Self-supervised Analogical Learning using Language Models

Anonymous ACL submission

Abstract

Large language models have been shown to suffer from reasoning inconsistency issues. That is, they fail more in situations unfamiliar to the training data, even though exact or very similar reasoning paths exist in more common cases that they can successfully solve. Such observations motivate us to propose methods that encourage models to understand the high-level and abstract reasoning processes during training instead of only the final answer. This way, models can transfer the exact solution to similar cases, regardless of their relevance to the pre-training data distribution. In this work, we propose SAL, a self-supervised analogical learning framework. SAL mimics the human analogy process and trains models to explicitly transfer high-quality symbolic solutions from cases that they know how to solve to other rare cases in which they tend to fail more. We show that the resulting models after SAL learning outperform base language models on a wide range of reasoning benchmarks, such as StrategyQA, GSM8K, and HotpotQA, by 2% to 20%. At the same time, we show that our model is more generalizable and controllable through analytical studies.

1 Introduction

While large language models (LLM) have achieved tremendous progress in reasoning benchmarks (OpenAI, 2023), they have been shown to have consistency issues (Li et al., 2023; Mirzadeh et al., 2024) — LLMs cannot produce consistent solutions on questions sharing identical or similar problem-solving processes. Previous studies (Li et al., 2024) have also shown that these models tend to perform well on cases involving familiar or common entities and words but fail on others. In Fig. 1, we demonstrate this issue with three questions that share an identical reasoning process (i.e., checking absence by checking location mismatch). The first question is a well-known fact that has

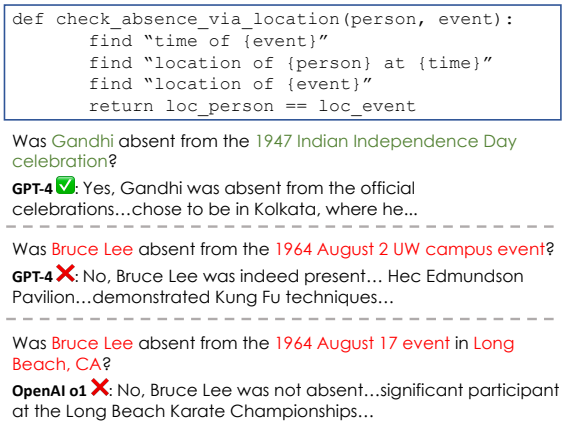

Figure 1: An illustration of models' (including o1) inconsistent reasoning processes for three questions that share the same problem-solving process (as outlined on top). Models memorize all relevant factual information (e.g., correctly answers where Bruce Lee was on those days). The correct answers are all "yes," because Bruce Lee was in California on 1964.8.2, and he was in Seattle to get married on 1964.8.17.

been widely reported in written text, and models answer it correctly with the correct reason. The other two, on the other hand, involve less common entity combinations, and the models fail to answer them correctly, even though they possess all relevant knowledge by successfully pointing out where the person was at the time of the event. This consistency issue is not significantly improved even with newer models demonstrating much stronger reasoning performances: OpenAI o1[1] also fails on the third question, albeit answering the previous two correctly. Such inconsistency prevents these models from deploying in mission-critical tasks (e.g., medical chatbots) requiring trustworthy and unbiased decision-making.

Following these observations, we are motivated to build models that can better understand the underlying high-level and abstract reasoning pro-

[1] https://openai.com/o1/

cesses instead of only memorizing the final answers or relevant tokens during training time. This way, models can re-use such reasoning paths in all similar cases and achieve better consistency. To this end, we propose a data-driven analogical learning framework called SAL (**S**elf-supervised **A**nalogical **L**earning). SAL extracts supervision signals with some inference processes using a base language model, and trains the same base language model with these signals, which forms a self-supervised learning scheme. The supervision signals we use are questions and their corresponding abstract and symbolic solutions, e.g., as shown on the top of Fig. 1, so that models can learn the high-level process that can bridge the inconsistency gap between various similar questions. We propose two novel supervision-extraction methods in SAL. The first method called *conceptualization*, starts with the original question and finds a group of similar questions that share **identical** high-level solutions (Zhou et al., 2024) of the original question. With these questions, we collect symbolic solutions in the form of Python program from the questions that the base language model can successfully answer, and use these symbolic solutions as supervision targets. The second method, called *simplification*, finds questions that share **sub-routines** of answering the original question, which can be used to eventually find the high-level solution of the original questions via composition.

SAL collects supervision signals from both extraction methods and trains the base language model with the resulting supervision signals. At its essence, this training process enforces a soft but symbolic constraint over the questions that share similar or simpler problem-solving processes. We also design an optional factual retrieval step to help models better generate a single solution for complex questions where considering all possible related factors is infeasible. After training, SAL inference first generates a symbolic program and executes the program to find the final answer. On a selection of complex reasoning datasets, including StrategyQA (Geva et al., 2021), ARC (Clark et al., 2018), CommonsenseQA (Talmor et al., 2018), GSM8K (Cobbe et al., 2021), and HotpotQA (Yang et al., 2018), we show that SAL effectively improves base language models' performance with direct inference methods such as chain of thought and human-supervised baselines, by 2% to 20%. On top of the performance gains, SAL provides higher interpretability and controllability because of its programmatic inference scheme.[2]

2 Related Work

Our work builds on analytical works that discusses language model's inconsistency (Tang et al., 2023; Li et al., 2023, 2024; Mirzadeh et al., 2024), and SAL is a learning scheme that mitigate such issues by encouraging consistency high-level solutions among relevant questions. SAL is related to previous works that use LLMs to generate programmatic solutions (Chen et al., 2022; Gao et al., 2023) or other structures (Feng et al., 2024). Some of them also aims to improve model consistency (Lyu et al., 2023) via programs, but our work considers abstraction and high-level solutions, which can transfer better. Several works have also been using analogy to encourage generalization (Hu et al., 2023; Yasunaga et al., 2024; Yuan et al., 2024; Zhou et al., 2024). Our work shares a similar motivation, but we are the first to propose automatic self-supervision methods and improve model performances in the training step. Our work is also related and motivated by other works that use self-supervision or distillation signals (Zhu et al., 2024; Zelikman et al., 2024). Decomposition-based inference pipelines (Khattab et al., 2022; Hao et al., 2023; Yao et al., 2023) also inspires our work, as our programmatical solutions resembles a decomposed inference process.

3 Self-Supervision Extraction

This section introduces how we collect the self-supervision signals in SAL through the two extraction methods: conceptualization and simplification. In both settings, we assume an original question q, and a base language model LM. Both extraction pipelines use LM to find a set of questions q'_0, q'_1...q'_N that share identical or sub-processes of the underlying problem-solving process of q. From $q'_i s$, we find a symbolic solution that describes the high-level reasoning process of q that also applies to $q'_i s$ to some extent. We assume and require **no knowledge** regarding the gold answer or process of q, which makes SAL self-supervised.

The symbolic solutions we acquire are in the form of Python programs. The overall program structure design follows Zhou et al. (2024) and supports general-domain question-answering (QA), including factual and math questions. On top of Zhou

[2]The code and data will be released upon publication.

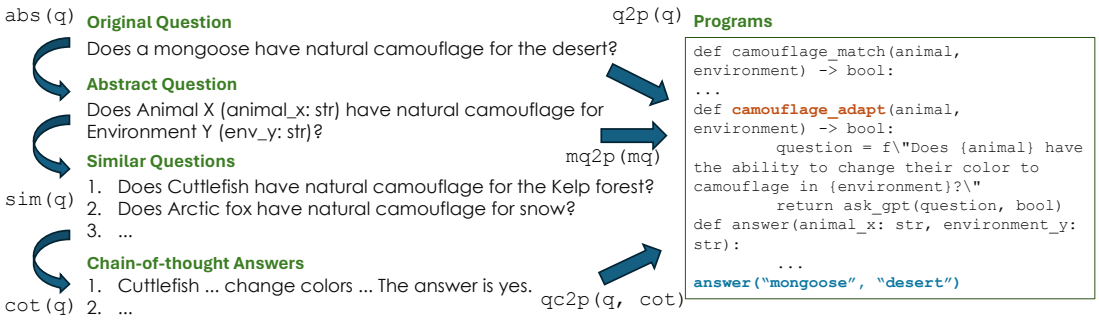

Figure 2: Overview of the conceptualization pipeline. Functions refer to sub-tasks and prompt templates.

et al. (2024) that uses ask_llm calls for knowledge retrieval and soft value comparisons, we add a step to identify the units for physical values. This avoids a knowledge retrieval query such as "What is the density of water?" without providing the expected unit. We detail this process in §B. All programs will contain a function named "answer()", which will return a binary yes/no value for general questions and a specific number for math questions.

3.1 Overview of Conceptualization

The conceptualization extraction pipeline finds q'_0, $q'_1 ... q'_N$ that all share the same reasoning paths as q, but are much easier for LM to confidently answer. Specifically, we first conceptualizes q into a conceptualized version q_{abs}, as shown in Fig. 2. Then, we generate a list of similar questions based on q_{abs}. With these questions, we select a set of high-confidence questions that language models tend to answer more accurately and use them as $q'_i s$. These components so far are primarily based on Zhou et al. (2024). Then, we propose a novel symbolic solution-generation process that generates high-quality programs from $q'_i s$ and their silver chain-of-thought outputs. We view these high-quality programs as supervision targets to q and q_{abs} to form self-supervision signals.

The intuition behind this is supported by recent works on LLM biases (Li et al., 2023, 2024), where it is shown that LLMs do much better on questions that contain familiar entity and token combinations. Motivated by such observations, the conceptualization pipeline automatically creates easier-to-answer questions with identical or very similar solutions to q, acquiring high-quality symbolic solutions from these similar questions.

3.2 Overview of Simplification

The simplification extraction targets math questions. We formulate solving a question q as a stepwise decomposition process that iteratively finds the next single-hop question that is easier to answer and contributes to the final solution. We solve the next question and add the conclusion to a set of "known conditions" until the decomposition finds no new questions. These "next questions" are the q'_is. Fig. 3 provides an example of *simplification*. The initial set of conditions are "Hadassah took 6 hours to paint 12 paintings" and "Hadassah paints 20 more paintings." A valid next question would be "What is the rate at which Hadassah paints?"

Our simplification method shares similarities in intuition with decomposition-based inference pipelines (Hao et al., 2023). Yet, we focus on acquiring a better global solution by reducing cognitive overload (Xu et al., 2023), which suggests that models perform better when they have fewer tasks simultaneously. We observe similar findings where models produce better programs when fewer computations are needed for the current question. For example, the most common mistake when solving the question in Fig. 3 is that programs fail to add the 6 hours Hadassah took to paint the first 12 paintings, as the question asks for a total time. This is because the model has to deal with other subprocesses (i.e., computing the number of paintings per hour), reducing its capacity to think comprehensively. When provided with the results of part of the computations, models tend to think more comprehensively and generate better programs.

3.3 Base Multi-function Model

As overviewed in §3.1 and §3.2, we need multiple components (i.e., outputs from LM) in both extraction pipelines. We annotate a small set of

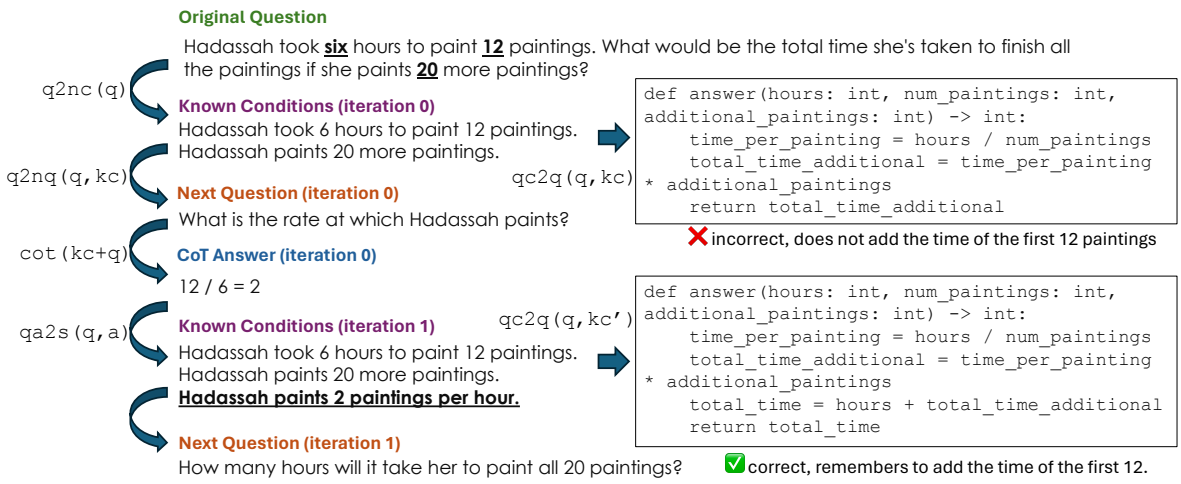

Figure 3: An example math question and overview of how our simplification process can acquire more correct incidental programs.

domain-agnostic seed supervision signals for individual components to achieve this. This is because smaller models cannot sufficiently accomplish the specific sub-tasks we need even after regular instruction tuning. Larger language models can work with few-shot prompting and waive the need for such seed supervision. With such seed supervision signals, equivalent to instruction-tuning data, we train LM to acquire an instruction-tuned base language model that can handle all sub-tasks needed in the pipelines. In this section, we introduce the individual components we need for both conceptualization and simplification extraction pipelines, as well as how we supervise LM to achieve these components. Instruction templates are listed in §A.

Chain of thought. We collect seed supervision signals of questions to their chain-of-thought (CoT) answers, with a prompt template of $\text{cot}(q)$, as shown in Fig. 2. We borrow the CoT results of 228 binary questions from the StratgyQA development set from Zhou et al. (2024) and acquire CoT outputs of an additional 265 math questions from the GSM8K training set using Mixtral-8x7B-Instruct.[3]

Questions to abstract questions. We collect seed supervision data of questions to their abstract or conceptual questions. We use the data from Zhou et al. (2024) on the same 228 StrategyQA questions. For the 265 math questions, we use Mixtral-8x7B-Instruct and few-shot prompting to generate similar abstractions. This process of generating abstractions is denoted as $\text{abs}(q)$ in Fig. 2.

Abstract questions to similar questions. We use the data from Zhou et al. (2024) to form seed supervision data of abstract questions to concrete similar questions, as $\text{sim}(q)$ shows in Fig. 2.

Questions to programs. We collect gold programs that can successfully solve the seed questions. We borrow the programs from Zhou et al. (2024) for the 228 StrategyQA questions. We similarly use Mixtral-8x7B-Instruct to find candidate programs that execute the correct answer for math questions. With such programs, we can formulate two sub-tasks and their seed supervision, namely original question to program ($\text{q2p}(q)$ as in Fig. 2) and abstract question to program ($\text{aq2p}(aq)$). We remove the "answer()" call in the target program in the second setting because the abstract question does not contain the specific input parameters.

Questions and relevant knowledge to programs. We want to train a model sensitive to relevant knowledge and reference answers so that the model can generate precise and faithful programs when chain-of-thought answers or retrieval results are provided. For example, as Fig. 2 shows, we want the generated program based on the CoT results related to "animal changing colors to adapt to an environment" to **explicitly** consider whether an animal can change colors. To do this, we formulate seed supervision data that uses the questions and their chain-of-thought answers as inputs and corresponding programs as outputs. This subtask is denoted as $\text{qc2p}(q, cot)$).

[3] https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Original question to initial known conditions. We generate seed supervision instances from original math questions to their initial sets of known conditions by few-shot prompting Mixtral-8x7B-Instruct using the 265 seed math questions in §3. This task is denoted as $\text{math_q2kc}(q)$.

Question to next questions. This task generates the next question based on the original and current known conditions. The next questions are designed to be the immediate next computation step, given the current known conditions, to solve the original question. Similarly, we acquire seed supervision instances by few-shot prompting Mixtral-8x7B-Instruct with the 265 seed math questions. This task and its prompt template are named $\text{math_q2nq}(q, c)$, where q is the original question, and c is the current known conditions.

Question and answer to statements. A final sub-task we define is transforming a question and its answer into a statement. We collect seed supervision data similarly by prompting Mixtral-8x7B-Instruct. This prompt template is named $\text{qa2s}(q, a)$.

Final Model. We train the base language model using the seed supervision data mentioned above, so this unified model can handle all sub-tasks as described by identifying the prompt templates we define in the seed supervision data. This is similar to all instruction-tuning methods that combine multiple sub-tasks during fine-tuning (Khashabi et al., 2020; Wei et al., 2022).

3.4 Conceptualization Extraction

With such a multi-function model, we can now automatically extract self-supervision signals using the conceptualization pipeline overviewed in §3.1. For each question q, we generate its abstract question q_{abs} by querying the multi-function model with template $\text{abs}(q)$. We then generate a list of N concrete questions that fit the abstract question q_{abs} by prompting the model with the similar question generation prompt $\text{sim}(q_{abs})$. For each concrete similar question q'_i, we acquire their chain-of-thought answers by prompting $cot(q'_i)$ and random sample K times. We keep only the questions where X out of the K CoTs agree with each other to select high-quality questions that are easy and confident enough for the model to answer. For each of the selected similar questions q'_i, we find one of the CoT results cot'_i that infer to the majority-voted answer and generate a programmatic solution using $p_i = \text{qc2p}(q'_i, cot'_i)$.

3.4.1 Selection Criteria and Parameters

In this work, we set $N = 20$, $K = 10$, and $X = 9$, meaning that we generate 20 similar questions per abstract question and keep those with at least a 9/10 agreement in their chain-of-thought predictions. After the inference process, we would have a set of programs generated from these high-confidence similar questions. Because we have multiple programs from this generation process, we are motivated to further select from them and use the ones with the highest quality possible as the self-supervision signals. We now describe a set of selection criteria we use for this purpose. All hyper-parameters are tuned with manual examinations over a small set of examples, and we expect minimum variances because of the large data sizes.

Accuracy-based criteria. A natural selection process is to verify each programmatic solution against the high-confidence similar questions by trusting their corresponding silver labels. We execute each of the generated programs with the parameters corresponding to each selected similar question. Let the number of remaining similar questions be M; we set a high threshold of $max(3, M * 0.75)$ and select programs that execute to the correct silver labels[4] over such a threshold number of questions. We also employ a low threshold of 2, which will be used later.

Similarity-based criteria. We employ a similarity-based metric using a paraphrase model from sentence transformers[5] to encourage the model to generate programs that do not repeat the original question and perform actual decomposition and reasoning. We drop all programs containing a ask_llm query with a paraphrase score higher than 0.95.

Complexity-based criteria. We also want the model to self-supervise on programs considering more abduction factors, which implies more query calls to ask_llm. As a result, we drop all programs that only make one query using this helper function.

Soundness-based criteria. We also check for details in the generated program to ensure soundness.

[4] Here the silver labels refer to the CoT answers of each generated similar questions, as we do not use gold labels of the original questions in any way.

[5] https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2

System	#Instances	Accuracy	Collected Pct.
Original	344	91.3	25.3%
Iteration 1	103	86.4	32.9%
Iteration 2	92	90.2	39.7%
Iteration 3	46	87.0	43.1%
Iteration 4	15	80.0	44.2%
Iteration 5	1	100.0	44.3%

Table 1: Number of math questions that produced ten programs that execute to the same answer at each timestamp of the simplification iteration. Collected Pct. indicates the number of accumulated questions from which we can acquire high-confidence programs.

Specifically, we drop all programs that contain unused parameters in the $\text{answer}()$ function call and all programs that produced an error message when executing on more than two similar questions.

We formulate two kinds of self-supervision instances from the selected programs. The first is question-to-program, where the corresponding similar question is used as input, and we use the $\text{q2p}()$ prompt template and the low accuracy threshold described above as long as the program successfully executes the correct silver answer on that specific similar question. The second is abstract-question-to-program, using the high accuracy threshold to ensure the target program is representative enough for all scenarios and the $\text{aq2p}()$ input template.

3.5 Simplification Extraction

At inference time, for each math question q, we first generate the initial set of known conditions by prompting the base multi-function model with $\text{math_q2kc}(q)$. Then, we propose an iterative process with the following steps. 1) Generate the next question using $\text{math_q2nq}(q, c)$; 2) Answering the next question with $\text{cot}(kc + nq)$, where we concatenate the current known conditions with the next question into a single query; 3) Adding the next question and its question to the set of known conditions by transforming them into a statement using $\text{qa2s}(nq, a)$. 4) Generating solutions in our programmatic space using $\text{qc2p}(q, kc)$, where q is the original question, and kc is the current known conditions. The iteration stops if one of the two criteria happens. A) The generated next question contains "no more decomposition," which is part of the above seed training data. B) 9/10 of the generated programs from 4) execute the same answer, which implies high confidence in the predictions.

Sanity-check Experiments We conduct a sanity-check experiment to verify the effectiveness of our proposed simplification method for mathematical questions. We aim to acquire as many high-confidence programs as possible and evaluate the quality of these high-confidence programs. We run our proposed simplification pipeline on 2k math questions from GSM8K, among which 1,358 questions produced valid programs. Because we want to collect high-confidence programs, we count the number of math questions that can produce ten programs that lead to the same answer after execution. As Table 1 shows, without our simplification pipeline, only on 25.3% of the questions can the model generate ten programs that lead to the same answer, which we consider as a valid self-supervision instance. At the 5th iteration of our simplification process (i.e., generating 5 next questions iteratively), we can collect high-confidence programs on 44.3% of the math questions, with high accuracy on par with the original generation (i.e., generating programs directly from the original question). This suggests that our proposed method can effectively acquire more self-supervision signals without undermining their qualities.

4 Training with Analogical Supervision

As mentioned in §1, SAL first collects self-supervision signals and then trains the model with them. We now describe the training process.

4.1 Direct Training

The direct training scheme simply combines all the self-supervision instances and the seed training instances and trains a new model with a supervised fine-tuning (SFT) process. Our goal is to show that the models trained with our automatic self-supervision data can outperform two baselines: 1) chain-of-thought inference with the base language model's instruction-tuned version, 2) the base language model supervised with the seed supervision from §3.3 only.

4.2 Retrieval-augmented Generation

The reasoning process sometimes requires some prior knowledge to reduce the size of the abduction. For example, consider the question, "Can I visit Museum X 20 times for under 200 dollars?" When generating programs, the model will almost always consider the ticket price for a single entry because it is the most common pricing structure. However,

System	StQA-seen	StQA	GSM-seen	GSM	ARC	CsQA	Hotpot
CoT (few-shot)	58.8	58.8	44.6	39.2	70.5	62.2	32.7
Seed Only	57.6	56.1	45.4	45.0	68.3	73.5	50.3
Ours							
SAL (concept. only)	61.1	58.8	44.7	43.7	68.3	72.0	51.5
SAL (all)	61.0	59.7	49.4	50.8	71.0	74.7	52.4
SAL (all) + RAG	63.7	62.9	49.4	50.8	73.7	77.0	55.1

Table 2: System performances on a range of reasoning datasets. The base model is Mistral-7B. Few-shot CoT uses Mistral-7B-Instruct-v0.1.

System	StQA-seen	StQA	Avg. Δ
Seed Only	45.8	45.8	-11.1
SAL	58.0	56.4	-3.2

Table 3: System performances after removing all programs that internally query the language model with a question very similar to the original question.

there is a complicated pricing structure for certain museums, such as the MET museum, including annual memberships, senior and student discounts, and residential passes. It is impractical for models to consider all possible pricing structures in the first pass. We build a retrieval-augmentation mechanism to allow models to consider external world facts before generating the solutions, reducing the search space and improving the feasibility of generating a single program to solve the question. To achieve this, we take all the ask_llm queries in the generated programs from the first pass and ask the base language model to answer them with a short paragraph. Then, we use $\mathrm{qc2p}(q, ans)$ to generate the second-pass programs, where ans is the short answering paragraph. Whenever a voted answer from the first-pass programs is "unknown," we use the retrieval-augmented programs instead.

5 Experiments

In this section, we conduct experiments to show the effectiveness of our proposed framework SAL.

5.1 Self-Supervision Extraction

The seed supervision data contains 4619 instances that have programs as outputs. To expand them with self-supervision, we apply the conceptualization pipeline on 2k questions from StrategyQA and 800 questions from HotpotQA. We apply simplification on 5k questions from GSM8K. All questions come from the training sets. Such processes produced 5865, 1000, and 4439 self-supervision training instances, respectively.

5.2 Baselines, Datasets and Parameters

Due to resource concerns and other constraints, we use Mistral-7B-v0.1 as our base language model. Both the training for the multi-function model in §3.3 and later experiments on evaluating the effectiveness of SAL supervision signals are done with the same model because of the self-supervision assumption. ask_llm is handled by few-shot prompting to its instruction-tuned version (Mistral-7B-Instruct-v0.1). All fine-tuning processes use LoRA (Hu et al., 2021) (r=32) and a learning rate of $2e-4$. We use $k = 10$ as the number of candidate programs at inference time.

We evaluate five reasoning datasets: StrategyQA, GSM8K, ARC, CommonsenseQA, and HotpotQA. Because of the limitations of our program space (i.e., only support binary answers), we transform all questions to binary for multiple-choice datasets. Specifically, we design a few-shot prompt and use Mixtral-8x7B-Instruct to transform the question with its correct answer to a question with a correct answer of "yes". As this is a simple rephrasing task, Mixtral demonstrates 100% accuracy in the 50 human-selected examples. For StrategyQA and GSM8K, we create a special evaluation split called "seen", which contains questions we use as raw questions when generating self-supervision signals. Models do not see or use the gold labels of these questions in any way. The overall evaluation data contains 500 StrategyQA (seen), 490 StrategyQA, 500 GSM (seen), 500 GSM, 112 ARC, 336 CommonsenseQA and 339 HotpotQA instances.

We consider two main baselines: a few-shot chain-of-thought (**CoT**, self-consistency@10) of Mistral's instruction-tuned version, and the base model trained on seed supervision instances (§3.3) only, named "**Seed Only**." All models are trained to the same steps, which equals five epochs of the full training data (seed + self-supervision). We report averaged results from two random seeds for

System	StQA Dev.	StQA test	BoolQ	CsQA	Hotpot
Mixtral 8x7B (few-shot)	53.9	52.9	41.1	10.7	55.3
Llama-70B (few-shot)	56.8	59.7	58.9	53.4	65.6
Mistral7B + Seed Only	60.1	44.3	40.1	59.0	50.0
Mistral7B + SAL	59.9	56.0	51.2	59.0	51.2

Table 4: Conceptual Reasoning Performance. We use a slightly different program space, as described in §3.

experiments that require SFT.

5.3 Results

Table 2 shows our main experiment results of reasoning performances across the five reasoning datasets. We first observe that the seed only baseline performs relatively similarly to the CoT baseline, indicating that the seed supervision data mostly teach the model forms of the programmatic space without improving their reasoning capabilities by much. On the other hand, we see that the base LLM trained on self-supervision data from conceptualization extraction only (SAL (concept. only)) improves 3.5%, 2.7%, and 1.2% on StrategyQA's seen, unseen, and HotpotQA, demonstrating the effectiveness of our proposed self-supervision signals. By adding the simplification self-supervision (SAL (all)), we improve 4% and 5.8% on GSM8K's seen and unseen sets, showing that our simplification pipeline can effectively extract self-supervision data on math-like questions. In addition, we observe better domain transferability as SAL (all) improves on ARC and CommonsenseQA by 2.7% and 1.2%, respectively. Model performance is further improved by adding the retrieval-augmented generation in §4.2. The final model is consistently better than the CoT baseline (+6.8% on StrategyQA, 5.8% on GSM8K, and 4.8% on HotpotQA). At the same time, our model is consistently better than the seed only baseline, demonstrating the overall effectiveness of our proposed self-supervision framework.

6 Analysis

To further demonstrate the capability of our self-supervision signals, we conduct an experiment that disregards all programs that contain a $\mathrm{ask_llm}$ query that is at least 0.9 similar to the original question, using the same paraphrase detection model in §3. As the results show in Table 3, the seed baseline model drops significantly by 11% on StrategyQA, meaning that a significant portion of its generated programs simply repeat the original question and delegate reasoning to the instruction-tuned model behind $\mathrm{ask_llm}$. This is not as much a concern in our self-supervised model as it only drops 3%. This suggests the self-supervised model is better at reasoning by conducting comprehensive abduction and decompositions.

We also consider the conceptual reasoning benchmark proposed in Zhou et al. (2024), which considers four datasets and 1597 test instances. In this conceptual setting, systems see an abstract version of the question and must make conceptual reasoning considering all possible corner cases, which we can achieve by querying our models with $\mathrm{aq2p}(aq)$. As Table 4 shows, the model trained with self-supervision signals improves 11.7% on StrategyQA, 11% on BoolQ, and 1.2% on HotpotQA. At the same time, it outperforms the larger Mixtral-8x7B-Instruct model on several tasks, demonstrating the effectiveness of using automatic self-supervision signals. Such high performance on conceptual reasoning suggests that our proposed self-supervised model improves neutral and unbiased reasoning by establishing internal consistencies in the actual reasoning processes.

7 Conclusion

This work introduces SAL, self-supervision analogical learning that encourages LLM consistent reasoning in common and rare situations. We propose two methods to collect self-supervision signals: 1) conceptualization, a pipeline that finds programmatic solutions from similar questions that models can confidently answer and share identical reasoning processes; 2) simplification, which finds high-quality solutions to math questions by gradually decomposing a question and reducing models' logical overload. Experiments show that the self-supervision signals are high-quality and effective, which can be used to train models more attentive to the actual reasoning process instead of the semantic inductive biases. SAL leads to performance improvements on a wide-range of reasoning tasks.

Limitations

Our work has several limitations in its current form.

Limited Data Sizes. We only consider 3k questions for the conceptualization pipeline and 5k questions for the simplification pipeline. Because of the selection criteria, the resulting data size is only 2.5 times larger than the seed supervision. This limits our arguments on large-scale self-supervision.

Binary Questions Only. We only consider binary questions in the conceptualization pipeline because it is the most straightforward way to evaluate a program solution's accuracy on similar questions. While all multiple-choice questions can be easily transformed into binary questions, this limits the scalability to more free-form questions and needs to be addressed in future works.

Single Base Model. Due to computational constraints, we only experiment with one base LLM (Mistral-7B), and more extensive experiments with other families of models may provide more insight into the capabilities of different models. Yet, we hypothesize that the general trend should be consistent with other base models.

Single Base Model. Due to computational constraints, we only experiment with one base LLM (Mistral-7B), and more extensive experiments with other families of models may provide more insight into the capabilities of different models. Yet, we hypothesize that the general trend should be consistent with other base models.

Limited Ablation Studies. Due to computational constraints, we do not conduct ablation studies on the hyper-parameters used in §3, although these parameters can be further fine-tuned to achieve better performances. However, we believe that the hand-picked parameters we employ are good enough because of manual examinations, and we expect little changes in empirical results if these numbers are changed slightly because we extract a relatively large set of supervision instances.

References

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

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*.

Yu Feng, Ben Zhou, Weidong Lin, and Dan Roth. 2024. Bird: A trustworthy bayesian inference framework for large language models. *arXiv preprint arXiv:2404.12494*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *EMNLP*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *ICLR*.

Xiaoyang Hu, Shane Storks, Richard L. Lewis, and Joyce Yue Chai. 2023. In-context analogical reasoning with pre-trained language models. In *ACL*.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system. In *ACL (findings)*.

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

Bangzheng Li, Ben Zhou, Xingyu Fu, Fei Wang, Dan Roth, and Muhao Chen. 2024. Famicom: Further demystifying prompts for language models with task-agnostic performance estimation. *arXiv preprint arXiv:2406.11243*.

Bangzheng Li, Ben Zhou, Fei Wang, Xingyu Fu, Dan Roth, and Muhao Chen. 2023. Deceptive semantic shortcuts on reasoning chains: How far can models go without hallucination? *NAACL*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

OpenAI. 2023. Gpt-4 technical report.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Xiaojuan Tang, Zilong Zheng, Jiaqi Li, Fanxu Meng, Song-Chun Zhu, Yitao Liang, and Muhan Zhang. 2023. Large language models are in-context semantic reasoners rather than symbolic reasoners. *arXiv preprint arXiv:2305.14825*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Nan Xu, Fei Wang, Ben Zhou, Bang Zheng Li, Chaowei Xiao, and Muhao Chen. 2023. Cognitive overload: Jailbreaking large language models with overloaded logical thinking. *arXiv preprint arXiv:2311.09827*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed Huai hsin Chi, and Denny Zhou. 2024. Large language models as analogical reasoners. *ICLR*.

Siyu Yuan, Jiangjie Chen, Changzhi Sun, Jiaqing Liang, Yanghua Xiao, and Deqing Yang. 2024. Analogykb: Unlocking analogical reasoning of language models with a million-scale knowledge base. *ACL*.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.

Ben Zhou, Hongming Zhang, Sihao Chen, Dian Yu, Hongwei Wang, Baolin Peng, Dan Roth, and Dong Yu. 2024. Conceptual and unbiased reasoning in language models. *arXiv preprint arXiv:2404.00205*.

Xuekai Zhu, Biqing Qi, Kaiyan Zhang, Xinwei Long, Zhouhan Lin, and Bowen Zhou. 2024. Pad: Program-aided distillation can teach small models reasoning better than chain-of-thought fine-tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2571–2597.

A Prompts

A.1 Few-shot Prompts for Seed Supervision

Table 5 lists the prompts we use for collecting seed supervision of math question abstractions. Non-math question abstractions are done in Zhou et al. (2024), and we directly borrow their data. Table 6 shows the prompts we use to collect seed supervision for generating gold math questions. The generated programs' validity is further verified by the gold answer. Table 7 details the prompt for collecting seed supervision data for generating the next questions in the simplification pipeline. This prompt asks for the entire decomposition with multiple next questions in one query, later formulated as multiple instances with one next question per instance. Table 8 shows the prompts for collecting seed supervision instances on known conditions. It handles two scenarios: extracting the list of known conditions from the original question and transforming the answer to the next question into a statement. These two scenarios are formulated as separate seed supervision instances.

A.2 Input Templates

Table 9 details the input prompt templates for each sub-task as described in §3.1 and §3.2. A multi-function model trained with the same prompt templates can recognize the different goals based on the query because the seed supervision instances use the same templates.

B Implementations

Table 10 shows a partial implementation of our programmatic solution space. In addition to this, we also employ the soft value comparison helper functions as described in Zhou et al. (2024).

All SFT experiments are conducted with 4x RTX6000 GPUs and a single-run fine-tuning finishes in 3 hours. We useHuggingface's Autotrain framework for training.[6] All parameters are default parameters unless explicitly mentioned otherwise.

[6] https://github.com/huggingface/autotrain-advanced

Prompt	Content
system	Identify numerical values in the given question, then replace some of them with Python parameters that are either int or float, so that the resulting abstract question is still answerable with the same general solution as the original question. Follow the provided examples.
user	Jack has a stack of books that is 12 inches thick. He knows from experience that 80 pages is one inch thick. If he has 6 books, how many pages is each one on average?
assistant	Becuase this is a math question, we identify all numerical values. We identify: 12 inches, 80 pages, one inch, 6 As a result, we can replace "12 inches" to "Number of Inches X" (num_inches_x: int) "80 pages" to "Number of Pages Y" (num_pages_y: int) "one inch" to "Number of Inches Z" (num_inches_z: int) "6" to "Number W" (num_w: int) So the question becomes Jack has a stack of books that is Number of Inches X thick. He knows from experience that Number of Pages Y is Number of Inches Z thick. If he has Number W books, how many pages is each one on average? With parameters num_inches_x=12, num_pages_y=80, num_inches_z=1, num_w=6

Table 5: Prompts for collecting seed supervision of math question abstractions. More examples are omitted.

Prompt	Content
system	Based on the given question, write a Python program with some abstraction that solves the given question and all other similar questions that can be solved in a similar fashion. Think as comprehensively as possible, so that the program would work on any inputs. You can use the following helper function: ask_gpt(query: str, type) to ask a simple question to the knowledgeable GPT model and returns the answer in the desired type.
user	Benny bought Number of Soft Drinks X (num_soft_drinks_x: int) for Cost per Soft Drink Y (cost_per_soft_drink_y: int) each and Number of Candy Bars Z (num_candy_bars_z: int). He spent a total of Total Amount Spent W (total_spent_w: int) dollars. How much did each candy bar cost?

assistant	`def answer(num_soft_drinks_x: int, cost_per_soft_drink_y: int, num_candy_bars_z: int, total_spent_w: int) -> float:`
	`  total_cost_soft_drinks = num_soft_drinks_x * cost_per_soft_drink_y`
	`  total_cost_candy_bars = total_spent_w - total_cost_soft_drinks`
	`  cost_candy_bar = total_cost_candy_bars / num_candy_bars_z`
	`  return cost_candy_bar`

Table 6: Prompts for collecting seed supervision of math programs. More examples are omitted.

Prompt	Content
system	Given a math question, your job is to decompose it into sub-questions that can stack together and lead to the correct answer of the original question. Each sub question should be answerable on its own. Use '####' to mark your actual answer when answering each sub question. Reference the correct answer.
user	Benny bought 2 soft drinks for$ 4 each and 5 candy bars. He spent a total of 28 dollars. How much did each candy bar cost?
assistant	Question 1: Benny bought 2 soft drinks for$ 4 each. How much does the soft drinks cost in total? Answer 1: The total cost of the soft drinks are «2*4=8» #### 8 Question 2: Benny spent $8 on soft drinks and bought 5 candy bars. He spent a total of 28 dollars. How much did the candy bars cost? Answer 2: The candy bars cost «28-8=20» #### 20 Question 3: Benny spent $20 to buy 5 candy bars. How much did each candy bar cost? Answer 3: Each of the candy bars cost «20/5=4» #### 4 END
user	There are some lions in Londolozi at first. Lion cubs are born at the rate of 5 per month and lions die at the rate of 1 per month. If there are 148 lions in Londolozi after 1 year, how many lions were there in Londolozi at first?
assistant	Question 1: There are some lions in Londolozi at first. Lion cubs are born at the rate of 5 per month and lions die at the rate of 1 per month. What is the number of new lions per months? Answer 1: There are «5-1=4» new lions per months. #### 4 Question 2: There are some lions in Londolozi at first. There are 4 new lions every months. If there are 148 lions in Londolozi after 1 year, how many lions were there in Londolozi at first? Answer 2: The number of new lions in one year is «4*12=48». The original lions are «148-48=100» #### 100 END

Table 7: Prompts for collecting seed supervision of math next questions. More examples are omitted.

Prompt	Content
system	Given a math question, extract all the known conditions that are explicitly mentioned. If the answer is provided, extract only the condition that the answer implies.
user	Question: Benny bought 2 soft drinks for$ 4 each and 5 candy bars. He spent a total of 28 dollars. How much did each candy bar cost? Answer: unknown
assistant	Benny bought 2 soft drinks for $4 each. Benny bought 5 candy bars. Benny spent 28 dollars in total.
user	Question: Wickham is throwing a huge Christmas party. He invites 30 people. Everyone attends the party, and half of the guests bring a plus one (one other person). He plans to serve a 3-course meal for the guests. If he uses a new plate for every course, how many plates does he need in total for his guests? Answer: unknown
assistant	Wickham invites 30 people. Half of the guests bring a plus one. Wickham prepares 3 courses for each person. He uses a new plate for each course.
user	Question: If lion cubs are born at the rate of 5 per month and lions die at the rate of 1 per month, what is the net increase in the number of lions per month? Answer: 4
assistant	The number of lines increase by 4 every month.
user	Question: It takes 7 years for an apple tree to bear fruit. If Lydia planted a tree when she was 4 years old, how old would she be when the tree bears fruit for the first time? Answer: 11
assistant	Lydia would be 11 years old when the tree bears fruit for the first time.
user	Question: Wickham invites 30 people to the party. Half of the guests bring a plus one. How many additional people (plus ones) are coming to the party? Answer: 15
assistant	15 additional people are coming to the party.

Table 8: Prompts for collecting seed supervision of extracting initial known conditions and transforming a question and its answer to a statement. "unknown" answers refer to the initial known condition generation scenarios.

Subtask	Content
$\mathrm{cot}(q)$	Answer this question by thinking step by step. Question: {q} Output:
$\mathrm{abs}(q)$	Convert this question to the abstract form. Question: {q} Output:
$\mathrm{q2p}(q)$	Solve this question with a Python program with necessary abstractions. Also generate the corresponding values for the parameters. Question: {q} Output:
$\mathrm{aq2p}(aq)$	Solve this abstract question with a Python program. Abstract Question: {aq} Output:
$\mathrm{qc2p}(q, cot)$	Solve this question with a Python program based on the provided reference answer. Question: {q} Reference Answer: {cot} Output:
$\mathrm{math_q2kc}(q)$	Extract the given conditions in the math question. Question: {q} Output:
$\mathrm{math_q2nq}(q, c)$	Decompose the math question to steps, based on currently known conditions. Question: {q} Known Conditions: {c} Output:
$\mathrm{qa2s}(q, a)$	Convert the given math question and its answer to a statement. Question: {q} Answer: {a} Output:

Table 9: Input prompt templates for each sub-task that the multi-function model handles.

```
def query_llm(messages):
  # library function that makes a query to an LLM.


def get_unit(question):
    messages = [
        {"role": "user", "content": "Does the answer to the following
                                                question require a
                                                measurement unit? If
                                                it is physical metric
                                                , use SI units. If it
                                                 is monetary metric,
                                                use 'USD'.\nQuestion:
                                                 What is the density
                                                of copper?\nAnswer
                                                with the proper unit,
                                                 or 'None' if not
                                                applicable."},
        {"role": "assistant", "content": "kg/m^3"},
        {"role": "user", "content": "More examples omitted"},
        {"role": "user", "content": "Can the following question be
                                                answered with a
                                                metric in the
                                                International System
                                                of Units (SI Units)?\
                                                nQuestion: {}\nAnswer
                                                 with the proper unit
                                                , or 'None' if not
                                                applicable.".format(
                                                question)},
    ]
    result = query_llm(messages)
    if 'None' not in result:
        return result
    return None


def find_json(s):
    # finds valid JSON structure in the output


def ask_llm(question, t, use_cache=False):
    unit = get_unit(question)
    question += " Answer in {}.".format(unit)
    messages = [
        {"role": "system", "content": "Answer the question in the
                                                expected type. Use
                                                your best educated
                                                guess or estimation
                                                if needed."},
        {"role": "user", "content": "How many people today are related
                                                to Genghis Khan? (int
                                                )"},
        {"role": "assistant", "content": "{\"answer\": 35000000}"},
        {"role": "user", "content": "More examples omitted"},
        {"role": "user", "content": "{} ({})".format(question, t.
                                                __name__)}
    ]
    response = query_llm(messages)
    r = find_json(response)
    return r
```

Table 10: Implementations of the helper functions in our programmatic solution space. Some details are omitted for simplicity.