Self-Training Language Models in Arithmetic Reasoning

Marek Kadlčík[♣*] **Michal Štefánik**[♣*] **Ondřej Sotolář**[♣] **Vlastimil Martinek**[◊]

[♣]**Faculty of Informatics, Masaryk University, Czech Republic**

[◊]**The Central European Institute of Technology, Masaryk University, Czech Republic**

Abstract

Recent works show the impressive effectiveness of an agent framework in solving problems with language models. In this work, we apply two key features from the framework, interaction with tools and goal-oriented training, to improve models' arithmetical reasoning.

First, we curate and transform existing datasets to create Calc-X, a standardized collection with over 300,000 problems with step-by-step solutions. We use Calc-X to train models we call Calcformers that interact with a calculator during inference. Calcformers achieve twice the accuracy of standard baselines.

Finally, we optimize Calcformers via self-training using preference optimization and supervised loss by checking the model's predicted results. We find that self-training can achieve substantial improvements on out-of-domain problems and that traditional supervised loss is a strong baseline for preference optimization. Our results show that preference optimization converges faster and isn't prone to forgetting pre-trained abilities.

1 Introduction

While language models (LMs) demonstrate effectiveness in working with unstructured language data in many settings, they often struggle with arithmetical computations (Patel et al., 2021) or multi-step reasoning (Hendrycks et al., 2021), which are necessary prerequisites for solving math problems.

The need for exact computations was recently addressed by integrating a calculator tool into the LM inference process (Thoppilan et al., 2022). The interaction with the tool can be learned, but a scarcity of suitable data caused a substantial effort in related work towards heuristics, prompting, few-shot, or reinforcement approaches (Section 2) with compromises in the quality and reproducibility.

We address the issue by creating Calc-X, a collection of over 300,000 arithmetical problems in a standardized format (Section 3.1) by curating, transforming, and cleaning existing datasets. Calc-X contains annotation of calculations in solutions, enabling the training of tool-using models (Fig. 1). We describe the process of creating Calc-X in Section 4, and in Section 6, we show that the mere data standardization and interaction with a calculator roughly *doubles* the accuracy of resulting models.

Subsequently, Calc-X allows us to explore the potential of preference optimization methods to improve reasoning by training for *achieving the correct result* in self-training. We experiment with three preference optimization methods: *Direct Preference Optimization* - DPO (Rafailov et al., 2023), *Kahneman-Tversky Optimization* - KTO (Ethayarajh et al., 2024), and *Identity Preference Optimization* - IPO (Azar et al., 2023), and we assess the quality of resulting models against different self-training supervised baselines.

In evaluation on 6 arithmetic datasets, we find that self-training with *any* assessed method can bring significant improvements compared to the original model before self-training. However, different methods *vary* in the robustness of their delivered gains (Section 7). The methods' robustness *can* be enhanced with parameter-efficient training. We make our code, datasets, and models publicly available (Appendix G).

[*]Equal contribution

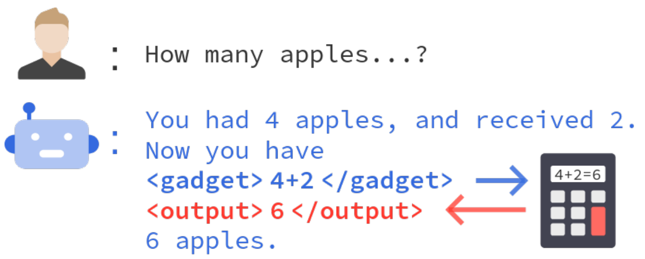

Figure 1: **Generation process of Calcformer models**: By generating the *gadget* tag, model calls an external tool. The following tokens are inserted into the model's context by the tool. Then, the model continues generation using conditioned on all tokens.

2 Related Work

Arithmetic Reasoning Datasets *GSM8K* (Cobbe et al., 2021) contains grade-school math problems with human-written CoT explanations and explicit annotation of formulas. *ASDiv* (Miao et al., 2020), *SVAMP* (Patel et al., 2021), *MAWPS* (Koncel-Kedziorski et al., 2016) and larger Chinese *Ape210K* (Zhao et al., 2020) contain problems of similar complexity, but the solutions are written as nested expressions. *AQuA-RAT* (Ling et al., 2017) contains multiple-choice problems with selected answers and free-text rationales. *MathQA* (Amini et al., 2019b) is a subset of *AQuA-RAT* with additional annotation: nested expressions that lead to an answer, similar to those in *Ape210K* or *ASDiv*. *MATH* and *AMPS* (Hendrycks et al., 2021) consist of more challenging problems and contain CoT solutions formatted in LaTeX. Specifically, *MATH* is a set of high-school math competition problems, and *AMPS* is a large-scale pre-training dataset, partly scraped and partly synthetically generated. *Mathematics Dataset* (Saxton et al., 2019) is another generated dataset but contains only the final results without rationales. *PRM800K* (Lightman et al., 2023) is a dataset of per-step correctness annotations for model-generated solutions.

Tool-using LMs A main contribution of much of the previous work in building tool-using models addresses the problem of data scarcity. Komeili et al. (2022), *WebGPT* (Nakano et al., 2021), and *LaMDA* (Thoppilan et al., 2022) let crowd workers annotate tool calls to train models to use a web search, a calculator, and a machine translation system. *PAL* (Gao et al., 2023) applies prompt engineering to make an LM use a Python interpreter without training. *Toolformer*'s approach (Schick et al., 2023) is to prompt an LM to insert gadget tags ("API calls") into CoT datasets and filter out irrelevant ones using the trained model's perplexity. In evaluation, Toolformer simplifies the problem to only one tool call per example and supports generation with several heuristic rules.

TaLM (Parisi et al., 2022) extends a training dataset by *self-training*: They start with a small set of fully annotated data, including the annotations of tool calls, and then iteratively generate CoTs with tool calls for a larger dataset with incomplete annotation.

We note that *none* of the referenced work publicly releases the resulting models. In combination with the many training and inference heuristics, it is largely difficult to reproduce and build upon the proposed methods. However, the availability of an extensive, standardized collection of tool-assisted datasets like the one presented by Calc-X will allow future work to substantially simplify the methods needed for creating tool-assisted models.

Self-training for math problems *WizardMath* (Luo et al., 2023) augments *GSM8K* and *MATH* datasets using pretrained LMs and applies PPO (Schulman et al., 2017) optimization against feedback from *ChatGPT 3.5* given to individual reasoning steps in predicted solutions. Uesato et al. (2022) self-train on *GSM8K* and compare the effect of per-step feedback given by an LM verifier and algorithmically checking the final result. Lightman et al. (2023) use a similar methodology on a larger scale with *MATH* and *PRM800K* datasets and extend the comparison with an LM verifier for whole CoTs.

3 Calc-X Collection

3.1 Interaction Format

We propose a semi-structured format for CoT data to provide both the flexibility of unstructured text and the precision of structured formats. The HTML-based structure of interactions is compatible with existing parsers, such as BeautifulSoup (Richardson, 2007). This allows future work to easily extend Calc-X collection or convert it to other desired formats.

Our format, displayed in Figure 2, uses three tags: *gadget*, *output*, and *result*. Tag *gadget* is intended for calls to an external function. Tag *output* wraps the response of the external system. The tag *result* wraps the final result of the thought chain.

```
After buying the bread and candy bar, you have 32−3−2=
<gadget id="calculator">32−3−2</gadget><output>27</output>$27.
You spend 27/3=<gadget id="calculator">27/3</gadget><output>9</output>
9 dollars on the turkey. You have 27−9=<gadget id="calculator">27−9</gadget>
<output>18</output>$18 left. The final result is 18.<result>18</result>
```

Figure 2: An example of target text from a chain-of-thought dataset encoded in our proposed format. Our format is designed to allow the interaction of LMs with a calculator.

4 Calc-X Curation Process

Out of the datasets reviewed in Section 2, we create the Calc-X collection from these datasets: *GSM8K*, *AQuA-RAT*, *MathQA*, *Ape210K*, *MAWPS*, *SVAMP* and *ASDiv*. Our selection considers the datasets' size, primary focus on arithmetics, and parseability of the tool calls.

The resulting Calc-X collection is designed to simplify the *correct* usability of the whole collection in *both* training and evaluation while persisting the maximum of datasets' original information. Most importantly, the process includes (1) the unification of format over all datasets and (2) the elimination of data leakages between different (train/val/test) data splits throughout the *whole* collection.

We perform the second step based on a lexical overlap between pairs of samples' input texts from different splits across *all* datasets. We consider a pair of train and test examples a leak if the Jaccard similarity of their 1-gram and 2-gram representations is over 0.5. This results in data splits composed of *subsets* of the original datasets, but thanks to this step, the whole Calc-X collection can be used to perform both validation and tests over *all* datasets when *all* datasets are also used in training.

The remainder of this section describes the conversion process of each dataset included in Calc-X.

GSM8K (Cobbe et al., 2021) is a CoT dataset with over 8,000 examples containing arithmetical expressions that can be evaluated using a calculator. The syntax is not standard but can be easily parsed.

"Natalia sold 48/2 = ⟨⟨48/2 = 24⟩⟩ 24 clips in May.
Natalia sold 48+24 = ⟨⟨48 + 24 = 72⟩⟩ 72 clips altogether in April and May.
72"

Figure 3: The original syntax of the *GSM8K* dataset.

In *GSM8K*, the calculations are explicitly annotated, and removing the tags from chain-of-thought results in natural language sentences. A final result is a single number that is also explicitly annotated at the end of the solution.

We parse the formulas using regular expressions, evaluate them using the sympy library (Meurer et al., 2017), and verify that all outputs are numerically close to the values in the data. The conversion into our unified format is a direct one-to-one mapping.

Our analysis shows that the original validation and test splits of *GSM8K* do not contain duplicates and are not contained in a training split of other datasets.

Nested expression:

```
(2 - 8) + (2 - 8) * (50% + 3)
```

Linear chain:

```
2 - 8 = -6
50 / 100 = 1/2
(1/2) + 3 = 7/2
(-6) * (7/2) = -21
(-6) * (-21) = -27
```

Figure 4: Example of *linearization* of nested expressions applied in creating Calc-X reasoning chains for *Ape210K*, *MathQA* and *MAWPS* datasets.

Ape210K (Zhao et al., 2020) is a dataset of over 200K Chinese math problems involving simple arithmetics. The solutions are represented as nested arithmetical expressions and a single numerical result to which they evaluate. We automatically translate the questions to English using Google Translate and linearize the nested expressions into a sequence of simple expressions using depth-first traversal of the expression tree. Figure 4 illustrates the process of linearization.

Furthermore, we discard all examples that cannot be parsed. Then, we evaluate all steps and remove examples where the end result does not numerically match the result saved in the data. We also discard all examples containing expressions of form "⟨number⟩(⟨fraction⟩)", such as 1(1/2) because of the ambiguity between implicit multiplication and mixed fractions, which are both used in the data indistinguishably. In total, more than 97% of the examples in each split were kept in the dataset.

Finally, the linearized examples can be directly transformed into our unified format. While *Ape210K* is much larger than *GSM8K*, the exported chains do not contain any comments in natural language, and the English prompts are machine-translated.

Analysis of overlaps shows that around 60% of both validation and test examples present duplicates or near-duplicates to the *Ape210K*'s training split. In Calc-X, we remove these examples from validation and test splits with around 1700 remaining in each.

AQuA-RAT (Ling et al., 2017) is a dataset of 100K math problems. The annotations consist of 1) multiple choices, 2) the correct choice, and 3) an informal, free-text rationale that explains the selected choice. The answer is usually a single number but can also be a pair of numbers (coordinates), include a unit, a ratio, "None of the options," and others.

The rationale is in free-text format and generally not parseable with formal grammar. In some cases, calculations are written in words, such as "ten pages per day means seventy pages per week." We approach this in a best-effort manner and use regular expressions to find equations in the form of *expression* = *number*. We remove all the characters (such as units) from both sides and evaluate the left-hand side using a calculator. Finally, we compare the calculator output with the right-hand side, and if they match, we insert a tagged calculator call into the rationale. This results in 1.6 calculator calls on average per reasoning chain. For applications with high priority of recall in the injected gadget calls, we propose to further filter our dataset to the samples with at least three calculator calls.

Analysis of data leaks shows around 2% of the training split are near-duplicates of around 30% and 25% of test and validation *AQuA-RAT* samples, respectively. In Calc-X collection, we remove these samples from the train split.

MathQA (Amini et al., 2019b) is a subset (37K) of *AQuA-RAT* with further annotations. Human annotators have corrected errors inside the *AQuA-RAT* rationales and annotated the solution with a nested expression that leads to the correct answer.

We parse the nested expressions and linearize them using a procedure similar to that used for *Ape210K*. Less than 0.3% of examples were removed due to parsing or evaluation problems. We also replace all function calls (such as circle_area) with operations executable with a sympy calculator.

Next, we keep the examples only if their expression evaluation result is in ±5% range of the selected correct choice in the data, which results in a loss of around 30% of the data. We note that the mismatch of the computed results with annotated options is not consistent with the authors' claim that the expressions in the dataset are guaranteed to evaluate to the selected option (Amini et al., 2019a), but is consistent with observations by Parisi et al. (2022). We attribute most of these errors to the inconsistency in the original *MathQA* dataset. We remove all inconsistent examples in Calc-X.

Evaluation of data leakages shows that *all* samples of *MathQA* originate from the *training* split of *AQuA-RAT*. Hence, we completely remove the validation and test splits of *MathQA* in Calc-X and omit evaluations on *MathQA* in our results.

MAWPS (Koncel-Kedziorski et al., 2016) is a collection of around 5000 elementary school-level problems from online websites. The solution to each problem is annotated as an equation with a single result variable x. We isolate x from the equations by manual annotation and then linearize the corresponding expression into a sequence of calculations to convert the data into our unified format.

Around 70% of *MAWPS*'s train samples are near-duplicates of its train split, test split, or *ASDiv-A* test split. We remove these samples from Calc-X's train collection.

ASDiv-A (Miao et al., 2020) is an arithmetics benchmark with problems of similar difficulty as *MAWPS*. *ASDiv-A* picks around 1,200 samples with a number as a solution and a nested expression evaluating to the correct result.

SVAMP (Patel et al., 2021) comprises 1,000 math problems derived from *ASDiv*, overcoming some of its statistical biases. Whole *ASDiv-A* and *SVAMP* datasets were directly convertible to our common format. With no official train-test split, we use both for testing only.

5 Experiments

5.1 Training on Calc-X collection

To explore the potential of large and consistent arithmetic reasoning data collection, we train models in three configurations:

1. **Train baselines on the original datasets**: We use all of the selected datasets (see Section 4) to train standard generative models that produce a CoT on a given problem. All training samples removed from Calc-X are also removed from baseline data for fair comparison.
2. **Train Calcformers on the Calc-X datasets**: We train models with identical objective, but on the corresponding Calc-X datasets, to enable interaction with a calculator. Then, we evaluate the models in 2 variants that differ in generation to measure the effect of the calculator. In Calcformer ON variant, whenever the model generates a *gadget* tag, the model's generated text is extended with the *output* tag containing the response from a calculator (see Figure 1). In Calcformer OFF variant, models generate every output token, including the content of *output* tags.
3. **Retrain the best-performing Calcformer on Calc-X with style instructions**. We use the same configuration as in 2. but prepend a dataset-specific *style instruction* before each training input. The details of the style instructions can be found in Appendix C. During inference, we apply *GMS8K* style instruction to guide the model to mimic the high-quality solutions present in *GMS8K* dataset, which include explanations and consistent usage of a calculator in every reasoning step.

In the experiment, we fine-tune pre-trained T5 models of Raffel et al. (2020), and Chung et al. (2022) in 700-million and 3-billion parameters' versions, using cross-entropy loss, commonly applied on sequence-to-sequence Transformers (Bahdanau et al., 2016; Vaswani et al., 2017). We use greedy decoding in inference.

We evaluate both systems by numerically comparing the value of the final result extracted from the generated answer with the ground truth result.

In the case of the Calcformer models, the numerical result is enclosed in the <result> tags that we use for extraction. For the baseline models, we extract the result from the phrase "The final result is", which is present in all the baseline training samples. Our training setup is further detailed in Appendix A.

5.2 Offline Self-training on Arithmetic Reasoning

5.2.1 Offline Self-training dataset

We use the first 50 000 problems in *Ape210K* as a base of the self-training dataset. Using only a single dataset for self-training allows us to inspect the accuracy on the remaining datasets, indicating the methods robustness.

We generate up to 20 solutions per example with a top-k=5 sampling with the best three checkpoints of Calcformer-Flan-XL-instruct. As the original *Ape210K* dataset contains only a sequence of operations (without the natural-language description of operations), we applied *GSM8K*-style instruction during inference, causing the model to generate chains *including* explanations instead of reproducing the *Ape210K* training data from memory. After generating the predictions, we assess that the model is indeed sensitive to the instructed style by comparing the proportions of *gadget calls* and *natural explanations*: In the predictions, 52.6% of characters are explanations, compared to 54.0% in the *GSM8K* train set (our target style) and 0% in the original *Ape210K* train set.

We evaluate these predictions based on whether the result is correct and sample the following quadruple for each problem: *problem*, *correct chain 1*, *correct chain 2*, and *incorrect chain*. Only the problems with at least two correct chains and one incorrect chain are kept, which results in a self-training dataset of 24,000 examples. The self-training dataset is referenced in Appendix G.

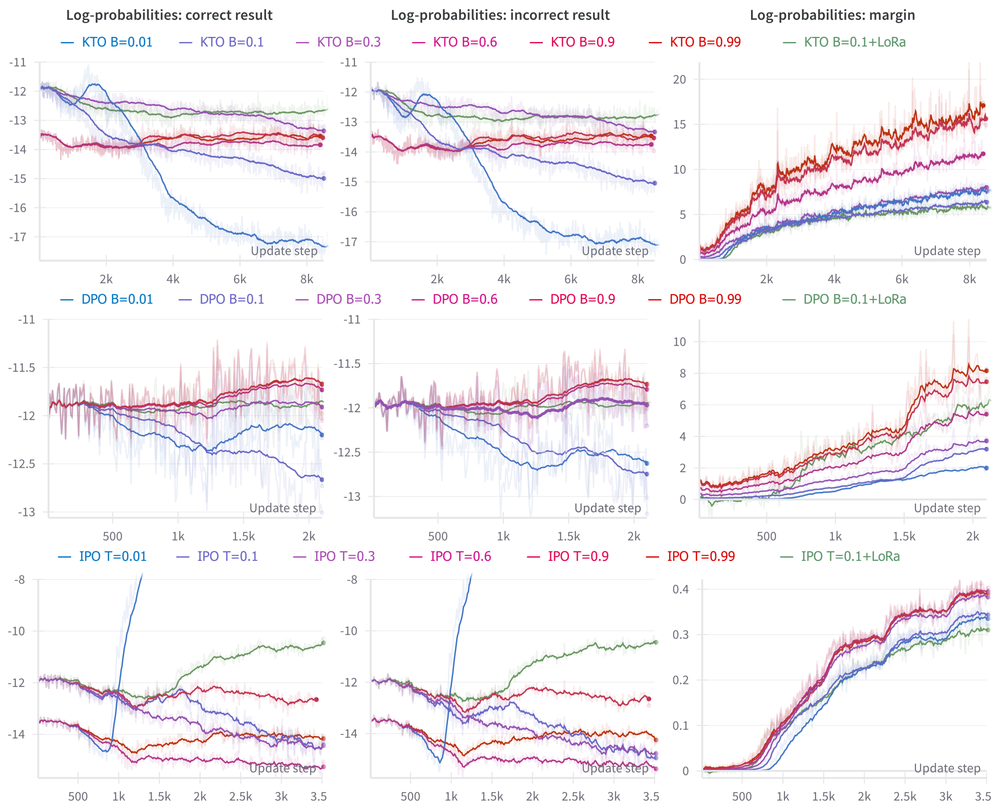

Figure 5: Log-probabilities for preference optimization methods on chains leading to correct answers (left), incorrect answers (middle), and their margin (right) show large sensitivity to hyperparameter selection. We note that smaller margins tend to correspond to the best-performing configurations.

5.2.2 Offline Self-training settings

In all variants of self-training, we fine-tune a trained Calcformer-Flan-XL-instruct checkpoint from the previous experiment. We use the same values for common training hyperparameters shared between all variants. The details are listed in Appendix B.

Supervised self-training We use next-token cross-entropy loss in all supervised runs using predictions with correct results as gold labels. Specifically, in Supervised-baseline, we train on pairs *(problem, correct chain 1)* from our self-training dataset. For Supervised-balanced, we create two training examples from each row: *(problem, correct chain 1)* and *(problem, correct chain 2)*. The aim is to have a *comparable* amount of training tokens in the data as the preference training settings. In Supervised-with-neg, we also create two training examples from each row of the self-training examples: *(problem, correct chain 1)* and *(negative prompt + problem, incorrect chain)* where the negative prompt is a phrase "Write incorrect solution for the following question." In this setting, we assess whether supervised training can benefit from being exposed to negative samples via a naive prompting approach. For better reproducibility, all variants of the self-training dataset are publicly available and referenced in Appendix G.

Preference self-training In preference training, we use triplets *(problem, correct chain 1, incorrect chain)* from our self-training dataset and pose a preference for the correct chain over the incorrect one. We experiment with three preference objectives: DPO (Rafailov et al., 2023), KTO (Ethayarajh et al., 2024), and IPO (Azar et al., 2023).

All objectives have a single hyperparameter (β in DPO and KTO, τ in IPO) expressing the strength regularization by the *reference model* (initial checkpoint), for which we perform a hyperparameter search over values 0.01, 0.1, 0.3, 0.6, 0.9 and 0.99. In addition, we replicate the best-performing runs using a low-rank adaptation, LoRA (Hu et al., 2022), to measure the effect on performance and robustness. We apply the adapters with a rank of 32 to all linear layers.

5.3 Online Self-training on Arithmetic Reasoning

In the online self-training experiment, we generate the training data dynamically during training. To create new data, we sample a problem from *Ape210K* and generate 16 solutions with the currently trained model. Then, we classify whether the predicted result matches the result in the data. In the supervised variant, we filter the solutions with correct results and use them as training labels. In the preference variants, we sample pairs of solutions where exactly one has a correct result and is marked as preferred. The configuration details of the training pipeline are listed in Appendix F.

In this experiment, we use the same base checkpoint as in the offline experiment, and we select the best-performing hyperparameter configuration per training method according to the offline experiment.

6 Results

6.1 Training on Calc-X collection

Table 1 compares the accuracy of the conventional generation and calculator-using generation of models trained on Calc-X datasets. calculator-using models surpass the accuracy of the baseline models 2-3 times, with the exception of the *AQuA-RAT* dataset. The overall improvement of the Calcformers in reaching the correct answer is 97.1% on average across all datasets compared to corresponding baselines trained on the original data format. In addition, by evaluating Calcformer models with standard generation, we can attribute 65% of the improvement to tool-using generation and 35% to the formatting of the training data alone.

On the *AQuA-RAT* dataset, Calcformers perform comparably with baselines in the case of two out of three base models. We find that this is due to the necessity of using intermediate variables in many *AQuA-RAT* questions, which will require integrating more complex symbolic systems than a calculator, such as the Python interpreter, leaving this a challenge for future work.

We also see encouraging the model to use *GSM8K* output style via style instruction significantly hurts the accuracy on *Ape210K* dataset and helps on the remaining five. We hypothesize that this is because *Ape210K* is the only dataset without natural explanations and with most solutions over two reasoning steps long, making it the hardest for the model to extrapolate to. We illustrate the effect of style instruction on output in Appendix D.

	GSM8K	AQuA-RAT	Ape210K	MAWPS	SVAMP	ASDiv-A
GPT-3 (175B)				19.9*	10.0*	14.0*
Toolformer (6.7B)				44.0*	29.4*	40.4*
T5-L (700M)						
Calcformer ON	**31.6**±2.5	27.2±6.6	**52.9**±2.3	**38.5**±4.1	**34.9**±3.0	**55.8**±2.8
Calcformer OFF	22.8±2.3	27.2±5.5	37.4±2.2	16.0±3.2	25.2±2.7	39.2±2.7
Baseline	16.3±2.0	26.6±6.6	18.8±1.8	10.4±2.6	17.2±2.3	30.0±2.6
T5-XL (3B)						
Calcformer ON	**39.3**±2.7	33.5±6.9	**53.9**±2.3	**49.2**±4.3	**44.0**±3.0	**67.1**±2.6
Calcformer OFF	28.5±2.5	25.6±5.3	37.8±2.3	21.0±3.5	32.4±2.8	48.4±2.8
Baseline	19.0±2.1	31.8±6.9	20.8±1.9	16.2±3.2	25.1±2.7	37.2±2.7
Flan-XL (3B)						
Calcformer ON	**42.4**±2.7	31.2±6.6	**53.4**±2.4	**45.4**±4.3	**46.8**±3.1	**72.2**±2.5
Calcformer OFF	31.2±2.5	24.8±5.3	36.6±2.2	19.0±3.4	32.4±2.9	51.3±2.8
Baseline	23.7±2.3	21.4±6.4	23.0±1.9	16.5±3.2	23.3±2.6	38.3±2.7
Flan-XL (3B)						
Calcformer instruct	43.2±2.7	$\underline{37.8}$±6.1	26.3±2.1	$\underline{61.9}$±4.2	$\underline{51.8}$±3.2	$\underline{78.7}$±2.3

Table 1: Test accuracy comparing tool-using generation and standard generation (ON vs. OFF). For reference, we compare the models to previous work and to Baselines trained on the same data but in the original formats. Values marked with * are self-reported by Schick et al. (2023). Confidence intervals are bootstrapped (sample size 500 with 1,000 repeats). Bold denotes the best results for each base model. The overall best base model was retrained with style instructions to mimic high-quality *GSM8K* solutions. The entries where it improved performance are underlined.

6.2 Offline Self-training on Arithmetic Reasoning

Table 2 shows the accuracy of models trained with different self-training methods described in Section 5.2. While at least one of the preference optimization methods performs better on each evaluation dataset, both approaches achieve gains of similar scale and **the difference in gains between the best supervised and preference optimization approach is small**.

Comparing improvements on the self-training dataset (*Ape210K*) and others suggest that **KTO provides more robust improvements**. However, the improvements of both DPO and IPO on *other* datasets can be fostered by parameter-efficient training.

Among all methods, the Supervised-with-neg method using negative samples in supervised training through negative prompt performs the worst, bringing small gains only on two datasets, including the self-training dataset. This result shows that using negative feedback in supervised training analogically to preference optimization methods might require more sophisticated approaches, presenting a challenge for future work.

Figure 5 provides a more detailed report of the impact of preference optimization in self-training. We can see that **preference optimization methods are not consistent in maximizing the likelihood of correct sequences and minimizing the likelihood of incorrect ones**. This behavior suggests potential improvements in their design in future work. Further, all methods *can* maximize the margin between two groups, but the margin does not correspond to the end task quality of the resulting configuration: for instance, the best-performing configuration of KTO with $\beta = 0.1$ maintains a relatively small correct/incorrect margin.

We also examined the convergence speed of all methods. On average, the preference trainings without LoRA listed in Table 2 achieved the best validation score at around 2,400 steps, compared to 16,600 steps in supervised setup. LoRA variants are slower, but the difference varied considerably between methods. A detailed report can be found in Appendix E.

6.3 Online Self-training on Arithmetic Reasoning

Table 3 compares the accuracy of training methods in the online setting described in Section 5.3. Notably, supervised setup performs considerably well on *MAWPS*, *SVAMP*, and *ASDiv-A* compared to preference training. These datasets contain mostly single-step problems (with reference solutions

	GSM8K	AQuA-RAT	Ape210K	MAWPS	SVAMP	ASDiv-A
Calcformer-Flan-XL-instruct	43.2±2.7	37.8±6.1	26.3±2.1	61.9±4.2	51.8±3.2	78.7±2.3
Supervised-baseline	**46.1**±2.7	**37.8**±5.9	32.9±2.2	**70.6**±3.8	56.2±3.0	81.9±2.2
Supervised-balanced	45.8±2.7	37.4±5.9	**33.6**±2.2	66.7±3.9	**58.4**±3.0	**82.0**±2.2
Supervised-with-neg	41.8±2.7	33.1±5.7	28.0±2.1	65.2±4.1	52.2±3.1	75.9±2.4
DPO (β = 0.99)	45.3±2.7	37.0±5.9	29.2±2.1	69.6±3.9	54.2±3.1	83.1±2.1
DPO (β = 0.9)	37.2±2.6	40.9±6.1	32.8±2.3	61.2±4.1	52.2±3.1	78.1±2.3
DPO (β = 0.9) + LoRA	45.9±2.7	**41.3**±6.1	32.4±2.2	64.4±4.0	57.1±3.1	84.7±2.0
KTO (β = 0.3)	**47.1**±2.7	38.6±6.1	36.4±2.2	**78.3**±3.5	55.8±3.1	85.3±2.0
KTO (β = 0.1)	47.0±2.7	40.6±6.1	**37.9**±2.3	68.3±3.9	57.2±3.1	86.4±1.9
KTO (β = 0.1) + LoRA	43.1±2.7	36.2±5.9	37.6±2.2	64.2±4.1	58.5±3.3	87.0±1.9
IPO (τ = 0.9)	38.4±2.7	39.0±5.9	26.9±2.1	71.3±3.8	64.6±3.0	87.4±1.9
IPO (τ = 0.99)	40.7±2.7	36.6±5.9	28.1±2.1	66.3±4.0	64.5±3.0	**87.8**±1.8
IPO (τ = 0.99) + LoRA	36.0±2.6	39.4±5.9	30.2±2.1	66.7±4.0	**65.6**±3.0	**87.8**±1.8

Table 2: Correct results obtained in offline self-training on *Ape210K*. For each preference optimization method, we report results for its two best-performing configurations. Bold entries denote the best results among supervised and preference optimization methods per dataset.

	GSM8K	AQuA-RAT	Ape210K	MAWPS	SVAMP	ASDiv-A
Calcformer-Flan-XL-instruct	43.2±2.7	37.8±6.1	26.3±2.1	61.9±4.2	51.8±3.2	78.7±2.3
Supervised	37.1±2.6	2.4±2.0	48.1±2.3	98.3±1.3	69.8±2.8	89.8±1.7
DPO (β = 0.9)	49.1±2.7	39.8±5.9	37.9±2.3	79.6±3.4	57.3±3.1	85.6±2.0
KTO (β = 0.1)	52.7±2.7	36.6±6.1	49.6±2.4	85.2±3.0	62.6±3.1	90.6±1.6
IPO (τ = 0.99)	49.1±2.8	35.8±5.9	42.2±2.3	81.5±3.4	56.8±3.0	86.6±1.9

Table 3: Comparison of training methods in online self-training on *Ape210K*. Results on *AQuA-RAT* with answer-selection prompts suggest that preference optimization methods are better at maintaining abilities obtained in pre-training, while SFT outperforms preference optimization on single-step and two-step tasks (MAWPS, SVAMP and ASDiv-A).

having 1.02, 1.24, and 1.17 steps on average). In contrast, supervised training significantly decreased the accuracy on more challenging *GSM8K* (3.25 steps), where all preference methods made a significant improvement. *AQuA-RAT*, the only dataset with choice selection, shows the largest difference in accuracy, where supervised self-training dropped accuracy nearly to zero, while 20% can be achieved by choosing options randomly. We found that the supervised self-training on data with numerical results caused the model to forget how to answer choice selection questions, while preference optimization kept this pre-trained ability intact.

7 Conclusion & Future Directions

This paper introduces a Calc-X dataset collection, transforming over 300,000 arithmetic reasoning problems into a unified chain-of-thought format with explicit calculation annotation. Calc-X enables integrating a calculator in the reasoning of language models via traditional supervised learning. We eliminate the datasets' mutual data leakages, making Calc-X a convenient default for any future research addressing arithmetic reasoning or tool-using models.

Next, we demonstrate the potential of Calc-X by training tool-using models. They outperform existing work and approximately *doubles* the accuracy of baselines. We make the Calc-X collection and calculator-using Calcformer models publicly available to facilitate further research.

Finally, we explore the application of preference optimization methods for self-training on arithmetical problems and find that the supervised approach presents a competitive baseline to preference optimization methods. B both approaches can improve performance on a majority of benchmarks, preference methods may provide more robust improvements, as evidenced by out-of-distribution evaluations. For a price of slower convergence, parameter-efficient adaptation methods might further complement robustness improvements of preference optimization methods.

References

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa - dataset, annotation, validation, and examples, 2019a. URL https://math-qa.github.io/. Accessed on 10/05/2024.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *CoRR*, abs/1905.13319, 2019b. URL http://arxiv.org/abs/1905.13319.

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences, 2023.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, 2016. URL https://arXiv.org/abs/1409.0473v7. arXiv:1409.0473v7.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling Instruction-Finetuned Language Models. *arXiv e-prints*, art. arXiv:2210.11416, October 2022. doi: 10.48550/arXiv.2210.11416.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874, 2021. URL https://arxiv.org/abs/2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8460–8478, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.579. URL https://aclanthology.org/2022.acl-long.579.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *CoRR*, abs/1705.04146, 2017. URL http://arxiv.org/abs/1705.04146.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2023.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021. URL https://arxiv.org/abs/2112.09332.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models, 2022.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(146):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Leonard Richardson. Beautiful soup, 2007. URL https://pypi.org/project/beautifulsoup4/.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *CoRR*, abs/1904.01557, 2019. URL http://arxiv.org/abs/1904.01557.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena

Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Agüera y Arcas, Claire Cui, Marian Croak, Ed H. Chi, and Quoc Le. Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239, 2022. URL https://arxiv.org/abs/2201.08239.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Proc. of the 31st NIPS conference*, volume 30 of *NIPS '17*, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. Ape210k: A large-scale and template-rich dataset of math word problems. *CoRR*, abs/2009.11506, 2020. URL https://arxiv.org/abs/2009.11506.

A Calc-X training details

A.1 Training settings

We trained the models using the HuggingFace Transformers library. We applied standard sequence-to-sequence training with **cross-entropy** loss on all tokens. We optimized the model with **AdamW** optimizer and effective **batch size** of 32. We used **learning rate** of 5e-5 with 1000 **warmup steps**, and a **linear lr decay** to 0 in 400000 steps. The models were trained in bf16 **precision**. All models were trained on a mixture of all datasets, either in Calc-X or in original format, with **data upsampling** to balance different dataset sizes.

During training, we monitored the percentage of validation predictions with a correct final result generated with greedy decoding. We compute the performance on each dataset separately and average them together, which we use for **early stopping** and selecting the best checkpoint after training.

B Offline self-training details

All self-trained supervised models were trained using the HuggingFace Transformers library, all preference models using the HuggingFace TRL library. HuggingFace PEFT library was used for LoRA models.

All self-trained models were optimized with **Adafactor** optimizer with an **effective batch size** of 32, a **learning rate** of $2e-5$ with 1000 **warmup steps**, and a **linear lr decay** to 0 in 1 million steps. The models were trained in bf16 **precision**. All LoRA models used adapters on all linear layers with a **rank** of 32 and α of 32.

The self-training checkpoints used for testing were early-stopped and selected according to accuracy (correct results produced with greedy decoding) on a random subset of 200 validation examples from Calc-X *Ape210K*. The random subset was consistent across variants.

C Style instructions

Datasets in the Calc-X collection differ in their chain format. Calcformer-Flan-XL-instruct was informed about the expected output style in the prompt during training. We list here all style instructions for each dataset below. Curly brackets **{}** denote a place where the math problem is inserted into the instruction. Templates were sampled randomly during training based on their probability weights.

Ape210k

	template	weight
0	Solve the math problem. Use a calculator for all calculations. Do **not** write down the reasoning. {}	0.2
1	Solve this problem. Use a calculator program. No explanations are allowed, just write all intermediate steps. {}	0.2
2	{} No need to write down how you solved it. Just call calculator API to obtain intermediate values.	0.2
3	Answer this question: {} No explanations are allowed, but explicitly state each computation.	0.2
4	{} You must use a only html-like tags to format your answer. Free-text is forbidden.	0.2

AQuA-RAT

	template	weight
0	Solve this: {} You can use a calculator, but you don't have to. Good formatting is not important.	0.2
1	Answer the question. Explain your reasoning step-by-step, but no need to be thorough. You can call calculator API when it's convenient. {}	0.2
2	{} Can you explain how to find the solution step-by-step? you can use function calling with a calculator app, but it's not that important.	0.2
3	Explain how to solve this: {} calculations tags as annotation are not required but can be used. Don't worry about formatting much, just get to the answer.	0.2
4	You can see a math problem below. Write down the solution step-by-step. You can use a calculator. Although, neither calculator calls or nice formatting are strictly necessary. {}	0.2

GSM8K

	template	weight
0	{}	0.5
1	Solve the following math word problem using a calculator. {} Describe each step in words.	0.05
2	You can use a calculator. Solve this problem and explain your reasoning. {}	0.05
3	Solve the math problem below. State each calculation and provide explanation for each step. {}	0.05
4	Answer the following question and give a step-by-step solution. You can call calculator API. {}	0.05
5	{} Calculate the solution to the math problem with a calculator program. Explain your method clearly an format it well.	0.05
6	Please compute the answer using a calculator function calls.{} Write down the reasoning for each step.	0.05
7	{} You can call a calculator for this problem to obtain correct intermediate results. Make sure to solve it correctly.	0.05
8	I'm stuck with this problem. Please help me solve it. You are allowed to utilize a calculator to minimize risk of wrong computation. {}	0.05
9	Can you explain to me how to solve this? {} Don't try to compute it manually, just use a calculator.	0.05
10	{} I don't know how to solve this. Write down each step in words and explicitely annotate each calculation.	0.05

MathQA

	template	weight
0	{} Don't try to eliminate answer choices, just compute the answer and then check if it's in the options. Use calculator but don't write down the reasoning.	0.2
1	{} Compute the answer and then select the corresponding choice. Keep the output structured as a sequence of calculations and output tags.	0.2
2	Try to solve this problem using a calculator function. Then find the answer in the choices. Avoid free-text rationales, stick to calculations only. {}	0.2
3	Solve this word problem with a series of calculator API calls. No explanations are allowed, just write down all intermediate calculations. {}	0.2
4	You have access to a calculator program. Use it to find the solution. {} Avoid unstructured text, record computations only.	0.2

MAWPS

	template	weight
0	Solve the math problem. Use a calculator for all calculations. Do **not** write down the reasoning. {}	0.2
1	Solve this problem. Use a calculator program. No explanations are allowed, just write all intermediate steps. {}	0.2
2	{} No need to write down how you solved it. Just call calculator API to obtain intermediate values.	0.2
3	Answer this question: {} No explanations are allowed, but explicitly state each computation.	0.2
4	{} You must use a only html-like tags to format your answer. Free-text is forbidden.	0.2

D EXAMPLE OUTPUTS WITH STYLE INSTRUCTION

To illustrate the effect of style instruction on models' output, we sample 10 examples randomly (with seed 0) from CALC-X *Ape210K* test set and list here the generated predictions using CALCFORMER-FLAN-XL-INSTRUCT with *GSM8K*-style instruction and with *Ape210K*-style instruction. We use greedy decoding. The whitespace is edited for visual clarity.

id: ape210k__00678281

prediction with *GSM8K*-style instruction:

```
The altitude of Mount Fuji in Japan is 8844 − 1296 =
<gadget id="calculator">8844−1296</gadget>
<output>7_548</output>
7548 meters.
<result>7_548</result>
```

prediction with *Ape210K*-style instruction:

```
<gadget id="calculator">8_844 − 1_296</gadget>
<output>7_548</output>
<gadget id="calculator">7_548 / 2</gadget>
<output>3_774</output>
<result>3_774</result>
```

id: ape210k__00083872

prediction with *GSM8K* style instruction:

```
The volume of a cylinder is given by pi ∗ r ∗ h.
The volume of the iron ball is given by pi ∗ 12 ∗ 12 ∗ 14 =
<gadget id="calculator">12∗12∗12∗14</gadget>
<output>24_192</output>
24192 cubic decimeters.
<result>24_192</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">12 ∗∗ 2</gadget>
<output>144</output>
<gadget id="calculator">14 − 13.5</gadget>
<output>0.5</output>
<gadget id="calculator">3.14 ∗ 144 ∗ 0.5</gadget>
<output>226.08</output>
<result>226.08</result>
```

id: ape210k__01103039

prediction with *GSM8K* style instruction:

```
The original price is 8+2=
<gadget id="calculator">8+2</gadget>
<output>10</output>
10 yuan. So, he bought the pencil case for 8/10=
<gadget id="calculator">8/10</gadget>
<output>4/5 = around 0.8</output>
0.8 yuan less than the original price.
Therefore, the percentage that Xiaolong bought the pencil case
cheaper than the original price is 0.8/10∗100=
<gadget id="calculator">0.8/10∗100</gadget>
<output>8</output>
```

```
8%
<result>8</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">8 + 2</gadget>
<output>10</output>
<gadget id="calculator">2 / 10</gadget>
<output>1/5 = around 0.2</output>
<result>1/5 = around 0.2</result>
```

id: ape210k__00140349

prediction with *GSM8K* style instruction:

```
If 2/15 of the canal's length is 60 meters, then the canal is
2/15*60 =
<gadget id="calculator">2/15*60</gadget>
<output>8</output>
8 meters long.
<result>8</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">2 / 15</gadget>
<output>2/15 = around 0.133333</output>
<gadget id="calculator">60 / (2/15)</gadget>
<output>450</output>
<result>450</result>
```

id: ape210k__01034935

prediction with *GSM8K* style instruction:

```
The total number of words that Xiaopang can type in 20 minutes is
20 * 60 =
<gadget id="calculator">20*60</gadget>
<output>1_200</output>
1200 words. The number of words that Xiaoqiao can type in 15
minutes is 1200/15 =
<gadget id="calculator">1200/15</gadget>
<output>80</output>
80 words.
<result>80</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">20 / 15</gadget>
<output>4/3 = around 1.333333</output>
<gadget id="calculator">60 * (4/3)</gadget>
<output>80</output>
<result>80</result>
```

id: ape210k__00627727

prediction with *GSM8K* style instruction:

```
The number of tourists who were received on the second day is 1/6
* 480 =
<gadget id="calculator">1/6*480</gadget>
<output>80</output>
```

```
80 more than on the first day. The total number of tourists who
were received on the second day is 480 + 80 =
<gadget id=”calculator”>480+80</gadget>
<output>560</output>
560.
<result>560</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id=”calculator”>1 / 6</gadget>
<output>1/6 = around 0.166667</output>
<gadget id=”calculator”>1 + (1/6)</gadget>
<output>7/6 = around 1.166667</output>
<gadget id=”calculator”>480 * (7/6)</gadget>
<output>560</output>
<result>560</result>
```

id: ape210k__00349226

prediction with *GSM8K* style instruction:

```
There are 123 + 78 =
<gadget id=”calculator”>123+78</gadget>
<output>201</output>
201 students participating in art training. Mathematics training
is twice the number of students in art training, meaning 201 x 2 =
<gadget id=”calculator”>201*2</gadget>
<output>402</output>
402 students.
<result>402</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id=”calculator”>123 + 78</gadget>
<output>201</output>
<gadget id=”calculator”>201 * 2</gadget>
<output>402</output>
<result>402</result>
```

id: ape210k__00047506

prediction with *GSM8K* style instruction:

```
Twice the number of participating countries in the 2005 Aichi
World Expo is 121*2 =
<gadget id=”calculator”>121*2</gadget>
<output>242</output>
242. So far, 242 - 4 =
<gadget id=”calculator”>242-2</gadget>
<output>240</output>
240 countries have participated in the Shanghai World Expo.
<result>240</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id=”calculator”>121 * 2</gadget>
<output>242</output>
<gadget id=”calculator”>242 - 4</gadget>
<output>238</output>
<result>238</result>
```

id: ape210k__00376600

prediction with *GSM8K* style instruction:

```
First , we need to determine how many tons of the cargo are left .
To do this , we perform 119/5=
<gadget id="calculator">119/5</gadget>
<output>119/5 = around 23.8</output>
23.8 tons of cargo in 1 truck. Since the next vehicle added to the
 group has a load capacity of 4.5 tons, we divide the 23.8 tons by
 4.5, performing 23.8/4.5=
<gadget id="calculator">23.8/4.5</gadget>
<output>5.288889</output>
5.288889 trucks in 1 day. Since the next truck is able to
transport another 5 tons, we complete the shipment by taking out
5+2=
<gadget id="calculator">5+2</gadget>
<output>7</output>
7 trucks. Since the remaining cargo is 119−76=
<gadget id="calculator">119−76</gadget>
<output>43</output>
43 tons, we do this by taking 43/4.5=
<gadget id="calculator">43/4.5</gadget>
<output>9.555556</output>
9.555556 trucks in 1 day. Since the next truck is able to
transport another 5 tons, we do this by taking 9.555556=
<gadget id="calculator">9.555556</gadget>
<output>9.555556</output>
9.555556 trucks in 1 day. Since the final truck is able to
transport another 4.5 tons, we do this by taking 9.555556=
<gadget id="calculator">9.555556</gadget>
<output>9.555556</output>
9.555556. Since the final truck is able to transport another 4.5
tons, we do this by taking 9.555556=
<gadget id="calculator">9.555556</gadget>
<output>9.555556</output>
9.555556. Since the final truck is able to transport another 4.5
tons, we do this by taking 9.555556=
<gadget id="calculator">9.555556</gadget>
<output>9.555556</output>
9.555556.
Since the final truck is able to transport a total of 9.555556, we
 do this by subtracting 9.555556 from 119.5788889 to find out the
number of additional trucks that will be used to transport the
remaining cargo.
<result>9</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">119 − 5</gadget>
<output>114</output>
<gadget id="calculator">5 + 4.5</gadget>
<output>9.5</output>
<gadget id="calculator">114 / 9.5</gadget>
<output>12</output>
<result>12</result>
```

id: ape210k__00288459

prediction with *GSM8K* style instruction:

```
50 - 2 =
<gadget id="calculator">50-2</gadget>
<output>48</output>
48 parts are good. The pass rate is 48/50 =
<gadget id="calculator">48/50</gadget>
<output>24/25 = around 0.96</output>
0.96. Uncle Wang should process 192 / 0.96 =
<gadget id="calculator">192/0.26</gadget>
<output>738.461538</output>
738.461538 parts.
<result>738.461538</result>
```

prediction with *Ape210K* style instruction:

```
<gadget id="calculator">50 - 2</gadget>
<output>48</output>
<gadget id="calculator">48 / 50</gadget>
<output>24/25 = around 0.96</output>
<gadget id="calculator">192 / (24/25)</gadget>
<output>200</output>
<result>200</result>
```

E Offline self-training convergence speed

Method	Training steps
Supervised baseline	16000
Supervised balanced	14000
Supervised with-neg	20000
KTO $\beta = 0.3$	3800
KTO $\beta = 0.1$	4800
KTO $\beta = 0.1$ LoRA	16400
DPO $\beta = 0.99$	1800
DPO $\beta = 0.9$	1800
DPO $\beta = 0.9$ LoRA	2600
IPO $\tau = 0.9$	1200
IPO $\tau = 0.99$	1200
IPO $\tau = 0.99$ LoRA	1600

Table 4: Comparison of number of steps needed for self-training methods to convergence.

F Details of the online self-training pipeline

Supervised setting: For each problem, 16 solutions are generated with the current model using top-k=50 sampling. Solutions with a correct result will be used for training, the rest are discarded. The correct solutions are oversampled (at most 4 times each) - to generate 32 training samples per problem.

Preference optimization setting: For each problem, 16 solutions are generated with the current model using top-k=50 sampling. Then, we generate all possible pairs of solutions where one solution has a correct result and the other one does not. We then sample with repetition from the pairs, such that (i) every correct solution is used at most 4 times, (ii) the number of training samples (preference pairs) per problem is 32 if possible without violating condition (i), and (iii) all correct (and all incorrect) solutions are used almost the same number of times.

In both cases (supervised and preference optimization), the generated data examples are put into a buffer of size 8192 and are sampled from it for training. When a training example gets chosen, it is removed from the buffer, and new data are generated to fill the empty slot.

G Datasets, Models, and Code

We make our code, datasets, and models publicly available.

Code: `https://github.com/prompteus/calc-x`

Models: `https://hf.co/MU-NLPC`

Datasets:

- Calc-X: `https://hf.co/datasets/MU-NLPC/Calc-X`
- Style instructions: `https://hf.co/datasets/MU-NLPC/Calc-X_style-instructions`
- Offline self-training predictions: `https://huggingface.co/datasets/MU-NLPC/Calc-ape210k_selftrain`
- Offline self-training dataset variants used in experiments:
  - `https://hf.co/datasets/MU-NLPC/Calc-ape210k_selftrain_experiment`
  - `https://hf.co/datasets/MU-NLPC/Calc-ape210k_selftrain_experiment_balanced`
  - `https://hf.co/datasets/MU-NLPC/Calc-ape210k_selftrain_experiment_negative`