLMs achieve above 90% (Figure 1) accuracy on math

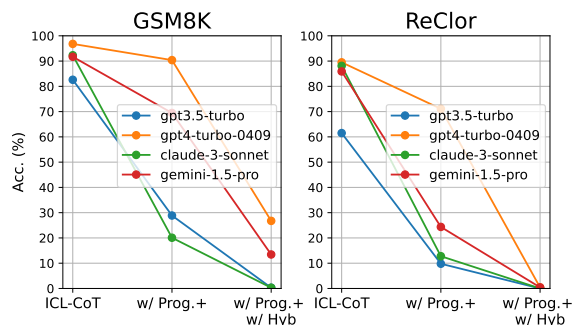


Figure 1: Commonly used metrics such as accuracy, while suggesting LLMs perform well on reasoning tasks in free-form (*ICL-CoT*), fail to reflect their capability in a more fine-grained reasoning benchmark. We find LLMs’ accuracy drops significantly when tasked to solve problems with a non-trivial program (*w/ Prog.+*); this further decreases when tasked to reason in the wild, where the tasks are blended with other contexts and their reasoning types are unknown (*w/ Hyb*).

benchmarks such as GSM8K (Cobbe et al., 2021), and above 80% accuracy on commonsense reasoning benchmarks such as ReClor (Yu et al., 2020). However, these results are typically obtained in the in-context learning (ICL) setting with answers generated in free text. Such a way is criticized for lacking reliability and is prone to question distribution shift and hallucination (Kabra et al., 2023; Schick et al., 2023; Hao et al., 2023).

Agent-like¹ LLM frameworks show great potential in addressing these limitations, which solves reasoning tasks by writing programs (Yao et al., 2022; Gou et al., 2023; Yuan et al., 2024). While being a promising direction, most of these frameworks are trained and evaluated on benchmarks with well-defined scope and a clear formalism to solve, such as GSM8K, where questions are guaranteed to be an arithmetic problem that can be solved with a single math model. Such

¹We use "Agent-like" as many of the existing approaches that augment LLMs with external mechanisms only have partial components of a language agent (Sumers et al., 2023).

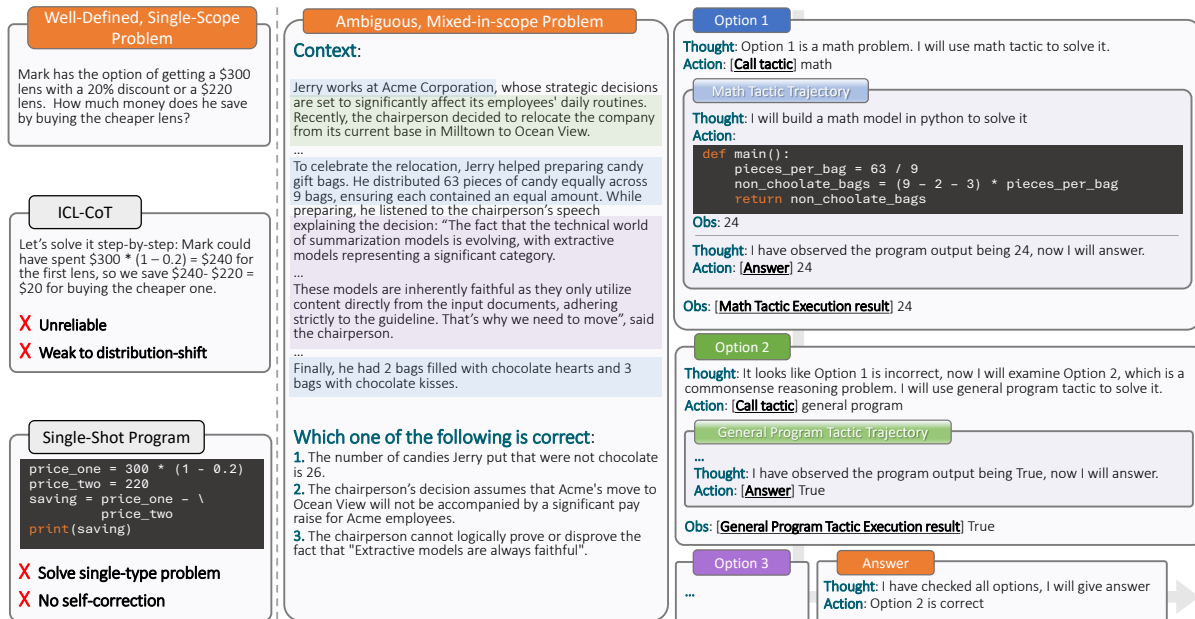


Figure 2: Solving ambiguous, mixed-in-scope problem via reasoning in the wild.

an assumption does not hold in real-world scenarios, where the problems could: (1) **be ambiguous in scope**, where the problem is not a well-defined math or formal logic task, and the program written cannot fully solve the problem and requires further reasoning to derive the final answer; (2) **require multiple formalisms to solve**, where the problem needs to be decomposed into subproblems with each requiring different sets of skills to solve; (3) **be mixed with irrelevant context**, where the context contains irrelevant information that needs to be excluded when building a formal model. Such problems are widely present in the real world, for example, Figure 2 shows an example of a multichoice question, where Option 1 requires a math model to verify, Option 2 requires a generic program, which does not fit into any of the well-defined formalisms, and Option 3 requires a predicate logic model to verify. Solving this problem poses a higher demand on the *reasoning-in-the-wild* capability of an LLM agent, who needs to recognize the subproblems, identify the formalism, and write the program to finally answer it. Yet, there lacks such a comprehensive benchmark to evaluate LLMs' capability in these aspects.

To move towards bridging this gap, we introduce the task of *reasoning in the wild*. Given a reasoning problem of an *unknown type*, the task is to solve the problem by iteratively identifying the sub-problems and their corresponding formalisms, then writing the suitable programs to

solve each subproblem. To better control and assess the agent's behavior, we introduce the notion of *tactics*. We draw inspiration from the tactic concept in interactive theorem prover literature, where a person proves a math theorem by decomposing and transforming the problem into sub-problems and solving them by corresponding tactics (Lean, 2024; Isabelle, 2024). In our setting, a tactic is a guideline consisting of descriptions, code templates, and an action space, which defines a specific way to tackle the given problem. As we will see in later sections, by explicitly applying the constraint of tactic we obtain a fine-grained insight into LLMs' behavior in problem-solving beyond common holistic evaluation protocols.

To enable such a study, we create a large tactic-guided trajectory dataset, namely ReWild, that consists of problem-solving trajectories (formatted similar to ReAct (Yao et al., 2022) as a chain of Thought, Action, and corresponding Observation) over a wide range of reasoning problems. These problems range from well-defined single-form reasoning such as GSM8K (Cobbe et al., 2021), FOLIO (Han et al., 2022) to ambiguous one such as ReClor (Yu et al., 2020). On top of existing problems, we also create new hybrid problems that require multiple formalisms to solve. ReWild is generated through GPT4 with expert-written prompts and is further post-processed and verified by a combined pipeline of manual annotation and automated fil-

tering. The resulting dataset consists of 6.7K trajectories with a total of 21.7M tokens.

In our experiments, we evaluate a diverse set of the most powerful LLMs to date on our benchmark. Remarkably, we find existing LLMs fail significantly when tasked to solve reasoning problems with tactic-guided programs (see Figure 1, *Prog.*+) and the performance further deteriorates for hybrid problems (see Figure 1, *Hyb*). Through the lens of tactics, **we analyze the results and identify three critical limitations of existing LLMs:** (1) Many LLMs show “overfitted” behavior and fail to follow the tactic on popular problems such as GSM8K, leading to a drop in performance; (2) Most LLMs, except for GPT4 series, show a lack of the capability of “instruction-following in long context”, where it fails to follow the tactic on trajectories that are typically 3K long; (3) Powerful LLMs including GPT4 tend to hallucinate and generate “trivial programs” on ambiguous reasoning problems, showing a poor generalizability over out-of-distribution problems. Finally, we show that these limitations can be alleviated via fine-tuning. We train and release a LLaMA3-8B model on ReWild, which we refer to as **TactIc-Guided ReasonER** (TIGER-8B), and show it achieves GPT4 level performance.

2 Related Work

Program-aided LLMs. Recent research has improved LLMs’ reasoning capabilities with the help of programs, achieving better performance on math (Gao et al., 2023; Kabra et al., 2023; Chen et al., 2023b) and logic (Feng et al., 2023; Pan et al., 2023; Yang et al., 2023; Olausson et al., 2023; Ye et al., 2023) reasoning tasks. These works are “hardwired” to solve a specific type of reasoning problem, and do not explicitly model the interactions with the environment as a trajectory, limiting their in-the-wild applications.

Trajectory-based LLM agents Agent-like LLMs models such as ReAct (Yao et al., 2022) explicitly model the interaction as a trajectory, and several works study its potential in solving reasoning problems. The most prominent ones in this space are FireAct (Chen et al., 2023a), ToRA (Gou et al., 2023), and EURUS (Yuan et al., 2024), which collect problem-solving trajectory data from GPT models in solving various known reasoning tasks and train local models, showing they can achieve better performance after fine-

Tactic

Tactic name: predicate_logic_z3
Problem type and tactic: This tactic builds a formal logical model using predicate logic using z3 lib ...
Code template:

```
import z3
def check_model(solver):
    ...
def main():
    s = z3.Solver()
    <your code>
```

Action Space:
Plan
Input: the problem given
Functionality: give a plan for the question, include a sketch of the solution and libs to be used
Output: text description of the plan, code snippets
Build FOL model
Input: the problem given
Functionality: build the FOL system ...
Output: the main() function with z3 code
Revise code
 ...
Aggregate and answer
 ...

Figure 3: Example of a predicate logic tactic.

tuning.

These approaches highlight the benefits of programs or fine-tuning but do not uncover critical limitations of existing LLMs. Our work underscores several key aspects neglected in previous works. We demonstrate: (1) how to set up a unified reasoning framework to tackle various existing and new (mixed) reasoning problems, (2) how to incorporate various mechanisms (§3) in the reasoning process which will enable a more fine-grained analysis of system’s abilities, and (3) how to evaluate such complex trajectories to gain deeper insight into LLMs’ limitations and behavior beyond common holistic evaluation protocols.

3 Reasoning in the Wild

The task of *reasoning in the wild* considers an LLM agent solving reasoning problems by writing programs and interacting with an environment defined by a *tactic*. Similar to prior agent-like LLM frameworks (Yao et al., 2022), the problem-solving process consists of a series of Thoughts, Actions, and observations. At each step, the LLM agent f_{LLM} generates a thought t that serves as a chain-of-thought (CoT) context for its action, and an action a which carries out the actual action in the environment: this could be outputting a plan in free text (i.e., Plan), generating a program (i.e., Write Program), or returning an answer (i.e., Answer). Then it receives observation o from the environment. The series of $\mathcal{E} = [\langle t, a, o \rangle^{(1)}, \langle t, a, o \rangle^{(2)}, \dots]$ forms a problem-solving *trajectory* \mathcal{E} that will be included as prompt for the next round of interaction.

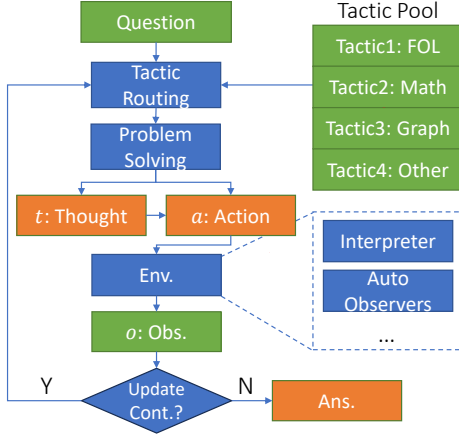


Figure 4: Solving reasoning problems by applying Routing and problem specific tactics.

Different from the prior work, **we extend this problem-solving trajectory by explicitly applying a tactic to specify and monitor the agent’s behavior.** Formally, let $T \in \mathcal{T} = \{T_1, T_2, \dots\}$ be a tactic, and each tactic is defined by $T = \{x_T, \mathcal{A}, O\}$ (Figure 3). Here, x_T is the tactic description, which is a passage describing what this tactic is suitable for and how it is generally used; $\mathcal{A} = \{a_1, a_2, \dots\}$ is the action space defining the allowable actions in this tactic, for example, Plan and Write Program. Each action is defined by its input, functionality, and output $a = \{a_{in}, a_{func}, a_{out}\}$; and O is the tactic-corresponding observer that parses, monitors, executes the agent’s action, and finally provides observations o . A complete tactic is shown in §A.

Let q be a reasoning problem and f_{LLM} be the LLM agent, the agent solves the problem by performing two levels of reasoning: (1) At the top level, a Routing trajectory is formed that picks the subproblems and the corresponding tactics; (2) At the second level, the chosen tactic is carried out, forming a “sub-trajectory” for subproblem solving. As shown in Figure 4:

1. Routing: Given the problem q , the agent executes the Routing tactic, that is T_r , where it extracts the subproblem q_{sub} from q , and identifies the best tactic T to solve it by checking all the tactic descriptions included in T_r and reflecting on the past experience \mathcal{E} (where $\mathcal{E}^{(0)} = \emptyset$). At every (i) -th iteration:

$$t_r^{(i)}, a_r^{(i)} \leftarrow f_{LLM}(q, T_r, \mathcal{E}^{(i-1)}),$$

and $[q_{sub}, T_{sub}]$ are parsed from the output part of the action $a_r^{(i)}$ (i.e., $a_{r,out}^{(i)}$).

2. Problem solving: Solves subproblem q_{sub} with the chosen tactic T_{sub} . At every (j) -th iteration:

$$t^{(j)}, a^{(j)} \leftarrow f_{LLM}(q_{sub}, T_{sub}, \mathcal{E}_{sub}^{(j-1)}),$$

$$o^{(j)} \leftarrow O(a^{(j)}), \mathcal{E}_{sub}^{(j)} = \mathcal{E}_{sub}^{(j-1)} \cup \{\langle t, a, o \rangle^{(j)}\},$$

where sub-trajectory $\mathcal{E}_{sub}^{(0)} = \emptyset$. The sub-trajectory terminates if the Answer action is called where the output part a_{out} is returned as the observation $o_r^{(i)}$ to the Routing trajectory.

3. Update Routing: Update the routing trajectory

$$\mathcal{E}^{(i)} = \mathcal{E}^{(i-1)} \cup \{\langle t, a, o \rangle_r^{(i)}\},$$

and then repeat step 1 until the Answer action is called in the routing trajectory, where the final answer to problem q is aggregated and returned.

By explicitly applying and monitoring the tactics and actions, we (1) make it easy to control the agent’s behavior, defending against potentially undesired behaviors; (2) gain deeper insight into the LLMs, by tracing if it uses the right tactic for the right subproblems; (3) make it possible to create a challenging hybrid problem that requires multiple tactics to solve with fine-grained metrics.

4 Data Generation

To enable evaluating (and training) LLMs on our task, a large dataset of problem-solving trajectories over diverse reasoning problems needs to be created. To do so, we select several existing datasets and generate the trajectories with the specific tactic and ICL prompts and select those successful ones with *non-trivial programs* (introduced shortly) to include in our dataset, ReWild.

Tactics and datasets. We manually create a pool of diverse tactics, each corresponding to the following datasets: (1) **Math Tactic** for GSM8K; (2) **Logic Tactic** for FOLIO, which is a logic-grounded natural language inference (NLI) dataset; (3) **Graph Tactic** for ProScript (Sakaguchi et al., 2021), which is a dataset containing graph-like steps for achieving certain goals such as “opening a bank account”; and (4) **General Program Tactic** for ReClor, which is a commonsense reasoning dataset. To generate trajectories, we manually created examples and included them as ICL prompts to guide the generation using a mixture of three models: gpt4-turbo-0409, gpt-4o, and claude-3-opus. More details in §B.

Post-processing. We filter the generated trajectories with the following steps: (1) We filter those

that do not write any programs or those with programs that failed to run. This indicates the model “shortcuts” the process by directly outputting the answer. (2) A more severe shortcutting happens with ReClor dataset for all the LLMs we tested. We refer to this as “**trivial programs**”: ReClor problems are typically ambiguous in scope and do not fit into any existing formalisms are known to be trained by these LLMs. When LLMs are “forced” to generate programs, they often generate programs that “hardcoded” the answer and put CoT free-form reasoning in the comments. We filter them by labeling a set of programs and using them as ICL prompts of an LLM classifier (Examples and details are provided in §B). In experiments, we show this phenomenon is widely present and causes a significant performance drop.

Hybrid data generation. Apart from the problem-solving trajectories, we also generate hybrid problems and routing trajectories. As an example, let $q_1 = [q_{c,1}, q_{a,1}]$ and $q_2 = [q_{c,2}, q_{a,2}]$ be two problems randomly sampled from the original datasets, where q_c is the “context” part and q_a the “answer” part of the question. We create a multichoice hybrid problem by (1) “blending” the context part into a new coherent passage using an LLM (i.e., gpt4-turbo-0409) $q_c^* = f_{\text{LLM}}(q_{c,1}, q_{c,2})$; and (2) putting each answer $q_{a,i}$ as an *option* to the new hybrid problem with only one of them correct and the rest replaced with incorrect answers. Figure 2 shows a hybrid problem of this kind. We apply a similar pipeline as in program filtering to ensure the blending procedure preserves all the original information.

To further test LLMs on solving hybrid problems of different levels of difficulties. We create hybrid problems with the following configurations: (1) **different numbers and types of options**: let G, F, R denote one option from GSM8K, FOLIO, and ReClor, we create problems of 5 levels of difficulties, that is GG, GF, GFX, GFR, and GFRX. For example, a GG problem is blended from two GSM8K problems with two options; a GFRX problem is blended from four problems with the first three each from G, F and R, and X means an option randomly sampled from any of the preceding datasets (Note that GFX only samples from G and F). We exclude proScript as its answer is a graph and is thus difficult to fit as an option; and (2) **different blending granularities**: for each level of problems, we have half of them blended by only adding limited transition words and relevant con-

text is generally put together, while the other half is blended with their sentences shuffled and interleaved together to make it difficult to extract the relevant context (see an example in Table 9 of §B).

Routing Trajectories. Creating routing trajectories is straightforward with the hybrid problems. Since we know the ground-truth original problem associated with each option in the problem, we create corresponding trajectories with each step “reversing” the blending process by pasting the original problem as the action and the ground-truth answer as the observation. Example routing trajectory and hybrid problems shown in §B.

The final ReWild contains a total of 6.7K trajectories and 21.7M tokens. Statistics and details are available at §B and Table 2.

5 Tactic-Guided Reasoner Fine-Tuning

Trajectories provided by ReWild make it possible to fine-tune local LLMs to perform the reasoning in the wild task. However, training on such trajectories is nontrivial and we discuss two different ways to train a LLaMA3-8B model. We refer to these fine-tuned models as **Tactic-Guided ReasonER (TIGER-8B)**.

Learning from imperfect trajectories. Trajectories collected in ReWild, while solving the problems, do not always have optimal steps. For example, a trajectory may involve writing a bad program in step 1 and correcting it with `Revise Code` action in step 2. In this case, training the model on predicting tokens of step 1 is improper, as it effectively encourages the model to “imitate” a non-optimal step. Theoretically, a principled solution is to bring in the reinforcement learning technique, as it resembles off-policy learning. Here, we introduce a simple yet effective approach to the issue while still enjoying the low variance and computational cost of a standard auto-regressive objective.

Our approach takes advantage of the fact that “a bad step is easy to detect in hindsight”. Consider a trajectory with one bad step $[\langle t, a, o \rangle_{c1}, \langle t, a, o \rangle_{w2}, \langle t, a, o \rangle_{c3}]$. Since the LLMs can perform the same action multiple times and some actions are dependent on other actions (e.g., `Revise Code` can only happen after `Write Program`), we can easily detect non-optimal steps with rule-based labeling. For example, if `Revise Code` is performed then any programs produced before it are incorrect; and if `Answer` is performed then any `Answer` before this action is incorrect. Once they are recognized, we experiment with

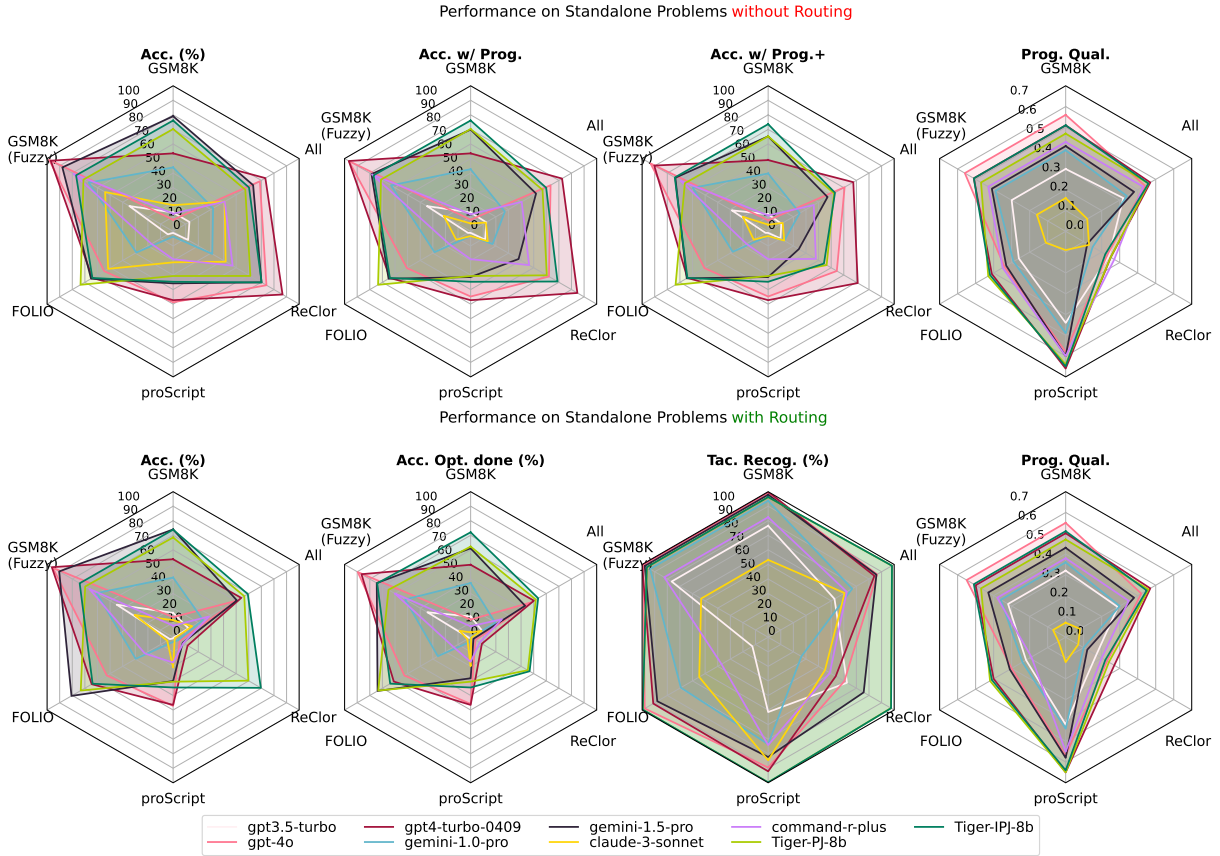


Figure 5: Results of the standalone problems. Exact scores provided in §D.

two different approaches: (1) imperfect trajectory training (**IPJ**): we mask out the label of t and a of incorrect steps but still train on the original trajectory, in which case the trainable tokens are $[\langle t, a, o \rangle_{c1}, o_{w2}, \langle t, a, o \rangle_{c3}]$; (2) perfect trajectory training (**PJ**): we recreate a perfect trajectory by replacing the incorrect ones with subsequent correct ones, in which case a new trajectory $[\langle t, a, o \rangle_{c1}, \langle t_{w2}, a_{c3}, o_{c3} \rangle]$ is created and trained (assuming a_{w2}, a_{c3} are same type). Training the model with PJ and IPJ approach is straightforward and we leave details in §C.

6 Experiments

Settings. We evaluate seven most powerful API LLMs to date listed in Figure 5 (excluding claude-3-opus due to its high API cost), and two of our models TIGER-PJ-8B and TIGER-IPJ-8B on the standalone and hybrid problems of ReWild. All API LLMs are given up to five trajectories as ICL prompts (detailed settings in §D).

6.1 Standalone problems

We first evaluate models on standalone problems that require only one formalism to solve. Specifi-

cally, we evaluate them in two settings: (1) **Without routing**: model is given the ground-truth problem-solving tactic, and tasked only to solve the standalone problem, and (2) **with routing**: model is tasked to first find the right tactic and then solve the problem. This is effectively a special case of hybrid problems with only one option. This can be seen as an easy hybrid benchmark that sits between the two types of problems.

We evaluate the model with the following metrics: (1) accuracy (**Acc**): the standard accuracy of models solving the problems; (2) Acc with program (**Acc w/ Prog**): a more strict notion of the **Acc** where trajectories that do not produce a program are considered as failed. This score reduces if the model fails to follow instructions to write the program before answering; (3) program quality (**Prog Qual**): recall the “trivial program” issue in §4, LLMs can generate a trivial program to shortcut the process. To quantitatively measure this behavior, we measure the difference between the generated program and the ground-truth one using CodeBleu (Ren et al., 2020), a BLEU score designed for code; (4) Acc w/ non-trivial program (**Acc w/ Prog+**): is a more strict version of **Acc w/**

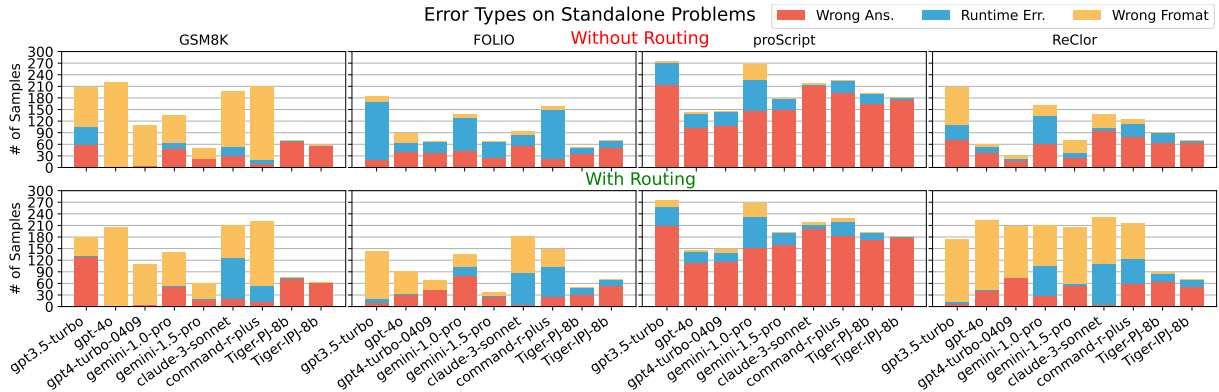


Figure 6: Error types of the standalone problems.

Prog where the trajectories with programs CodeBleu scores lower than a threshold ($= 0.15$) are considered as failed. This metric decreases if a model shortcuts the process and generates trivial programs; (5) **tactic recognition (Tac Recog)**: the accuracy of the model correctly recognizes the right tactic for the problem (with routing only); (6) **Acc w/ options done (Acc Opt done)**: the most strict accuracy where only trajectories that (a) correctly recognized the tactics for all options and (b) generated a program with CodeBleu higher than the threshold for all options, are considered correct. The model having the highest **Acc Opt done** is the most desired one of this task (with routing only).

LLMs overfit on popular problems. Figure 5 shows the results with and without the routing. One major issue we find is that many LLMs while performing well on GSM8K in ICL-CoT, fail significantly in our benchmark, where gpt-4o achieves only 7.95% on Acc and gpt4-turbo-0409 achieves only 53.56%. Upon further investigation, we find LLMs tend to ignore our instructions and ICL examples on only returning the numerical answer and proceed to return answers with explanations. To validate our observation, we categorize the errors into three types and summarize the per-dataset counts in Figure 6. Here, **Wrong Ans** means a valid answer is returned but is incorrect; **Runtime Err** means Answer action is never called; and **Wrong Format** means Answer is called but the format is invalid, e.g., included irrelevant context and explanations. Figure 6 shows that all LLMs have exceptionally high **Wrong Format** rates on GSM8K in both benchmarks; such a phenomenon does not exist for other datasets, precluding the possibility of bad prompting. Given that the GSM8K dataset

comes with CoT ground-truth answers, we conjecture that **most of the LLMs are trained on these CoT answers which leads to an overfitted behavior in answering**. Examples shown in §D.

Nevertheless, we implement a fuzzy match pipeline that considers the answer correct as long as the ground-truth numerical answer exists in the answer output. The fuzzy match results are shown under **GSM8K (fuzzy)**, which aligns much better with that of ICL-CoT, further confirming our observation. For the hybrid evaluation in §6.2, we use only the fuzzy match on GSM8K options.

Many LLMs lack instruction-following capability in long context. We find many LLMs struggle to follow the instructions in a long context. The most prominent case is with claude-3-sonnet, whose overall Acc decreases from 40.83% to 12.09% on Acc w/ Prog, indicating the model very frequently skips writing programs before answering. Other models generally see a decrease of 5%-10%, except for gpt4-turbo-0409, which is less than 1%.

Trivial programs and hallucination. Figure 5 sees a general decrease of 10%-45% on ReClor problems with Acc w/ Prog+, with the largest drop being 45.73% from gemini-1.5-pro. On top of this, the Acc Opt done scores drop to near zero for with-routing benchmarks, indicating a high error rate in recognizing the tactic for ReClor problems. This confirms our observation in §4 that **existing LLMs generalize poorly to rarely trained coding problems**. This observation is further validated in Prog Qual chart, where the CodeBleu scores of ReClor are the lowest.

Potential training data imbalance. Looking at Figure 6, we find a high **Runtime Err** rate with FOLIO, which primarily relies on the z3 library for building the logic model. We find this is

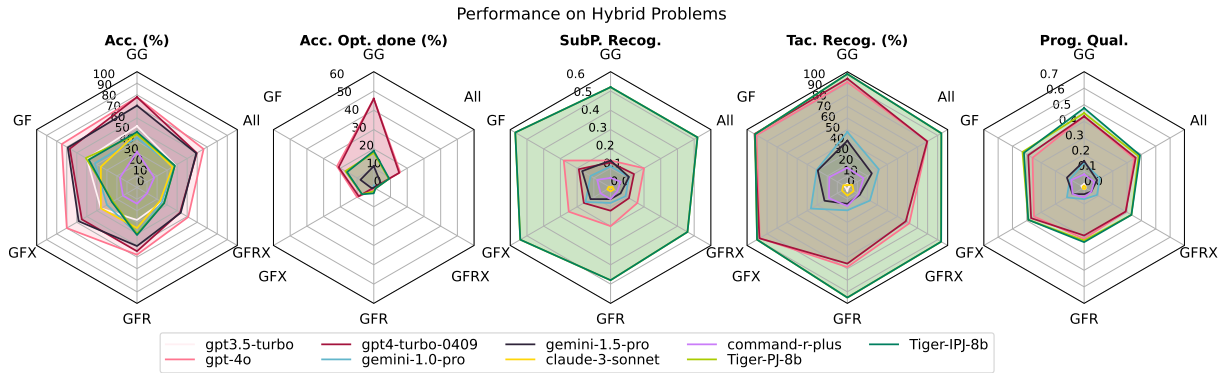


Figure 7: Results of the hybrid problems. Exact scores provided in §D.

Model	Blend easy			Blend hard		
	Acc. Opt. done (%)	SubP. Recog.	Tac. Recog.	Acc. Opt. done (%)	SubP. Recog.	Tac. Recog.
gpt-4o	16.60	0.22	81.40	14.94	0.19	78.18
gpt4-turbo-0409	15.83	0.15	80.69	14.52	0.14	78.46
gemini-1.5-pro	4.53	0.12	26.1	3.93	0.1	22.63
Tiger-PJ-8b	8.94	0.52	93.02	7.76	0.5	94.43
Tiger-IPJ-8b	8.88	0.52	93.02	9.54	0.5	4.43

Table 1: Results of hybrid problems with different blending strategies (zero-scored model removed).

because models frequently write syntactically incorrect code leading to runtime errors and cannot correct it despite multiple `Revise Code` attempts. This suggests that the training data likely leans toward popular libraries and the models struggle to generalize to rare libraries.

Fine-tuning alleviates above issues. TIGER-8B generally performs better for these issues, with the IPJ version performing slightly better than the PJ one. TIGER-8B has more consistent Acc scores for all metrics and maintains the highest score on ReClor in the with-routing setting. TIGER-8B’s overall performance is similar to GPT4 models, and we believe it would further improve if given more data and trained with larger models.

6.2 Hybrid problems

We evaluate LLMs on hybrid problems using a similar set of metrics: note that **Acc Opt done** here counts the trajectories that answer correctly the hybrid problem with all its options correctly recognized and solved with high-quality programs. We also include the subproblem recognition metric (**SubP Recog**) measuring the BLEU score between the extracted subproblem text and the original problem text.

Hybrid problems are highly challenging. We summarize the results in Figure 7. All models maintain an Acc score above 35% on hybrid problems except for `command-r-plus`. We observe a

consistent drop in scores of all models when moving to problems of higher difficulties, indicating that difficulty levels are effective. Remarkably, the Acc Opt done score drops significantly, where most LLMs drop to near zero, with only GPT series and TIGER-8B maintaining an overall score of around 10%. Furthermore, most of the scores are earned on easy ones such as GG, and all API LLMs could not even successfully answer a single problem from GFR and GFRX. Thanks to fine-tuning, TIGER-8B maintained the highest SubP and Tac Recog scores and the best Prog Qual, and managed to solve a few more problems in hard difficulties. We conjecture the task has gone beyond the capacity of an 8B model and we might see a bigger gap by fine-tuning larger models. Comparing all LLMs, GPT4 series has a clear advantage, still, their capabilities are far from obtaining a reasonable score in this benchmark. In summary, **we believe the hybrid problems remain a highly challenging and valuable benchmark that provides deep insights into LLMs’ reasoning capability.**

Blending strategies. We summarize the results on different blending strategies in Table 1. We observe a drop in the Acc and SubP Recog moving from easy to hard for many LLMs, but they are not overall very sensitive to different blending strategies. This suggests that LLMs are generally strong at retrieval in short context. We leave the investigation in long context in future work.

7 Conclusion

We present *reasoning in the wild*, a task that challenges LLMs’ reasoning capabilities in solving ambiguous and mixed-in-scope problems, where we create dataset ReWild. Our experiments reveal critical issues of the existing LLMs and show that they can be alleviated with fine-tuning, that is TIGER-8B.

Acknowledgements

This work was supported by a sponsored research award by Cisco Research.

8 Limitations

During the creation of ReWild, we used a combined approach of manual labeling and auto filtering. It is likely the trajectories, especially for ReClor data, still contain “trivial programs”.

During evaluation, we use two different ICL prompts for problem-solving trajectories to balance between cost and performance. While the total length of the message can go to 10K long, it does not exhaust all the context window length. That said, it is likely by providing more fully detailed trajectories, the API LLMs can achieve better performance than those reported in the paper.

9 Ethics Statement

Potential negative impact. The trajectory data contain fully the LLM-generated contents, and the code and text could potentially contain misinformation. It is also possible, while unlikely that running the programs in the trajectory could cause unexpected behaviors in the host machine due to different environment configurations.

Artifact statements. We release data, code, and weights under Apache 2.0 license and they are intended for research use only. Additionally, the usage of the TIGER-8B model should follow the license agreement of LLaMA and Alpaca. ReWild is released under CC BY NC 4.0 and the use of such dataset should follow the [policy of OpenAI](#).

References

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023b. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. ArXiv:2211.12588 [cs].

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

Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi Sharma, Yelong Shen, Dongyan Zhao, and Weizhu

Chen. 2023. [Language Models can be Logical Solvers](#). ArXiv:2311.06158 [cs].

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [PAL: Program-aided Language Models](#). ArXiv:2211.10435 [cs].

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujie Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. 2022. [Folio: Natural language reasoning with first-order logic](#). *arXiv preprint arXiv:2209.00840*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. [ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings](#). ArXiv:2305.11554 [cs].

Isabelle. 2024. [Isabelle overview](#).

Anubha Kabra, Sanketh Rangreji, Yash Mathur, Aman Madaan, Emmy Liu, and Graham Neubig. 2023. [Program-Aided Reasoners \(better\) Know What They Know](#). ArXiv:2311.09553 [cs].

FRO Lean. 2024. [Lean manual](#).

Theo X. Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. 2023. [LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers](#). ArXiv:2310.15164 [cs].

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. [Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning](#). ArXiv:2305.12295 [cs].

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Keisuke Sakaguchi, Chandra Bhagavatula, Ronan Le Bras, Niket Tandon, Peter Clark, and Yejin Choi. 2021. proscript: Partially ordered scripts generation via pre-trained language models. *arXiv preprint arXiv:2104.08251*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language Models Can Teach Themselves to Use Tools](#). ArXiv:2302.04761 [cs].

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2023. [Cognitive architectures for language agents](#). *CoRR*, abs/2309.02427.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2023. [Harnessing the power of large language models for natural language to first-order logic translation](#).

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. [React: Synergizing reasoning and acting in language models](#). *arXiv preprint arXiv:2210.03629*.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. [SATLM: Satisfiability-Aided Language Models Using Declarative Prompting](#).

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. [Reclor: A reading comprehension dataset requiring logical reasoning](#). In *International Conference on Learning Representations (ICLR)*.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024. [Advancing LLM Reasoning Generalists with Preference Trees](#). ArXiv:2404.02078 [cs].

A Reasoning in the Wild examples

Tactic description examples. An example of a complete tactic description is shown in Table 7.

B Data generation

Tactic Pool. We manually create a pool of diverse tactics for different types of reasoning including: (1) **Math Tactic**, where the agent builds a math model to solve problems involving math; (2) **Logic Tactic**, where the agent builds a logic model using the Python Z3 library to model and verify logical statements, which can be used for logical deduction, induction, and abduction problems; (3) **Graph Tactic**, where the agent builds a graph model using the Python NetworkX library to model a graph structure, which can be used for problems involving sorting items as a graph; (4) **General Program Tactic**, where the agent writes a general program using all libraries above to represent problems that do not fit into any of the formalisms above, such as commonsense reasoning; and (5) **Routing Tactic**, which is the main tactic,

which the agent follows to decompose the problem into subproblems and identify the best tactic for solving it.

Datasets. We select the following datasets each corresponding to our designed tactics above for trajectory generation: (1) GSM8K (Cobbe et al., 2021): a popular arithmetic math dataset; (2) FOLIO (Han et al., 2022): a logic-grounded natural language inference (NLI) dataset; (3) ProScript (Sakaguchi et al., 2021): a dataset containing graph-structured natural language steps for achieving certain goals such as “opening a bank account”; and (4) ReClor (Yu et al., 2020): a commonsense reasoning dataset curated from LSAT and GMAT tests in the form of multichoice problems.

Trajectory collection process. To generate trajectories that adhere to our setup, we first manually created trajectories for each dataset, then include them as ICL prompts to guide the generation. We use a mixture of three models in the process: gpt4-turbo-0409, gpt-4o, and claude-3-opus. In particular, we find claude-3-opus performs the best in zero-shot and out-of-distribution problems such as ReClor, and GPT4 series have the best instruction-following capability during generation. Based on our initial experiments, we set the maximum number of steps of each trajectory to 7 to balance between the final success rate and the cost. A trajectory is terminated if (1) the Answer action is called and the result is correct; (2) the max steps are reached; (3) the model makes 3 consecutive errors, which is typically a bad sign of a failed trajectory.

Post-processing. After generation, we filter the trajectories with the following steps: (1) We filter those that do not write any programs or those with programs that failed to run. This indicates the model “shortcuts” the process by directly outputting the answer. (2) A more severe shortcutting happens with ReClor dataset for virtually all the LLMs we tested. We refer to this as “**trivial programs**”: ReClor problems are typically ambiguous in scope and do not fit into any existing formalisms are known to be trained by these LLMs, such as math and formal logic. When LLMs are “forced” to generate programs for them, even with multiple ICL prompts given, they very often generate programs that either (a) are a fancy CoT, where they put CoT free-form reasoning in the comments and have programs directly return the answer; (b) or have the answer “hardcoded”

Subset	# Train	# Test	Avg. steps	Avg. Tokens
GSM8K	1.2K	239	4.2	2.1K
FOLIO	0.9K	192	4.3	4.6K
proScript	1.1K	277	5	3.4K
ReClor	1.1K	235	5.2	4.3K
Hybrid	1K	500	4	2K
All	5.3K	1.4K	4.5	3.2K

Table 2: ReWild dataset statistics.

as constants. Examples are shown in Table 8. We filter these trajectories by first manually labeling a subset of good and bad programs, and use LLMs to filter the rest using them as ICL prompts. To further boost the filtering accuracy, we use a majority vote of three LLMs: gpt4-turbo-0409, claude-3-opus, and gemini-1.5-pro. On our manually labeled test set, this framework achieves an accuracy of 72% with 16% being false positives (i.e., the bad programs inferred as good ones). We measure Krippendorff’s alpha, that is the inter-model agreement rate, which is 0.69, indicating the annotations are generally aligned. In experiments, we show that this phenomenon is widely present in all LLMs and causes a significant performance drop in all ReClor-related problems.

Trivial program examples. Trivial program examples shown in Table 8.

Hybrid problem examples. Hybrid problem examples shown in Table 9.

Routing trajectory examples. Routing trajectory examples shown in Table 10.

Dataset statistics. We show statistics of ReWild in Table 2. Due to budget limits, we did not generate trajectories for all the problems. We plan to release the generation pipeline for researchers interested in continuing the process. For each dataset and the hybrid one, we aim for 1K for training and 250 for testing (FOLIO has fewer total testing samples). ReWild contains a total of 6.7K trajectories and 21.7M tokens. In experiments, we find it supports a sufficient fine-tuning of an LLaMA3-8B model and evaluation of existing LLMs.

C Tactic-Guided Reasoner Fine-Tuning

In §5, we introduce two ways to prepare the trajectory data: perfect trajectory training (PJ) and imperfect trajectory training (IPJ). PJ training is straightforward as it resembles standard imitation learning and is applied in prior work such as ReAct (Yao et al., 2022) and ToRA (Gou et al., 2023).

We find IPJ training could further improve the model’s performance because it trains the model to correct the previous steps, making it more robust during inference time. In experiments, we fine-tune two versions of LLaMA3-8B: (1) TIGER-PJ-8B which trains on PJ data; (2) TIGER-IPJ-8B which trains on PJ and IPJ data. Both models are trained with LoRA $r = 64, \alpha = 64$. Additionally, we train the routing model by continuing training on the TIGER-PJ-8B on routing trajectories with LoRA $r = 16, \alpha = 16$. All training is performed for 3 epochs on a single A100 GPU.

D Experiments

Hyperparameters. For all experiments, We set the max steps allowed in a trajectory to 7, and the max consecutive errors to 3. For routing trajectory, all API LLMs are given two full routing trajectories as ICL prompts. For problem-solving trajectory, all API LLMs are provided with 5-shot ICL prompts consisting of the first two steps (typically, Plan and Write Program) to a problem. This prompt is given for the first two steps of the in-progress trajectory, then we swap it into ICL prompts of two problem-solving trajectories of the same problem type with their main output contents omitted. We do this because we find the most difficult part of a trajectory is the first few steps, where the model figures out the right overall approach, so we insert more detailed solutions. And for the rest, we insert a complete trajectory to guide the model with formatting. This helps to balance between the performance and cost as one trajectory ICL example is 3K long.

For CodeBleu metric (Ren et al., 2020), we set the weights of ngram match, weighted ngram match, syntax match, dataflow match to 0.15, 0.15, 0.35, 0.35 to favor programs with similar functionality rather than textual appearance;

GSM8K overfitted answers. Example overfitted outputs of GSM8K shown in Table 3.

Full results. We show the numerical results in Table 4, Table 5, and Table 6.

gpt4-turbo-0409 Output		gemini-1.5-pro Output	
out_content: The rainfall on Tuesday is 5 inches.	gt answer: 5	out_content: Dorothy has 4 Facebook friends. James has 16 Facebook friends.	gt answer: 16
out_content: The monthly earnings of the dance studio are a net loss of \$120.00.	gt answer: 480	out_content: 5 ===	gt answer: 12
out_content: James made a total commission of \$17,500 from selling 10 cars.	gt answer: 17500	out_content: 42.0 ===	gt answer: 1248
claude-3-sonnet Output		command-r-plus Output	
out_content: The number of pieces of candy that are not chocolate is 49.	gt answer: 20	out_content: The answer is **40**.	gt answer: 140
out_content: The total dollar amount in a stack containing two thirds of the 9,300 pennies is \$62.00.	gt answer: 62	out_content: The answer to the question is **180000 meters**.	gt answer: 180000
out_content: The total amount Molly will pay for catering the party is \$131.	gt answer: 101	out_content: There are 12 teachers at Dr. Wertz's school.	gt answer: 36

Table 3: Example of overfitted outputs on GSM8K problems.

Model	GSM8K				GSM8K Soft				FOLIO			
	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.
gpt3.5-turbo	12.97	12.97	11.72	0.3	34.73	34.73	28.87	0.3	4.17	2.08	2.08	0.26
gpt-4o	7.95	7.95	7.53	0.56	98.33	96.65	93.31	0.56	54.69	50.52	50.52	0.34
gpt4-turbo-0409	53.56	53.56	48.95	0.51	97.07	96.65	90.38	0.51	64.06	64.06	64.06	0.41
gemini-1.0-pro	43.93	42.68	39.75	0.39	66.95	63.18	59.83	0.39	29.17	28.65	28.12	0.29
gemini-1.5-pro	79.08	69.87	65.27	0.41	87.87	78.66	73.64	0.41	65.1	65.1	64.58	0.33
claude-3-sonnet	17.99	5.44	4.6	0.16	53.97	21.34	20.08	0.16	51.56	11.46	11.46	0.11
command-r-plus	12.13	11.72	10.88	0.43	79.08	78.24	69.46	0.43	17.19	17.19	17.19	0.35
Tiger-PJ-8b	70.29	70.29	65.27	0.47	71.13	71.13	66.11	0.47	73.44	73.44	73.44	0.43
Tiger-IPJ-8b	76.15	76.15	73.64	0.51	76.99	76.99	74.06	0.51	64.06	64.06	64.06	0.42
Model	proScript				ReClor				All			
	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.
gpt3.5-turbo	1.08	1.08	1.08	0.44	11.91	11.49	9.79	0.24	12.94	12.41	10.5	0.32
gpt-4o	48.74	44.77	44.77	0.58	74.04	62.13	54.47	0.23	68.82	63.41	60.66	0.44
gpt4-turbo-0409	47.29	47.29	47.29	0.66	86.81	84.68	71.06	0.25	73.17	72.53	67.55	0.47
gemini-1.0-pro	3.61	3.25	3.25	0.49	31.06	17.45	12.77	0.15	31.71	27.15	25.03	0.34
gemini-1.5-pro	35.74	31.41	31.41	0.6	70.09	38.03	24.36	0.12	63.48	51.91	47.13	0.38
claude-3-sonnet	21.3	3.25	3.25	0.09	41.7	13.62	12.77	0.13	40.83	12.09	11.56	0.12
command-r-plus	19.13	19.13	19.13	0.6	47.23	46.38	37.87	0.27	40.93	40.51	36.16	0.43
Tiger-PJ-8b	30.69	30.69	30.69	0.64	61.28	60.43	46.38	0.25	57.26	57.05	52.28	0.46
Tiger-IPJ-8b	34.66	34.66	34.66	0.65	70.21	68.94	44.26	0.22	60.23	59.92	53.02	0.46

Table 4: Results of standalone problems without routing.

Model	GSM8K				GSM8K Soft				FOLIO			
	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.
gpt3.5-turbo	16.74	14.23	76.57	0.32	44.77	34.31	76.57	0.32	2.6	1.04	12.5	0.22
gpt-4o	13.81	12.13	99.16	0.55	95.4	89.54	99.16	0.55	52.6	52.6	97.92	0.3
gpt4-turbo-0409	53.56	49.79	100	0.5	96.23	86.61	100	0.5	64.58	60.94	91.15	0.4
gemini-1.0-pro	41	37.24	93.72	0.36	60.67	51.88	93.72	0.36	29.69	26.04	69.27	0.24
gemini-1.5-pro	74.06	61.09	97.91	0.43	90.38	74.48	97.91	0.43	80.63	73.82	87.96	0.31
claude-3-sonnet	11.72	3.35	53.14	0.07	30.54	8.79	53.14	0.07	4.69	2.6	54.69	0.04
command-r-plus	7.95	7.53	82.43	0.38	66.53	60.25	82.43	0.38	22.4	13.54	36.98	0.19
Tiger-PJ-8b	68.55	62.28	96.2	0.47	69.76	65.19	96.2	0.47	73.29	73.29	99.5	0.42
Tiger-IPJ-8b	73.9	72.11	96.3	0.51	74.13	74.12	96.3	0.51	63.89	63.84	99.5	0.41

Model	proScript				ReClor				All			
	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.	Acc (%)	Acc w/ Prog.	Acc w/ Prog+	Prog. Qual.
gpt3.5-turbo	0.8	0.5	51.4	0.42	0.43	0	62.55	0.17	12.22	9.05	52.64	0.29
gpt-4o	47.2	45.8	89.52	0.57	5.11	3.83	60.43	0.23	50.03	47.81	86.42	0.42
gpt4-turbo-0409	46.5	46.5	92.12	0.65	11.49	8.94	53.62	0.27	54.06	50.24	84.32	0.47
gemini-1.0-pro	3.21	3.04	73.49	0.44	4.09	0	26.36	0.1	23.7	19.66	66.66	0.3
gemini-1.5-pro	30.1	28.21	82.31	0.58	7.62	2.24	75.78	0.12	50.58	43.24	85.91	0.38
claude-3-sonnet	20.8	19.9	84.56	0.12	1.28	0.43	45.11	0.07	15.12	8.71	60.68	0.08
command-r-plus	17.4	16.9	74.12	0.55	8.09	7.23	48.94	0.2	28.55	24.79	62.39	0.35
Tiger-PJ-8b	30.54	30.54	99.8	0.65	59.71	45.37	97.4	0.24	56.45	51.72	98.23	0.46
Tiger-IPJ-8b	34.53	34.53	99.7	0.64	69.57	46.77	97.5	0.22	59.24	53.58	98.25	0.45

Table 5: Results of standalone problems with routing.

Model	Acc (%)	Acc. Opt done (%)	gg			Acc (%)	Acc. Opt done (%)	gf			Acc (%)	Acc. Opt done (%)	gfx		
			SubP. Recog.	Tac. Recog.	Prog. Qual.			SubP. Recog.	Tac. Recog.	Prog. Qual.			SubP. Recog.	Tac. Recog.	Prog. Qual.
gpt3.5-turbo	53	0	0	3	0.02	50	0	0.01	3.5	0.02	36	0	0	1.33	0.01
gpt-4o	78	46	0.14	91	0.43	75	22	0.28	90.5	0.36	70	10	0.25	87.33	0.35
gpt4-turbo-0409	78	46	0.13	94	0.43	67	21	0.19	92.5	0.39	55	9	0.16	87.67	0.37
gemini-1.0-pro	43	1	0.11	48	0.15	42	0	0.12	30.5	0.1	30	0	0.15	36.33	0.12
gemini-1.5-pro	71	11	0.14	40.4	0.16	69	8	0.17	29.5	0.12	58	1	0.12	22.79	0.08
claude-3-sonnet	47	0	0	2	0.01	36	0	0.01	4	0.01	36	0	0.02	5.33	0.01
command-r-plus	31	0	0.05	17	0.08	17	0	0.08	18	0.09	14	0	0.06	18.33	0.08
Tiger-PJ-8b	46	18	0.52	98	0.45	51	17	0.57	92	0.43	24	7	0.54	90	0.39
Tiger-IPJ-8b	48	19	0.52	98	0.48	48	16	0.57	92	0.42	25	7	0.54	90	0.39

Model	Acc (%)	Acc. Opt done (%)	gfr			Acc (%)	Acc. Opt done (%)	gfxr			Acc (%)	Acc. Opt done (%)	all		
			SubP. Recog.	Tac. Recog.	Prog. Qual.			SubP. Recog.	Tac. Recog.	Prog. Qual.			SubP. Recog.	Tac. Recog.	Prog. Qual.
gpt3.5-turbo	28	0	0	3.33	0.01	25	0	0	0.75	0.01	38.4	0	0.01	2.38	0.01
gpt-4o	58	1	0.2	68.67	0.31	51	0	0.16	61.75	0.29	66.4	15.8	0.2	79.85	0.35
gpt4-turbo-0409	55	0	0.12	65.67	0.29	43	0	0.11	58.25	0.29	59.6	15.2	0.14	79.62	0.36
gemini-1.0-pro	40	0	0.08	19.67	0.07	25	0	0.09	22.5	0.09	36	0.2	0.11	31.4	0.11
gemini-1.5-pro	51	0	0.06	14.09	0.04	44	0	0.06	12.5	0.05	59.11	4	0	24.42	0.09
claude-3-sonnet	35	0	0.02	7	0.01	23	0	0.02	6.25	0.01	35.4	0	0.01	4.92	0.01
command-r-plus	14	0	0.06	16	0.06	10	0	0.04	11.25	0.05	17.2	0	0.06	16.12	0.07
Tiger-PJ-8b	40	2	0.48	95	0.32	27	1	0.46	93.5	0.33	37.6	9	0.52	93.7	0.38
Tiger-IPJ-8b	41	3	0.48	95	0.33	27	1	0.46	93.5	0.33	37.8	9.2	0.52	93.7	0.39

Table 6: Results of hybrid problems.

Predicate Logic Tactic

Tactic name
predicate_logic_z3

Problem type and tactic

This tactic builds a formal logical model using predicate logic formalism with the help of python z3 lib. This tactic is suitable for solving reasoning problems that involves deductive, inductive or, abductive reasoning. To do so, the tactic will represent the problem as a self-contained first-order logic (FOL) system that consists of Constants, Predicates, Logic Variables, Quantifiers, Functions, Logic Operators, Grounded Facts, Logic Formulas and so on; then it will seek to perform formal reasoning with the help with z3 lib.

Typical use cases

The tactic is suitable for problems that can be represented by an FOL system and solved by performing the following three types of formal reasoning

- Deductive reasoning: Given Facts and Logic Formulas, deduce new Facts from the system by applying the Formulas to the Facts.
- Inductive reasoning: Given Facts and Potentially some Formulas, induce new Formulas that entail the given Facts and are consistent with the preexisting Formulas.
- Abductive reasoning: Given Facts, Logic Formulas, and a consequence Fact, infer the missing Facts or Formulas, such that the consequence Fact can be entailed by the system.

Model and tactic outputs

- Model: To apply the tactic, one builds a self-contained FOL system that fully represent the problem using z3 lib
- Outputs: the z3 code should output either 'Agree', 'Contradict', or 'Uncertain'.
'Agree' means the Facts or Formulas agree with the system
'Contradict' means the Facts or Formulas contradict with the system
'Uncertain' means the Facts or Formulas contradict with the system

Note that the type of reasoning and the system built for the problem determine:

- How the output is interpreted.
 - Whether the output serves as the final answer or intermediate checks for the problem-specific answer
- For example: for a deductive reasoning task with a given hypothesis, one builds the system to determine if the hypothesis Agree/Contradict/Uncertain to the system; for a deductive reasoning task where one wants to deduce all possible Facts, then one should infer all Facts that Agree with the system; for inductive reasoning, one infers the Formulas that Agree with the system; for abductive reasoning, one infers the Facts or Formulas that Agree with the consequence and the system.

Tactic details

You will use the following python libs to solve the problem:
Any builtin Python libs
z3

Code template

You will use the following code template to solve the problem.

```
```python
import z3
from z3 import *

def check_model(solver):
 res = solver.check()
 if res == sat:
```

```

return 'sat'
elif res == unsat:
return 'unsat'
else:
return 'unsolvable'

def check_constraint(solver, c):
pos_res = solver.check(c)
neg_res = solver.check(Not(c))

if (pos_res == sat) and (neg_res == unsat):
return 'Agree'
elif (pos_res == unsat) and (neg_res == sat):
return 'Contradict'
elif (pos_res == unknown) or (neg_res == unknown):
return 'unsolvable'
else:
return 'Uncertain'

```

```

def main():
s = z3.Solver()
<your code>
"""

```

**\*\*Action space\*\***

You will use and **ONLY** use the following actions to solve the problem.  
You can apply actions in arbitrary order and arbitrary number of times.

**#A# Plan**

- Input: the problem given
- Functionality: give a plan on how to solve the question, including a sketch of the solution, libs to be used, and code snippets
- Output: text description of the plan and potential code snippets of the form

```

"""python
<your code>
"""

```

**#A# Build FOL model**

- Input: the original problem given
- Functionality: build the FOL system that represents the problem; use `check_constraint` or `check_model` to produce output
- Output: the `main()` function with z3 code of the FOL system of the form

```

"""python
def main():
<your code>
"""

```

**#A# Revise code**

- Input: z3 code built so far, with potential feedbacks from observations or users
- Functionality: reflect on the Input, specify if the tactic is good so far, and if not what are the issues; then, revise the code to continue the problem-solving process or address the issues.
- Output: the `main()` function with revised z3 code of the form

```

"""python
def main():
<your code>
"""

```

#A# Aggregate and answer

- Input: all z3 code, revisions, and observations so far
- Functionality: aggregate and summarize the outputs produced so far, and provide the problem-specific final answer
- Output: the problem-specific answer

#A# Tactic check

- Input: the original problem, all z3 code, revisions, and observations so far
  - Functionality: analyze the Input, determine if the tactic can solve the problem or not
  - Output: "Tactic Good" if tactic can solve the problem; "Tactic Bad" if tactic cannot solve the problem.
- 

Table 7: Tactic description of predicate logic.

---

=== Question and answer

Answer the question below by choosing the correct statement.

### Context:

In a study, pairs of trained dogs were placed side by side and given a command such as "sit. " After both obeyed the command, one dog was given a treat while its partner was given no reward at all. Over time, the dogs who went unrewarded began to disobey the command. This shows that dogs have an aversion to being treated unfairly.

### Question:

Which one of the following would be most useful to know in order to evaluate the argument?

### Statements:

1. Were dogs who were accustomed to receiving regular rewards prior to the study more inclined to obey the command?
2. How many repetitions were required before the unrewarded dogs began to disobey the command?
3. Is there a decline in obedience if rewards are withheld from both dogs in the pair?
4. Were dogs who received treats in one trial ever used as dogs that did not receive treats in other trials?

Answer: 3

=== Proposed program

```
“python
class DogBehavior:
def __init__(self, accustomed_reward, repetitions_until_disobey, responses_without_rewards, consistency_of_treatment):
self.accustomed_reward = accustomed_reward
self.repetitions_until_disobey = repetitions_until_disobey
self.responses_without_rewards = responses_without_rewards
self.consistency_of_treatment = consistency_of_treatment
def evaluate_statement_based_on_behavior(self, statement_number):
if statement_number == 1:
return self.accustomed_reward
elif statement_number == 2:
return self.repetitions_until_disobey
elif statement_number == 3:
return self.responses_without_rewards
```



```

elif statement_number == 4:
 return self.consistency_of_treatment
accustomed_reward = True
repetitions_until_disobey = 10
responses_without_rewards = False
consistency_of_treatment = False
dog_behavior = DogBehavior(accustomed_reward,
repetitions_until_disobey, responses_without_rewards, consistency_of_treatment)
outcome_1 = dog_behavior.evaluate_statement_based_on_behavior(1)
outcome_2 = dog_behavior.evaluate_statement_based_on_behavior(2)
outcome_3 = dog_behavior.evaluate_statement_based_on_behavior(3)
outcome_4 = dog_behavior.evaluate_statement_based_on_behavior(4)
print(f"Evaluation of statement 1 leads to: {outcome_1}")
print(f"Evaluation of statement 2 leads to: {outcome_2}")
print(f"Evaluation of statement 3 leads to: {outcome_3}")
print(f"Evaluation of statement 4 leads to: {outcome_4}")
"""

```

This code doesn't actually solve the problem and is intended for illustration only. It is meant to proceed with the tactic and show how we could evaluate the significance of statements if we had a more concrete model and data.

=== outputs

```

Observer: python interpreter
Feedback status: feedback ok
Content:
stdout:
Evaluation of statement 1 leads to: True
Evaluation of statement 2 leads to: 10
Evaluation of statement 3 leads to: False
Evaluation of statement 4 leads to: False

```

### Comments

This is a trivial program that assigns answers to the statements without representing them in details

### Program good

N

---

=== Question and answer

Answer the question below by choosing the correct statement.

### Context:

Light is registered in the retina when photons hit molecules of the pigment rhodopsin and change the molecules' shape. Even when they have not been struck by photons of light, rhodopsin molecules sometimes change shape because of normal molecular motion, thereby introducing error into the visual system. The amount of this molecular motion is directly proportional to the temperature of the retina.

### Question:

Which one of the following conclusions is most strongly supported by the information above?

### Statements:

1. Molecules of rhodopsin are the only pigment molecules that occur naturally in the retina.
2. The visual systems of animals whose body temperature matches that of their surroundings are more error-prone in hot surroundings than in cold ones.

3. As the temperature of the retina rises, rhodopsin molecules react more slowly to being struck by photons.
4. The temperature of an animal's retina depends on the amount of light the retina is absorbing.

Answer: 2

=== Proposed program

```

python
import numpy as np
def molecular_motion(temperature):
 error_rate = np.poly1d([1, 0])(temperature)
 return error_rate
temperatures = np.array(range(0, 100))
error_rates = molecular_motion(temperatures)
consistent_statement_1 = False
consistent_statement_2 = True
consistent_statement_3 = False
consistent_statement_4 = False
print(f"Consistent with Statement 2: {consistent_statement_2}")

```

=== outputs

```

Observer: python interpreter
Feedback status: feedback ok
Content:
stdout:
Consistent with Statement 2: True

```

```

Comments
This program trivially represents all the statements
Program good
N

```

---

Table 8: Examples of trivial programs.

---

#### Hybrid Problem Examples

---

Answer the question below.

### Question:

Becky has 50 necklaces in her jewelry collection. 3 of the necklaces have broken beads so she collects the unbroken beads for crafting and throws the other parts of the 3 the necklaces out. Becky buys 5 new necklaces that week. She decides to give 15 of her old necklaces to her friends as gifts. How many necklaces does she own now?

37

—

Answer the question below.

### Question:

There were 15 males and 10 more girls at the party. Each attendee received 2 cans of soft drinks. If Mary bought several boxes of soft drinks where a box contains 8 cans and is priced at \$5 for each box, how much did Mary spend on soft drinks?

50

===

qtype: gg, shuffle: False, label: 1

Becky's jewelry collection sees a variety of activity; she starts with 50 necklaces. Amid handling 3 broken necklaces, she salvages the unbroken beads for crafting purposes. Furthermore, she expands her collection by purchasing 5 new necklaces while generously giving 15 old necklaces to friends, continually rejuvenating her collection's character.

At the vibrant social gathering, there were 15 males and 10 more girls, totaling an attendance of 25 individuals. Each person at the party was provided with 2 cans of soft drinks, ensuring everyone could enjoy their time refreshingly.

Y 1. If a box contains 8 cans and is priced at \$5, Mary spent \$50 on soft drinks.  
N 2. Becky now owns 35 necklaces.

---

Answer the question below.

### Question:

Mark is 18 years old. He has a little brother, John, who is 10 years younger. If John and Mark's parents are currently 5 times older than John, how old were they when Mark was born?

22

---

Given a set of premises and a hypothesis, answer if the hypothesis agrees with the premises [Agree], contradicts with the premises [Contradict], or neutral with respect to the premises [Uncertain].

### Premises:

1. Jason Kramer is an American music supervisor.
2. Some American radio personalities are also music supervisors.
3. Anyone who hosts a show on a public radio station is a radio personality.
4. Joe Rogan is a radio personality.
5. Jason Kramer hosted a show on a public radio station.

### Hypotheses:

Joe Rogan is American.

Uncertain

---

Answer the question below by choosing the correct statement.

### Context:

Editorial: The threat of harsh punishment for a transgression usually decreases one's tendency to feel guilt or shame for committing that transgression, and the tendency to feel guilt or shame for committing a transgression reduces a person's tendency to commit transgressions. Thus, increasing the severity of the legal penalties for transgressions may amplify people's tendency to ignore the welfare of others.

### Question:

Which one of the following is an assumption required by the editorial's argument?

### Statements:

1. The threat of harsh punishment deters people from committing transgressions only if this threat is at least sometimes carried out.
2. Everyone has at least some tendency to feel guilt or shame for committing extremely severe transgressions.
3. People who are concerned about threats to their own well-being tend to be less concerned about the welfare of others.
4. At least some actions that involve ignoring the welfare of others are transgressions.

===

qtype: gfr, shuffle: True, label: 1

Jason Kramer, an American music supervisor, shares a fascinating linkage with Joe Rogan, both existing in the dynamic sphere of radio personalities. While Joe Rogan is widely recognized for his engaging shows, Kramer is known for his music supervision but also hosted a show on a public radio station, making him a radio personality by definition. This interconnection reflects on others in the industry, as it's noted that some American radio personalities double up as music supervisors, enriching their careers with versatility.

In a different setting, Mark, now 18 years old, grows through life's stages, standing ten years above his younger brother, John. Their family dynamics are captivating as their parents are currently aged at a remarkable five times older than John, placing the parents in a stage of life filled with experience and wisdom.

Amid these personal stories, an intriguing discussion surfaces regarding the justice system's approach to transgressions. An editorial suggests that the threat of harsh punishment, rather than cultivating a sense of remorse or deterring wrongdoing, might actually reduce feelings of guilt or shame associated with transgressions. Consequently, it posits that escalating legal penalties could unintentionally promote neglect for the welfare of others, pointing to a complex interplay between law, emotion, and societal behavior. This narrative weaves through the lives of individuals, questioning how societal structures influence personal and professional lives.

Y 1. "Joe Rogan is American." is Uncertain to the passage above

N 2. When Mark was born, John and Mark's parents were 21 years old.

N 3. The assumption "The threat of harsh punishment deters people from committing transgressions only if this threat is at least sometimes carried out." is required by the editorial's argument.

---

Table 9: Example hybrid problems. The first problem is blended without shuffling and interleaving, and the second is blended with shuffling and interleaving.

---

#### Routing trajectory Examples

---

below are example questions and the steps that solve them

=== Example question

Read the context and choose the correct statement.

### Context:

Fluoride's journey into groundwater commences when rain interacts with soil, dissolving minerals rich in fluoride. Amidst this scientific realm, a fascinating study revealed that when variables such as rainfall and mineral concentrations are stable, areas with high sodium levels in the groundwater portrayed significantly increased fluoride concentrations. This distinct geological scenario interlaces with an academic setting where logical structures hold sway.

If someone secures a job at a school, they find themselves on the payroll, a fundamental link established within educational employment regulations. Building on this, all faculty members definitely have a job at a school, thus securing their position on the payroll. It's from here that the dual possibilities for Nancy emerge: if she's a teacher, naturally, she is paid by the school; if not, she remains unpaid. This dichotomy resonates with the vehicular debates in sports cars, known for their loud presence – every sports car defies the silence.

Switching focus to specific automotive brands, all cars fashioned in Maranello are indeed Ferraris, embedding a mark of luxury and speed wherein being a Ferrari defaults to being a loud sports car, contrasting sharply with the Prius which is neither a sports car nor inherently loud, suggesting that it might be a Maranello-made model or perhaps, ironically, a loud car under different conditions.

Meanwhile, a high school bowling team, engaging in a sport of precision and teamwork, accumulated a total of 810 points



in their first competitive outing. The scoring dynamics were intriguing: the first bowler earned 1/3 the points of the second, who in turn scored three times higher than the third. In these educational and recreational settings, logical sequences and dependencies manifest vividly, drawing parallels to the environmental phenomenon of fluoride transmission through nature's unassuming actions.

### Statements:

1. "Nancy is paid by the school and has students." Contradicts the passage above
2. "If Prius is a Ferrari or a loud car, then Prius is an electric car." Contradicts the passage above
3. The third bowler scored 159 points.
4. The statement "Fluoride-bearing minerals are not the primary source of fluoride found in groundwater." can most reasonably be concluded on the basis of the researchers' findings.

=== Final answer is one of the following

1, 2, 3, 4

=== Example steps

### Thought

Option 1 is a logic problem. I will use formal logic tactic to solve it.

### Action

## Name

Call tactic: formal logic z3

## Input

The main problem, and the past results

## Output

### option

1

### subproblem

Given a set of premises and a hypothesis, answer if the hypothesis agrees with the premises [Agree], contradicts with the premises [Contradict], or neutral with respect to the premises [Uncertain].

### Premises:

1. If you are on the payroll, then you are being paid by the school.
2. If someone has a job at a school, then they are on the payroll.
3. All faculty members have a job at a school.
4. One can either be a faculty member or a teacher.
5. Every teacher has students.
6. If Nancy is a teacher, then they are on the payroll.
7. If Nancy is not a teacher, then they are not paid by the school.

### Hypotheses:

Nancy is paid by the school and has students.

=== observations ===

# Observer: Action parser

# Feedback status: feedback ok

# Content:

Solving subproblem with tactic predicate\_logic\_z3

# Observer: Runner

# Feedback status: feedback ok

# Content:

Tactic execution successful. Tactic output:  
Agree

=== response ===

### Thought  
Option 2 is a logic problem. I will use formal logic tactic to solve it.  
### Action  
## Name  
Call tactic: formal logic z3  
## Input  
The main problem, and the past results  
## Output  
### option  
2  
### subproblem  
Given a set of premises and a hypothesis, answer if the hypothesis agrees with the premises [Agree], contradicts with the premises [Contradict], or neutral with respect to the premises [Uncertain].

### Premises:  
1. All sports cars are loud.  
2. No loud cars are electric.  
3. If a car is a Ferrari, then it is a sports car.  
4. All cars made in Maranello are Ferrari  
5. If Prius is neither a sports car nor a loud car, then Prius is a Maranello-made car or a loud car.

### Hypotheses:  
If Prius is a Ferrari or a loud car, then Prius is an electric car.

=== observations ===

# Observer: Action parser  
# Feedback status: feedback ok  
# Content:  
Solving subproblem with tactic predicate\_logic\_z3

# Observer: Runner  
# Feedback status: feedback ok  
# Content:  
Tactic execution successful. Tactic output:  
Contradict

=== response ===

### Thought  
Option 3 is a math problem. I will use math tactic to solve it.  
### Action  
## Name  
Call tactic: math  
## Input  
The main problem, and the past results  
## Output  
### option  
3

```

subproblem
Answer the question below.

Question:
A high school bowling team's 3 members scored a total of 810 points in their first match. The first bowler scored 1/3 as many points as the second, and the second bowler scored 3 times as high as the third bowler. How many points did the third bowler score?

=== observations ===

Observer: Action parser
Feedback status: feedback ok
Content:
Solving subproblem with tactic math

Observer: Runner
Feedback status: feedback ok
Content:
Tactic execution successful. Tactic output:
162

=== response ===

Thought
Option 4 is a commonsense reasoning problem. I will use general program tactic to solve it.
Action
Name
Call tactic: general program
Input
The main problem, and the past results
Output
option
4
subproblem
Answer the question below by choosing the correct statement.

Context:
Delta green ground beetles sometimes remain motionless for hours at a stretch, although they are more active in wet years than in dry years. In 1989 an observer spotted ten delta green ground beetles in nine hours; in 1985 the same observer at the same location had counted 38 in about two hours. This difference probably does not reflect a drop in the population of these rare beetles over this period, however, because 1985 was a wet year and 1989 was relatively dry.

Question:
Which one of the following, if true, most strongly supports the conclusion drawn above?

Statements:
1. Fluoride-bearing minerals are not the primary source of fluoride found in groundwater.
2. None of the above

=== observations ===

Observer: Action parser
Feedback status: feedback ok
Content:
Solving subproblem with tactic any_program

```

```
Observer: Runner
Feedback status: feedback ok
Content:
Tactic execution successful. Tactic output:
2

=== response ===

Thought
I have solved all the subproblems, I will aggregate the results and produce the answer
Action
Name
Aggregate and answer
Input
all thoughts, actions, and observations so far
Output
2
```

---

Table 10: An example routing trajectory.