RetICL: Sequential Retrieval of In-Context Examples with Reinforcement Learning

Anonymous ACL submission

Abstract

Recent developments in large pre-trained language models have enabled unprecedented performance on a variety of downstream tasks. Achieving best performance with these models often leverages in-context learning, where a model performs a (possibly new) task given one or more examples. However, recent work has shown that the choice of examples can have a large impact on task performance and that finding an optimal set of examples is non-trivial. While there are many existing methods for selecting in-context examples, they generally score examples independently, ignoring the dependency between them and the order in which they are provided to the model. In this work, we propose Retrieval for In-Context Learning (RetICL), a learnable method for modeling and optimally selecting examples sequentially for in-context learning. We frame the problem of sequential example selection as a Markov decision process and train an example retriever using reinforcement learning. We evaluate RetICL on math word problem solving and scientific question answering tasks and show that it consistently outperforms or matches heuristic and learnable baselines. We also use case studies to show that RetICL implicitly learns representations of problem solving strategies.

1 Introduction

With the rising prominence of large pre-trained language models (LLMs), prior work has focused on how to best utilize them for various natural language tasks. One of the most popular methods for doing so is prompt tuning, which deals with carefully selecting the natural language prompt that maximizes model performance (Liu et al., 2021b). While there are many approaches to prompt tuning, a very successful one is in-context learning (ICL) (Brown et al., 2020). In ICL, examples of a new task that the LLM may not have been trained on before are included in the prompt, enabling it to leverage patterns in these examples in a few-shot way. However, the choice of which examples the LLM sees for a particular task can significantly affect the model's performance (Zhao et al., 2021).

The primary goal of ICL example selection is to find examples that, when used in the prompt, elicit a desired response from an LLM. Common practice is to define a function to measure the quality of a set of examples as $\phi(x, e_1, \ldots, e_T)$, where x is the input for the current task and $e_1, \ldots, e_T$ is a list of T examples drawn from a corpus $\mathcal{C}$, and use ϕ to rank candidate examples. Most existing works assume that examples work independently of each other, i.e., $\phi(x, e_1, \ldots, e_T) = \prod_{t=1}^{T} \phi'(x, e_t)$. Thus, one can find the best set of examples by selecting the top T examples in the corpus with the highest values of $\phi'(x, e_t)$, which is often the semantic similarity between x and e_t. However, there is significant interplay between the roles of different examples in deciding the output of LLMs. Some tasks benefit from example diversity (Su et al., 2022), while others benefit from combining specific information across examples (Levy et al., 2023). In these cases, simply selecting the top-T ranked examples may neglect ones that are ranked lower on their own but are useful in conjunction with other examples. Additionally, top-T selection ignores the *order* in which examples are provided as LLM input, which also has an impact on its output (Lu et al., 2021). See Section 2 for a more detailed discussion of related work on ICL example selection.

1.1 Contributions

In this work, we propose RetICL (Retrieval for In-Context Learning), a fully learnable method that sequentially retrieves ICL examples by conditioning on both the current problem and examples that have already been selected. We frame the problem of sequential example selection as a Markov decision process (MDP) and train an example retriever model using reinforcement learning (RL). We construct the model using a recurrent architecture

where hidden states act as latent representations of MDP states, and model the example ranking function using a bilinear transformation between the latent and corpus spaces, enabling efficient inference-time maximization of $\phi(x, e_1, \ldots, e_T)$. We also propose a novel *confidence* reward function, which uses the perplexity of the *generated* solution to help guide training. We validate RetICL on the math word problem (MWP) solving datasets TabMWP and GSM8K where it outperforms or matches both heuristic and learnable baselines. Additionally, to test RetICL's ability to generalize across domains, we validate it on the scientific question answering dataset QASC, where RetICL outperforms all baselines. Finally, we qualitatively analyze RetICL's learned policies and find that RetICL is able to implicitly infer problem solving strategies while learning its ICL example selection policy. We intend to make our implementation publicly available.

2 Related Work

In ICL, it is common to either randomly select in-context examples (Brown et al., 2020; Lewkowycz et al., 2022) or use a hand-crafted set of examples (Hendrycks et al., 2021; Wei et al., 2023). However, it is now well known that example selection and ordering can have a large impact on downstream text generation performance (Gao et al., 2020; Liu et al., 2021a; Zhao et al., 2021; Lu et al., 2021). There are many existing methods for in-context example selection that focus on different aspects of the problem. Several seek to maximize how much coverage a set of examples provides over the dataset and use diversity to measure coverage (Levy et al., 2023; Su et al., 2022; Pitis et al., 2023). Other methods consider semantic features of the examples (Liu et al., 2021a; Fu et al., 2022) or compare these features with outputs of the target LLM (Qin et al., 2023). Chang and Jia (2023) use random prompting but with a curated corpus based on how well examples perform on a validation set, and Rubin et al. (2021) use contrastive learning to retrieve examples that are likely to have similar labels to the target. While each of these methods tends to tackle a single aspect of example selection, we distinguish our method by combining several of these aspects, particularly considering groups of examples (including ordering) and using the LLM's outputs for a training signal. There are other works that use RL for ICL example selection (Lu et al., 2022; Zhang et al., 2022), although they either do not include previously selected examples in the state or only use high-level features of the examples, while RetICL uses their exact textual content.

3 Methodology

In this section, we detail how we frame ICL example selection as an MDP, how our example retriever model works, and how we train it using RL. We show an overview of our methodology in Figure 1.

3.1 MDP Formulation and Reward Function

We can view ICL example selection as a sequential decision making problem, where we select examples one at a time in such a way that we maximize our chances of achieving some goal when the examples are used as context. Our goal is to maximize $r(\mathcal{M}(x, e_1, \ldots, e_T), y)$, where $\mathcal{M}(\cdot)$ returns the generated output of an LLM given a prompt, y is the label corresponding to x, and r is a task-specific function that returns how good the generated output is. We note that in this setup, the order in which examples are selected matters since the order in which they are provided to $\mathcal{M}$ must be defined. We also note that while in this work we set T to a constant, it can also be dynamically set during the decision-making process, which we leave for future work. With this framing, we can naturally define an MDP where the state at time step t corresponds to both x and the first t examples that have been selected, and the action space is the set of potential candidates to be the next example. Formally,

$$S_0 = x, \quad S_t = x, e_1, \ldots, e_t, \quad A_t = e_{t+1} \in \mathcal{C}.$$

We now define the reward function for the MDP, which we break into two parts: a task-specific **goal** reward, and a supplementary **confidence** reward. We define the goal reward, R^G, simply as the output of r, as long as it can be formulated to return a scalar value. In settings with definitive correct and incorrect answers, it is natural for r to be binary, where it returns 1 when the generated solution results in a correct answer and -1 when the generated solution results in an incorrect answer. However, with chain-of-thought (CoT) prompting (Wei et al., 2023), this reward function does not account for the reasoning strategy in the solution process that led to the final answer and thus cannot distinguish between sound and flawed logic. To address this issue, we introduce the confidence reward, R^C, which we define as the inverse perplexity of the generated

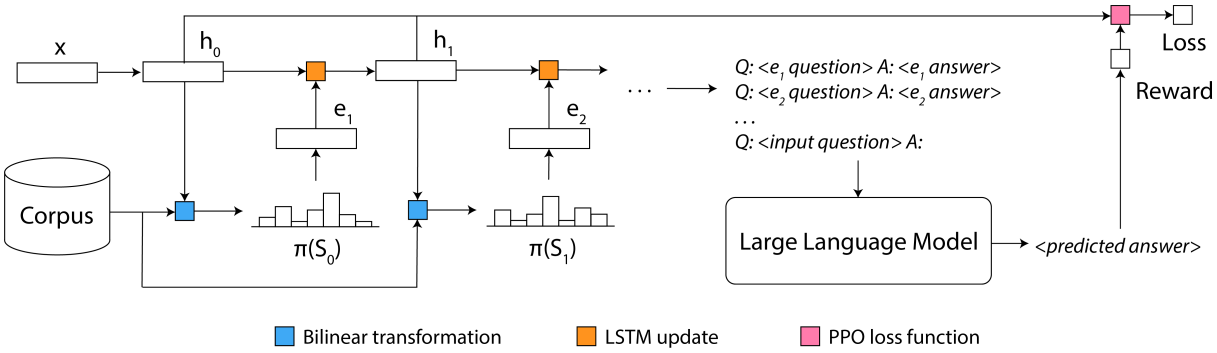

Figure 1: RetICL overview for question answering. Each latent state is constructed from the problem and previously selected examples. The next example is selected using a bilinear transformation between the latent state and examples in the corpus. After all examples are selected, we query the LLM, obtain a reward, and update the policy.

solution assigned by the LLM, normalized to the range $[-1, 1]$. We hypothesize that when an LLM generates a correct solution with high probability (low perplexity), it is likely that the model "knew" how to solve the problem, rather than getting it correct by guessing or using unsound reasoning to arrive at a final answer. Additionally, we hypothesize that when an LLM generates an incorrect solution with high probability, it may have sound reasoning overall but contain a small error, such as an incorrect calculation. Recent works have also found that model confidence is predictive of downstream performance (Tian et al., 2023; Kadavath et al., 2022). We define the final reward function to be the average of R^G and R^C at the final time step and 0 at all prior time steps. We formally define our reward function as

$$\hat{y} = \mathcal{M}(x, e_1, \ldots, e_T),$$
$$R^G = r(\hat{y}, y) = 2 \cdot \mathbb{I}[g(\hat{y}, y)] - 1,$$
$$R^C = 2 \cdot p_{\mathcal{M}}(\hat{y}|x, e_1, \ldots, e_T)^{\frac{1}{|\hat{y}|}} - 1,$$
$$R_t = \begin{cases} 0.5R^G + 0.5R^C & \text{if } t = T \\ 0 & \text{otherwise,} \end{cases}$$

where $\hat{y}$ is the generated solution, g is a function that checks if two solutions have the same final answer, $\mathbb{I}$ is the indicator function, and $p_{\mathcal{M}}$ returns the probability assigned by the LLM.

3.2 Retriever Model

We now detail our model for example retrieval. At a high level, the model constructs a latent representation for each state S_t in the MDP and uses this representation to construct the policy $\pi(S_t, e)$, which represents the probability of selecting e to be the next example. After using the policy to select an example, we add it to the current sequence of examples, giving us the state S_{t+1}, and the process continues sequentially.

We use a long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997) as the base model, where the hidden state $\mathbf{h}_t$ acts as the latent representation for S_t. We construct the initial hidden state of the LSTM, $\mathbf{h}_0$, using a vectorized embedding of the input x, and set the input of the LSTM at time step t to be a vectorized embedding of the example e_t. In this work, we construct these vectorized embeddings using a pre-trained S-BERT model (Reimers and Gurevych, 2019) and additionally provide learnable soft prompts (Lester et al., 2021) to S-BERT to help align the embeddings with the current task. In our experiments, we found that fine-tuning the S-BERT parameters directly did not improve performance.

We produce the policy $\pi(S_t, e)$ by first producing an unnormalized *activation* value for each example in the corpus, $\phi(S_t, e)$, and then using the softmax function to convert these activations into a probability distribution. We construct each $\phi(S_t, e)$ by performing a learnable bilinear transformation between $\mathbf{h}_t$ and the vectorized embedding of e. We choose to model ϕ using a bilinear transformation for two reasons. First, the bilinear transformation learns a mapping between the model's latent space and the example embedding space, enabling generalization to examples not seen during training and also adding some model interpretability, as we will show later in this paper. Second, the bilinear transformation enables efficient computation of the policy over a large corpus at inference time, which we describe in detail in Supplementary Material C. We additionally use $\mathbf{h}_t$ to produce an estimate of the value function, $\hat{v}(S_t)$, which is required for variance reduction techniques when training policy

gradient methods. Concretely, our model architecture is defined as

$$\mathbf{x} = \text{S-BERT}(\mathbf{P}_x, x), \quad \mathbf{e} = \text{S-BERT}(\mathbf{P}_e, e),$$
$$\mathbf{h}_0 = \tanh(\mathbf{W}_x \mathbf{x} + \mathbf{b}_x),$$
$$\mathbf{h}_{t>0} = \text{LSTM}(\mathbf{h}_0; \mathbf{e}_1, \ldots, \mathbf{e}_t),$$
$$\hat{v}(S_t) = \mathbf{h}_t^T \mathbf{w}_v + b_v,$$
$$\phi(S_t, e) = \begin{cases} \mathbf{h}_t^T \mathbf{W}_a \mathbf{e} & \text{if } e \notin \{e_1, \ldots, e_t\} \\ -\infty & \text{otherwise,} \end{cases}$$
$$\pi(S_t, e) = \exp(\phi(S_t, e)) / \textstyle\sum_{e' \in \mathcal{C}} \exp(\phi(S_t, e')),$$

where $\mathbf{P}_x \in \mathbb{R}^{m \times d_e}$ and $\mathbf{P}_e \in \mathbb{R}^{m \times d_e}$ are learnable soft prompts, $\mathbf{W}_x \in \mathbb{R}^{d_h \times d_e}$ and $\mathbf{b}_x \in \mathbb{R}^{d_h}$ transform the input embedding space into the latent space, $\mathbf{w}_v \in \mathbb{R}^{d_h}$ and $b_v \in \mathbb{R}$ produce the value function estimate from the latent space, $\mathbf{W}_a \in \mathbb{R}^{d_h \times d_e}$ performs the bilinear transformation between the latent space and example embedding space, m is the soft prompt length, d_e is the dimension of the S-BERT text embedding vector, and d_h is the size of the LSTM's hidden states. We set $\phi(S_t, e)$ to $-\infty$ when e has already been selected to avoid selecting the same example multiple times, which is in line with existing methods.

3.3 Training and Inference

We train the retriever model using proximal policy optimization (PPO) (Schulman et al., 2017) with generalized advantage estimation (GAE) (Schulman et al., 2015) as our advantage function. We use a reward discount of $\gamma = 1$ since all episodes have fixed length and the reward is assigned only at the final time step. We train the value function estimator using mean squared error (MSE) with R_T as the target at each time step and weigh the value function loss with a hyperparameter c_{VF}. We also encourage exploration by adding the negative entropy of the policy at each time step to the loss (Ahmed et al., 2019), where we additionally weigh the entropy by a hyperparameter c_{E} and normalize by a factor of $\frac{1}{\log(|\mathcal{C}|)}$ to account for training with different corpus sizes.

At training time, we select a batch of problems from the dataset, encode their inputs with S-BERT, encode all examples in the corpus with S-BERT, and then construct a sequence of examples for each problem by sequentially sampling from the policy, i.e., $e_{t+1} \sim \pi(S_t, \cdot)$. When T examples have been selected for each problem, we prompt the LLM with the examples and the current problem's input, calculate the reward from the LLM's generations, average the PPO loss, value function loss, and entropy loss over the batch, and backpropagate through our model. At inference time, we greedily select examples from the policy as $e_{t+1} = \text{argmax}_{e \in \mathcal{C}} \pi(S_t, e)$, which empirically outperforms sampling in our experiments.

4 Experiments

In this section, we validate RetICL on MWP solving and scientific question answering tasks and quantitatively compare its performance against several baselines. We also perform an ablation study and examine the effects of adjusting several parameters in order to determine which aspects of the methodology work well and which ones need improvement in future work.

4.1 Datasets

We validate RetICL on two MWP datasets that contain detailed solution steps: **TabMWP** (Lu et al., 2022), where solving each problem requires extracting and reasoning with information from tables, and **GSM8K** (Cobbe et al., 2021), where solving each problem requires multi-step mathematical reasoning and applying various arithmetic operations. In order to test whether RetICL generalizes beyond MWP solving, we also validate on **QASC** (Khot et al., 2019), which contains multiple choice science questions along with explanations for each answer, where answering each question requires logical deduction and leveraging world knowledge.

For all datasets, we use CoT prompting and evaluate correctness based on the final answer of the generated solution. We note that the official TabMWP evaluation code has some issues with regular expressions that cause both false positives and false negatives when evaluating correctness on multiple choice problems. We instead use our own, fixed code to evaluate correctness on TabMWP. We provide additional information on each dataset and the prompts we use in Supplementary Material B.

4.2 Experimental Settings

For each dataset, we randomly select 5,000 problems for training and an additional 200 problems for the corpus, both from the original training set. While it is possible to use a larger corpus, e.g., all remaining problems in the training set, we find that training on a smaller corpus results in higher accuracy. We randomly select 500 problems from the validation set to evaluate on after each epoch

and save the model with the highest accuracy on this set. For validation and testing, we use the entire training set as the corpus. We use OpenAI's *code-davinci-002* Codex model (Chen et al., 2021) as the LLM for all experiments since it is free and works well on our task, and we leave experimentation on larger, more expensive models for future work. We set the number of in-context examples to $T = 2$ for all experiments to enable a fair comparison to other methods (Lu et al., 2022). We note that while increasing T tends to increase performance for heuristic methods, we find that RetICL's performance does not tend to increase with T as expected. We provide details on the impact of T in Supplementary Material E and leave further exploration of increasing T for future work. We provide additional hyperparameter settings and implementation details in Supplementary Material A.

4.3 Baselines

We compare RetICL to the following baselines for in-context example selection, comprising both heuristic-based and learnable methods. We also estimate a performance upper bound by exhaustively checking all possible example combinations.

Random With random selection, for each problem, we randomly sample T unique examples from the corpus for the ICL prompt. We evaluate random selection on 3 random seeds and report the average accuracy across all 3 runs.

kNN With kNN selection (Liu et al., 2021a), for each problem, we select the T examples with the most similar problem inputs from the corpus and use those for the ICL prompt, putting more similar examples later in the prompt due to recency bias (Zhao et al., 2021). We evaluate similarity according to the Euclidean distance between the S-BERT embeddings of the problem inputs using the same pre-trained S-BERT model as RetICL.

Complexity With complexity-based selection (Fu et al., 2022), for each problem, we randomly select T examples that have the most *complex* reasoning in the label for the ICL prompt. For TabMWP and GSM8K, we define complexity as the number of steps in the solution, and for QASC we define it as the number of words in the label.

PromptPG With PromptPG (Lu et al., 2022), for each problem, a learned scoring function is evaluated on each individual example in the corpus, and the top T scoring examples are selected for the ICL prompt. While PromptPG also uses RL to learn its example selection policy, there are many key differences between their method and ours: they do not include previously selected examples in the state, they do not use a confidence-based reward, and they use a much simpler RL framework. We evaluate PromptPG's performance by running their code with modifications to match our prompting style and use our fixed evaluation code.

LSTM Classifier In order to determine the effectiveness of RetICL's training pipeline, we train a model with a similar architecture but in a supervised manner. Specifically, using the same input format as RetICL, we train an LSTM classifier to predict, at each time step, if the prompt will result in a correct or incorrect response from an LLM. We train on 20 randomly sampled prompts for each sample in the training set with the same training set and corpus as RetICL. At inference time, we greedily select examples that result in the highest predicted likelihood of getting a correct response. We note that this setup is equivalent to estimating the Q-function of a random policy, and provide further details in Supplementary Material D.

Exhaustive With exhaustive evaluation, for each problem, we construct a one-shot ICL prompt for each example in the corpus, and consider the current problem to be solved if a correct solution is generated from any of the prompts. We use one-shot prompts instead of few-shot prompts to reduce the search space. Additionally, we restrict the corpus size to 100 and only evaluate on the pre-defined 1,000-sample subset of the test set for TabMWP to reduce computation time.

4.4 Results

Table 1 shows the performance of RetICL and baselines on all datasets, with **Acc.** being problem solving accuracy and **Ex.** being the number of unique examples used for each method. We see that RetICL performs the best among non-exhaustive methods on all datasets, beating most baselines by a large margin, with the exception of TabMWP where kNN performs slightly better. We note that while the relative performance of baselines seems to depend on the dataset, RetICL performs well across datasets, showing that our methodology is more generalizable across tasks.

We note that kNN likely performs well on TabMWP due to the presence of many problems in the dataset with very high similarity. Many problems have exactly the same question text other than

Method	TabMWP		GSM8K		QASC	
	Acc.	Ex.	Acc.	Ex.	Acc.	Ex.
Exhaustive	98.30	37	97.95	47	98.49	36
Random	72.04	11,203	57.19	2,153	70.41	1,635
kNN (Liu et al., 2021a)	**88.95**	10,003	59.74	1,883	61.99	964
Complexity (Fu et al., 2022)	63.80	281	54.66	13	74.19	3
PromptPG (Lu et al., 2022)	73.43	7	56.94	8	73.65	2
LSTM Classifier	77.21	13	64.82	4	69.65	8
RetICL	88.58	407	**66.11**	97	**76.13**	135

Table 1: Test set accuracy and examples selected for RetICL and baselines.

a few numbers or names changed, making it easy for the LLM to generate a correct solution given a highly similar example. On the contrary, GSM8K does not tend to contain problems that are almost identical, which makes kNN ineffective since problems with high textual similarity may not have similar solution strategies. Additionally, kNN performs worse than Random on QASC since we find that using examples that are highly similar to the current question can cause the LLM to copy answers from the examples and ignore the details of the current question. Furthermore, we see that Complexity is a poor heuristic for TabMWP since it does not account for problem similarity, and also for GSM8K since it uses some examples with overly abstract reasoning that appear to confuse the model.

Perhaps surprisingly, we see that PromptPG is only slightly better than Random on TabMWP and performs on par with Random on GSM8K. And while PromptPG is better than Random and kNN on QASC, we observe that it selects the same examples for each question, thus lacking example diversity. While the results on TabMWP contradict the trends reported in (Lu et al., 2022), we believe the discrepancy is mostly due to using the fixed evaluation code. We believe that PromptPG's relatively low performance also highlights the challenges of solving the ICL example selection problem using RL; in practice, many training tricks are necessary to achieve high performance.

We see that the LSTM Classifier performs relatively well on GSM8K but not on other datasets. We believe the reason is that TabMWP and QASC require more targeted example selection strategies, which are difficult to extract from random example prompts due to sparse coverage of the exponentially large action space. This result highlights the benefit of on-policy learning for example selection. We also see that the Exhaustive method achieves almost perfect accuracy on all datasets. We find this surprising, especially due to the fact that Exhaustive only uses a single ICL example and has access to a smaller corpus. This result implies that there is significant room for growth in ICL example selection methods and also implies that one-shot ICL has the potential to be extremely powerful as long as the example corpus is informative, even for challenging text generation tasks.

In order to determine how well RetICL can generalize to low-resource settings, we examine the effect of reducing the number of available examples at test time. We evaluate RetICL and kNN on TabMWP and GSM8K where we use 0.1%, 1%, 10%, and 100% of all examples as the corpus and show our results in Figure 2. For a clearer visualization, we show the *relative accuracy* of each method compared to RetICL's accuracy when the full corpus is available. For TabMWP, we evaluate on the pre-defined 1000-sample subset of the test set. We see that while performance tends to decrease with less available examples, RetICL still retains at least 90% of its performance with only 0.1% of examples available, suggesting that it is still effective in low-resource settings. Additionally, we see that while RetICL and kNN perform similarly at most corpus sizes, RetICL performs better at 0.1%, likely because there are not enough highly similar examples for kNN to leverage. Finally, we note that the irregular trend for GSM8K with kNN implies that semantic similarity of examples is not correlated with problem solving accuracy on this dataset, suggesting that while kNN is very effective for some tasks it is not as robust as RetICL.

4.5 Ablation study

We now examine the impact of various modeling and algorithmic choices via an ablation study. We train on 1,000 problems instead of 5,000 ($\mathbf{T_{1k}}$), we no longer use the confidence reward, R^C, and instead only use the goal reward, R^G (**Conf. Rew.**), we no longer condition on previously selected examples by removing the LSTM architecture and instead set the latent state for all time steps to be $\mathbf{h}_0$ (**LSTM**), we no longer include an entropy term

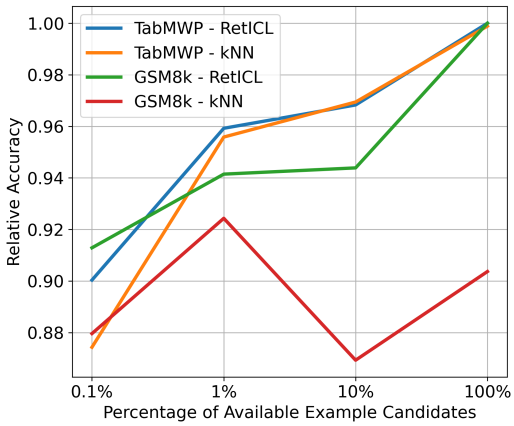

Figure 2: Change in relative accuracy as number of available example candidates increases.

Ablation	TabMWP		GSM8K	
	Acc.	Ex.	Acc.	Ex.
None	88.30	226	66.11	97
T_{1k}	87.30	159	65.96	34
T_{1k}, Conf. Rew.	86.00	84	64.67	20
T_{1k}, LSTM	85.10	120	63.91	38
T_{1k}, Ent.	79.90	15	62.77	6
T_{1k}, R	74.90	6	61.94	14
T_{1k}, RwB	85.30	104	66.19	5
T_{1k}, SP	88.40	163	66.26	3
T_{1k}, SP, TC_{20}	84.50	76	61.87	58
T_{1k}, SP, TC_{all}	85.50	122	65.88	58

Table 2: Ablation results on TabMWP and GSM8K.

in the loss function (**Ent.**), we no longer train with PPO and instead use REINFORCE (**R**) and REINFORCE with Baseline (**RwB**), we no longer provide learnable soft prompts to the S-BERT encoder (**SP**), and we vary the size of the corpus at train time, using a corpus with 20 problems (**TC_{20}**) from the training set and all remaining problems (**TC_{all}**) from the training set. Additionally, we use T_{1k} for all ablations for fast experimentation, and apply the SP ablation for the TC ablations since otherwise the partial gradients over S-BERT parameters will cause memory issues for TC_{all}.

Table 2 shows the results of the ablation study, which we run on both TabMWP and GSM8K. For TabMWP, we evaluate on the pre-defined 1,000-sample subset of the test set. We see that removing the confidence reward and LSTM architecture, which are our key contributions, both negatively impact accuracy, implying that these techniques have a positive impact on ICL example selection. We also see that removing the entropy term has a significant negative impact on accuracy and example diversity, implying that this term is necessary for training the model. Furthermore, we see that REINFORCE with Baseline is similar or slightly worse than PPO, while REINFORCE is significantly worse. Surprisingly, we see that removing soft prompts slightly increases accuracy, although significantly hurts example diversity on GSM8K, due to training instability. We hypothesize that soft prompts may make it harder to find an optimal policy due to increased model complexity, and plan on finding better ways to fine-tune S-BERT in future work. Finally, we see that using much smaller and much larger example corpora at train time both negatively impact accuracy, showing that corpus size is an important hyperparameter and confirming similar results in (Lu et al., 2022).

5 Qualitative Analysis

We now present several qualitative analyses to interpret RetICL's learned example selection policy. Our goal is to determine what features RetICL focuses on in individual examples and what strategy RetICL uses to select examples sequentially. We investigate these strategies by first visualizing learned latent example embeddings and then analyzing trends in per-problem example selections.

5.1 Latent Space Analysis

In order to identify features in the selected examples that are being emphasized by RetICL, we perform a visual analysis of the example embeddings in the model's latent space. Specifically, we transform each example embedding $\mathbf{e}$ into the model's latent space using the right half of the bilinear term in ϕ, i.e., $\mathbf{W}_a\mathbf{e}$. We note that since maximizing $\phi(S_t, e)$ is equivalent to maximizing the inner product $\langle \mathbf{h}_t, \mathbf{W}_a\mathbf{e} \rangle$, the most likely example to be selected is the one where $\mathbf{W}_a\mathbf{e}$ is closest to $\mathbf{h}_t$ in the latent space.[1] Therefore, analyzing local regions in the example embedding space reveals how RetICL's learned policy ranks examples.

For both TabMWP and GSM8K, we randomly select 1,000 examples from the corpus and then apply t-SNE (Van der Maaten and Hinton, 2008) to reduce their embeddings to 2 dimensions for visualization. Additionally, for the same sets of examples, we also visualize their pre-trained S-BERT embeddings in the same way in order to demonstrate how inter-example similarities change after RetICL training. We summarize our findings here and show the visualizations with a more detailed analysis in Supplementary Material F.

[1] Maximum inner product and minimum distance are equivalent in our case since $\mathbf{e}$ is normalized.

For TabMWP, the S-BERT embeddings are clustered based on problem template, since problems tend to fall into distinct structural and semantic categories. While RetICL mostly retains these clusters in the latent space, it also merges together some clusters of problems that can be solved with similar reasoning strategies, such as finding the largest or smallest value in a set. For GSM8K, the clusters are much less clear since problems cannot be easily placed into categories. However, while the S-BERT embeddings are generally grouped by problem topic, RetICL also groups them by the number of steps in a solution. This result implies that RetICL has learned that solution length is an important feature of ICL examples, and confirms findings from prior work that solution complexity impacts problem solving in LLMs (Fu et al., 2022). These findings suggest that RetICL can identify meaningful features, often related to solution strategy, that indicate an example's utility in ICL.

5.2 Per-Problem Example Selection

We now examine example selections at the per-problem level in order to gain further insight into RetICL's learned example selection policy.

Table 5 in Supplementary Material G shows the in-context examples selected to help solve a representative problem from the GSM8K dataset. We see that RetICL tends to select examples that share some unique high-level features with the current problem, such as subtracting from a total value, adding up values over some period of time, or defining variables to be proportional to other variables. We note that each problem in the dataset exhibits several such features, so RetICL has to implicitly decide which features are important to the current problem and identify examples with those features. We also see that RetICL tends to select examples with solutions that are relatively long and have numerous reasoning steps. Problem solving errors can be divided into several categories. First, the LLM can exhibit misconceptions when it lacks an example to provide context, such as misinterpreting the meaning of a "discount" when not explicitly instructed. Second, the LLM can try to follow the examples too closely and use reasoning that does not necessarily apply to the current problem. These errors indicate that RetICL's policy can be improved by selecting based on a broader and more targeted set of features. However, many incorrect solutions are caused by simple arithmetic errors or switching the roles of variables in the problem. We believe these errors are due to limitations of the LLM and are a likely source of noise in the training signal, making it harder to find an optimal policy. Such errors can be fixed by using self-consistency (Wang et al., 2023) or external computation engines (Wolfram, 2023), which we leave for future work.

Table 6 shows in-context examples selected for a problem from the TabMWP dataset. RetICL's selections tend to follow a surprising pattern: the first example is seemingly unrelated to the current problem, while the second example has similar reasoning steps to the current problem. This strategy has several implications. First, it suggests that RetICL can infer reasoning steps from the current problem and select examples based on this information. Second, it suggests that diversity of examples may help prevent overfitting to particular features in the examples. Problem solving errors tend to be caused by either RetICL selecting an unrelated second example or the second example being similar to the current problem but requiring a slightly different solution strategy. Therefore, RetICL can benefit from policy consistency regularizations and improvements to the retriever model to learn more accurate solution strategy representations.

6 Conclusions and Future Work

In this work, we proposed RetICL, a learnable method for sequential in-context example selection that, unlike existing methods that select all examples independently, takes previously selected examples into account. We framed the problem of sequential example selection as a Markov decision process and developed a novel reward function and example retriever model. We demonstrated that RetICL learns effective strategies for example selection and consistently performs well across math word problem and question answering tasks. There are many avenues for future work. First, we can explore RetICL's effectiveness on a wider array of tasks, including ones with open-ended goals. Second, we can explore other architectural modifications that could further improve the retriever model, such as using a Transformer instead of an LSTM. Third, since we used a fixed number of examples, we can extend RetICL to let it learn how many examples are needed. Fourth, we can explore whether RetICL can be applied to real-world educational settings, e.g., selecting worked examples to help students solve practice problems.

Limitations

We note that there are several practical limitations to our method. First, we note that RetICL can be expensive and time-consuming to train, with each of our main training runs requiring up to 250,000 LLM inferences. This high number of inferences makes training on paid models prohibitively expensive; for example, it could cost up to approximately $2,500 to train on OpenAI's *text-davinci* models. Additionally, newer OpenAI models, such as *gpt-3.5-turbo* and *gpt-4*, do not return likelihood information on generated text, making the confidence reward impossible to calculate for these models. We finally note that RetICL's performance not increasing with the number of examples in the prompt may limit its use in practical settings.

Ethical Considerations

We first note that the high number of inferences required to train RetICL give the method an outsized cost in terms of energy usage; however, we note that the method has a relatively low cost at inference time given its relatively low number of parameters and potential for optimization with MIPS. Additionally, we note that because RetICL uses a black-box LLM reward signal, its example selections are not guaranteed to be interpretable by humans. Finally, because we only experiment with question answering settings, we did not perform any analysis of bias in RetICL's selections. However, it is possible that RetICL could reflect biases in the LLM it is being trained on. As such, we recommend an analysis of bias in future works that use RetICL in sensitive settings such as student-facing educational tools.

References

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. 2019. Understanding the impact of entropy on policy optimization. In *International conference on machine learning*, pages 151–160. PMLR.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Ting-Yun Chang and Robin Jia. 2023. Data curation alone can stabilize in-context learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8123–8144, Toronto, Canada. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

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

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *CoRR*, abs/2012.15723.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt.

2021. Measuring massive multitask language understanding.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Alexander Jansen, and Ashish Sabharwal. 2019. Qasc: A dataset for question answering via sentence composition. In *AAAI Conference on Artificial Intelligence*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Itay Levy, Ben Bogin, and Jonathan Berant. 2023. Diverse demonstrations improve in-context compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1401–1422, Toronto, Canada. Association for Computational Linguistics.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021a. What makes good in-context examples for gpt-3?

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021b. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2022. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Silviu Pitis, Michael R. Zhang, Andrew Wang, and Jimmy Ba. 2023. Boosted prompt ensembles for large language models.

Chengwei Qin, Aston Zhang, Anirudh Dagar, and Wenming Ye. 2023. In-context learning with iterative demonstration selection. *arXiv preprint arXiv:2310.09881*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *arXiv preprint arXiv:2305.14975*.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Stephen Wolfram. 2023. Chatgpt gets its 'wolfram superpowers'! *Stephen Wolfram Writings*.

Yiming Zhang, Shi Feng, and Chenhao Tan. 2022. Active example selection for in-context learning.

Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models.

A Hyperparameters and Implementation Details

We implement the retriever model and RL algorithms in PyTorch. We note that our PPO implementation is slightly simpler than the standard implementation in that it does not use an inner training loop and instead takes a single training step on each batch sampled from the policy. We also note that we used GitHub Copilot to assist with minimal code-writing. We encode problem inputs and examples using the *all-distilroberta-v1* pre-trained S-BERT model with the *sentence-transformers* library (Reimers and Gurevych, 2019), take the normalized mean-pooled final layer outputs as the embeddings, and use a soft prompt length of 20. With Codex, we use greedy decoding and set the maximum number of generated tokens to 450/400/150 for TabMWP/GSM8K/QASC, respectively. We set the LSTM's hidden size to 800, PPO's ϵ to 0.1, GAE's λ to 0.9, and c_{VF} to 0.5. We set c_{E} to 0.05 for TabMWP and c_{E} to 0.1 for GSM8K, where different values are necessary since we find that our method performs differently across datasets and that c_{E} has a large impact on training stability. We use orthogonal initialization (Saxe et al., 2013) for all weight parameters, initialize all bias parameters to 0, and initialize soft prompts using a standard normal distribution. We train using the AdamW optimizer for 50 epochs with a learning rate of 0.001, a weight decay of 0.01, and a batch size of 20. We additionally apply gradient norm clipping on all parameters using a value of 2, which we find is critical to avoid spikes in training losses. We provide the values we experimented with in Table 3, where we selected values based on preliminary experiments aiming to optimize both accuracy and clock time. All final experiments were run on a *Lambda Vector* workstation with *NVIDIA RTX A6000* GPUs. With a single thread, training a RetICL model for TabMWP with 5,000 training samples and 500 validation samples for 50 epochs takes approximately 44 hours, and inference for 1,000 TabMWP samples takes approximately 10 minutes. We note that we decrease runtime by using a cache to avoid repeated LLM inferences at both train and test time, batching requests to OpenAI, and using an adaptive backoff for waiting in between OpenAI API calls in order to maximize throughput given the API rate limit. Finally, we note that a RetICL model with the above hyperparameters has approximately 5.5 million parameters.

We use *scikit-learn* for t-SNE and *matplotlib* for visualizing data. All software used in this work is either open source or does not specify a license. To the best of our knowledge, we are consistent with the terms and intended use of all software and services, particularly OpenAI.

Hyperparameter	Values Tried
Training size	1,000, **5,000**
Corpus size	20, **200**, 500, 1,000, all
Soft prompt length	5, 10, **20**, 40
LSTM hidden size	100, 200, 400, **800**, 1600
ϵ	**0.1**, 0.2
λ	**0.9**
c_{VF}	**0.5**, 1.0
c_{E}	0.01, **0.05**, **0.1**, 0.2, 0.5
Epochs	**50**, 100, 250
Learning rate	0.0001, 0.0005, **0.001**
Weight decay	0.001, **0.01**
Batch size	5, 10, **20**, 40, 64
Gradient clipping	0.5, 1.0, **2.0**

Table 3: Hyperparameter search space, where selected values are bold.

B Dataset Details

TabMWP has a pre-defined train/validation/test split of 23,059/7,686/7,686, GSM8K has a pre-defined train/test split of 7,473/1,319, and QASC has a pre-defined train/validation/test split of 8,134/926/920. We reserve 1,000 random problems from GSM8K's train set for validation. Because QASC's test set does not have labels, we instead use the validation set for testing and reserve 1,000 random problems from the train set for validation. All datasets are exclusively in English. TabMWP uses the CC BY-NC-SA license, GSM8K uses the MIT license, and QASC uses the Apache 2.0 license. We show our prompt templates, which we use for both S-BERT and the LLM, in Table 4.

C Inference-Time Optimization

We note that our formulation for ϕ enables efficient retrieval of the top-ranking example at each time step via maximum inner-product search (MIPS). We first note that $\phi(S_t, e)$ is maximized by finding the example e that maximizes the inner product $\langle \mathbf{h}_t, \mathbf{W}_a \mathbf{e} \rangle$. We can leverage this information by first pre-computing $\mathbf{W}_a \mathbf{e}$ for each example in the corpus and constructing a MIPS index over these vectors, using a library such as *faiss* (Johnson et al.,

Dataset	Template	Example
TabMWP	Table: [table] Problem: [problem statement] Solution: [solution steps] Final Answer: [final answer]	Table: [TITLE]: Siblings Number of siblings | Frequency 0 | 14 1 | 3 2 | 6 3 | 8 4 | 0 Problem: The students in Mr. Boyer's class recorded the number of siblings that each has. How many students have at least 2 siblings? Solution: Find the rows for 2, 3, and 4 siblings. Add the frequencies for these rows.\n\nAdd:\n\n6 + 8 + 0 = 14\n\n14 students have at least 2 siblings. Final Answer: 14
GSM8K	Problem: [problem statement] Solution: [solution steps] Final Answer: [final answer]	Problem: Marcos has to get across a 5 mile lake in his speedboat in 10 minutes so he can make it to work on time. How fast does he need to go in miles per hour to make it? Solution: 10 minutes for 5 miles means 10 minutes / 5 miles = 2 minutes/mile 1 hour is 60 minutes so 60 minutes/hour / 2 minutes/mile = 30 miles/hour Final Answer: 30
QASC	Question: [question] [options] Solution: [fact 1] [fact 2] [combined fact] Final Answer: [final answer]	Question: Mussels have what? (A) seaweed (B) arms (C) Energy (D) a shell (E) warmth (F) bacteria (G) Length (H) legs Solution: Most mollusks have shells. Mussels are bivalve mollusks. Mussels have shells. Final Answer: a shell

Table 4: Prompt template for a single in-context example and a formatted example for each dataset. We use the \n\n solution step delimiter for TabMWP to match the style in the original paper.

2019). At inference time, we can now leverage algorithms that perform approximate MIPS in sublinear time, i.e., maximize $\phi(S_t, e)$ without evaluating $\mathbf{h}_t^T \mathbf{W}_a \mathbf{e}$ for each example in the corpus. We note that we do not use MIPS in this work since the corpora we experiment on are sufficiently small such that evaluating $\mathbf{h}_t^T \mathbf{W}_a \mathbf{e}$ for each example is relatively inexpensive. However, we expect that significant computational time can be saved with MIPS when evaluating on corpora at much larger scales.

D LSTM Classifier Details

For our LSTM Classifier baseline, we use a similar architecture to RetICL but train using a supervised objective rather than RL. Specifically, we collect a dataset $\mathcal{D}$ of 20 prompts and resulting LLM outputs for each sample in the training and validation sets, where the examples in the prompts are randomly selected from the corpus. We feed samples $(x, e_1, \ldots, e_T) \in \mathcal{D}$ to an LSTM in the same way they are fed to RetICL. However, the LSTM classifier does not use a bilinear projection to define the policy $\pi(S_t, e)$, but rather uses a linear projection on LSTM hidden states to define an approximation of the Q-function $\hat{q}(S_t, e)$. We formalize this objective as

$$\hat{q}(S_{t-1}, e_t) = \mathbf{h}_t^T \mathbf{w}_q + b_q,$$

$$\ell = \sum_{t=1}^{T} (\hat{q}(S_{t-1}, e_t) - R^G)^2,$$

where $\mathbf{w}_q \in \mathbb{R}^{d_h}$ and $b_q \in \mathbb{R}$ are learnable parameters, and ℓ is the loss for a single sample in $\mathcal{D}$. At inference time, we use the approximated Q-function to guide an example selection policy via greedy decoding, i.e., $e_{t+1} = \text{argmax}_{e \in \mathcal{C}} \hat{q}(S_t, e)$.

We perform batched training on $\mathcal{D}$ and evaluate the *classification accuracy* on the validation set after each epoch. We define this accuracy as the percentage of samples where the estimated value of the final example in the prompt correctly predicts if $\hat{y}$ will be correct or not, i.e., $\text{sign}(\hat{q}(S_{T-1}, e_T)) = \text{sign}(R^G)$. We train using the AdamW optimizer for 20 epochs with a learning rate of 1e-4, a weight decay of 0.01, a batch size of 256, and perform early stopping based on the classification accuracy on the validation set. We use the same training set, validation set, and training corpus as RetICL. We do not use soft prompts for S-BERT in this set-

ting since we found that it hurts performance. We achieve classification accuracies of 85.54, 61.38, and 61.97 on TabMWP, GSM8K, and QASC, respectively. We note that these accuracies do not correlate with downstream performance when using the classifiers to guide example selection at inference time. This is likely because $\hat{q}$ is trained on a random policy but is used to guide a greedy policy at inference time, so there is an inherent difference between evaluations in these settings. Given these results, we can imagine a different inference time setting where the classifier is used to rank a list of randomly selected prompts rather than guiding a policy, but we leave this investigation for future work. Regardless, we believe these results further highlight the benefits of using RL algorithms for the example selection task.

E Scaling Number of In-Context Examples

We perform preliminary experiments on scaling T, the number of in-context examples used in each prompt. Specifically, with $T \in \{2, 3, 4, 5\}$, we calculate the test set accuracy on GSM8K using prompting via Random, kNN, and RetICL (trained on 1,000 samples), and show the results in Figure 3. We observe that while RetICL still performs better than baselines as T increases, the gap between the methods narrow, particularly because RetICL performs roughly the same with $T = 2$ as it does with higher values. While this result is unexpected, we note that finding an effective policy with a higher T is a more complex task due to increased degrees of freedom. Therefore, we hypothesize that RetICL's inability to capitalize on more examples is due to training difficulties, and possibly because of our simpler PPO implementation. We leave further investigations of this phenomenon and potential training improvements for future work.

F Latent Space Analysis Details

We show the visualized example embeddings in latent space in Figure 4. For GSM8K, we see that RetICL groups examples based on the number of solution steps, whereas the pre-trained S-BERT embeddings do not. We also see that clusters in the RetICL embeddings have been somewhat merged together from the pre-trained embeddings. This result can be interpreted by observing the pre-trained embeddings to be primarily clustered based on topic, e.g., problems about money and problems

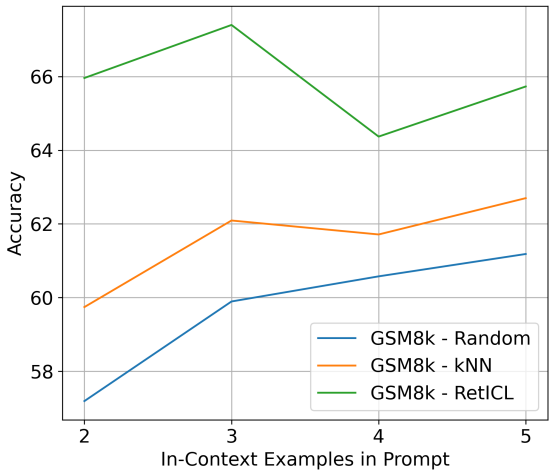

Figure 3: Accuracy as the number of in-context examples in the prompt increases, evaluated on GSM8K using prompting via Random, kNN, and RetICL.

about time belong to separate clusters, since S-BERT embeddings reflect the semantic content of the examples. While local neighbors in the RetICL embedding space also tend to have similar topics, the clusters are less well-separated, which implies that both topic and the solution strategy, which is partly reflected in the length of solution steps, are used for example selection by RetICL.

For TabMWP, we see that the space looks very different from GSM8K, with many separate clusters being present in both the RetICL and pre-trained spaces, primarily based on the problem's template. For example, there is one cluster for asking yes/no questions about schedules, one for asking if someone has enough money to buy something, and one for asking what the mean of a set of numbers is. Since RetICL retains these clusters, we can infer that an example's template is key to example selection. This observation is also validated by kNN's high performance on this dataset, since the most semantically similar problems are always from the same cluster. While there are not many differences between the RetICL and pre-trained S-BERT embedding spaces, we observe that RetICL has pulled several clusters closer together. For example, it partially merges together problems that require finding the largest value and problems that require finding the smallest value from a set. This observation suggests that problems across the merged template clusters can be used interchangeably as examples, since their problems tend to have similar reasoning strategies.

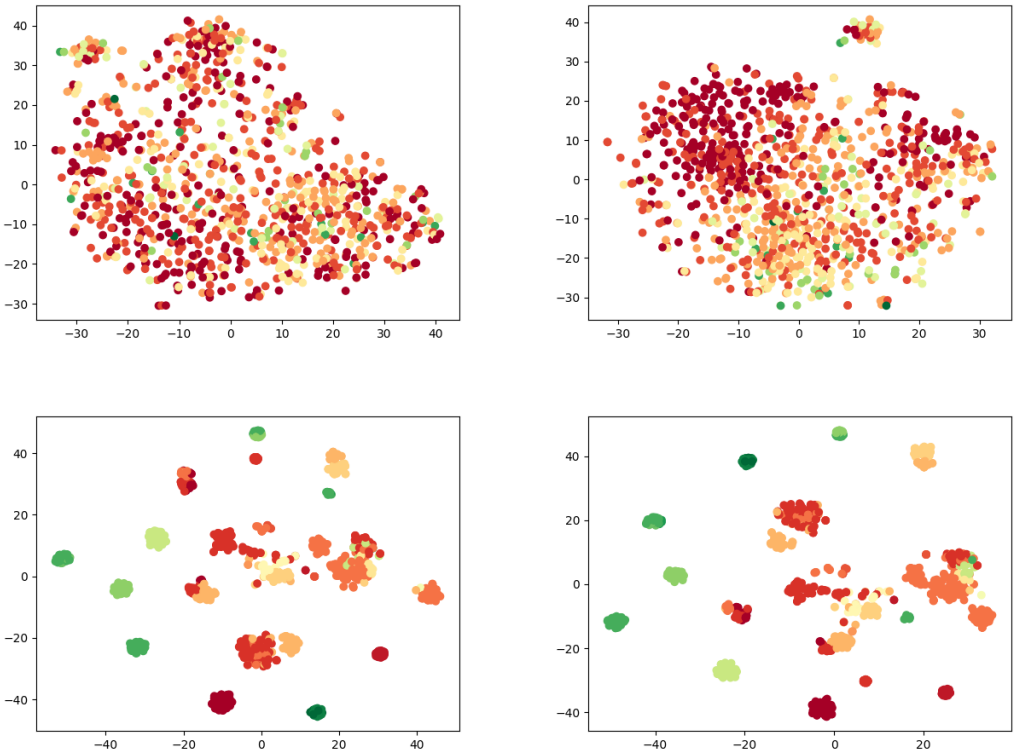

Figure 4: Example embedding visualizations. From left to right, top to bottom, GSM8K pre-trained embeddings, GSM8K RetICL embeddings, TabMWP pre-trained embeddings, and TabMWP RetICL embeddings. Points are colored based on the number of steps in an example's solution, with red being the least, green being the most, and yellow being in the middle.

G Representative Example Selections

Problem	
Marcus is half of Leo's age and five years younger than Deanna. Deanna is 26. How old is Leo?	
Gold Solution	
Marcus is 26 - 5 = 21 years old. Thus, Leo is 21 * 2 = 42 years old. Final Answer: 42	
RetICL	**kNN**
Selected Examples	
Problem: Katy, Wendi, and Carrie went to a bread-making party. Katy brought three 5-pound bags of flour. Wendi brought twice as much flour as Katy, but Carrie brought 5 pounds less than the amount of flour Wendi brought. How much more flour, in ounces, did Carrie bring than Katy? Solution: Katy's three 5-pound bags of flour weigh 3*5=15 pounds. Wendi brought twice as much flour as Katy, or 15*2=30 pounds of flour. Carrie brought 5 pounds less flour than Wendi, or 30-5=25 pounds of flour. Thus, Carrie brought 25-15=10 pounds of flour more than Katy. In ounces, the 10 pound difference is 10*16=160 ounces. Final Answer: 160 Problem: Vivian plays 10 Spotify songs every day. Her best friend Clara plays 2 fewer songs each day. If in June they didn't play any song during the weekends only, and there were 8 weekend days in June, what's the total number of songs they both listened to in that month? Solution: June has 30 days, so if they didn't play any songs on weekends, then they played 30 days - 8 days = 22 days. If Vivian played 10 songs each day, the total number of songs she played for the month is 10 songs/day * 22 days = 220 songs. Clara played 2 fewer songs, which is 10 songs/day - 2 songs/day = 8 songs/day. The total number of songs played by Clara for the whole month is 8 songs/day * 22 days = 176 songs. The total played by both is 220 songs + 176 songs = 396 songs Final Answer: 396	Problem: Leonard is 4 years younger than Nina who is half as old as Jerome. If the sum of their ages is 36, what is Leonard's age? Solution: Let x be the age of Nina. So, Leonard's age is x - 4. And Jerome's age is 2x. Since the sum of their ages is 36, then the equation is x + x - 4 + 2x = 36. By combining like terms, the equation becomes 4x = 40. So the value of x which represents the age of Nina is x = 40/4 = 10. Therefore, Leonard is 10 - 4 = 6 years old. Final Answer: 6 Problem: In 6 years, Carla will be 30 years old. The sum of the current ages of Carla and Louis is 55. How old is Louis now? Solution: Carla is currently 30-6=24 years old. Louis is 55-24=31 years old at present. Final Answer: 31
Generated Solutions	
Leo is twice Marcus' age, so Leo is 2*Marcus. Deanna is 5 years younger than Marcus, so Marcus is Deanna - 5. Deanna is 26, so Marcus is 26 - 5 = 21. Leo is 2*Marcus, so Leo is 2*21 = 42. Final Answer: 42 ✓	Leo is 26+5=31 years old. Marcus is 31/2=15.5 years old. Final Answer: 31 ✗

Table 5: Problem from GSM8K with examples and generated solutions from RetICL and kNN, where RetICL examples lead to a correct solution and kNN examples lead to an incorrect solution. A common feature between the current problem and the RetICL examples is that there are variables defined as offsets and multiplications of other variables. While the kNN examples are more semantically similar to the current problem, the required reasoning steps are very different. The generated RetICL solution more closely follows the style of the examples than the gold solution, breaking down the solution into more steps and using more verbal reasoning.

<table>
<tr><th colspan="2">Problem</th></tr>
<tr><td colspan="2">Table: [TITLE]: Pairs of shoes per store
Stem | Leaf
1 | 9
2 | 3, 9
3 | 2, 8
4 | 5
5 | 2
6 | 2, 3
Problem: Kristen counted the number of pairs of shoes for sale at each of the shoe stores in the mall. How many stores have at least 30 pairs of shoes but fewer than 40 pairs of shoes? (Unit: stores)</td></tr>
<tr><th colspan="2">Gold Solution</th></tr>
<tr><td colspan="2">Count all the leaves in the row with stem 3. You counted 2 leaves, which are blue in the stem-and-leaf plot above. 2 stores have at least 30 pairs of shoes but fewer than 40 pairs of shoes. Final Answer: 2</td></tr>
<tr><th>RetICL</th><th>kNN</th></tr>
<tr><th colspan="2">Selected Examples</th></tr>
<tr><td>Table: barrette | $0.88
bottle of hand lotion | $0.96
sewing kit | $0.94
box of bandages | $0.94
box of breath mints | $0.80
Problem: How much money does Eve need to buy 6 bottles of hand lotion and a barrette? (Unit: $) Solution: Find the cost of 6 bottles of hand lotion. $0.96 × 6 = $5.76 Now find the total cost. $5.76 + $0.88 = $6.64 Eve needs $6.64. Final Answer: 6.64

Table: [TITLE]: Rotten tomatoes per barrel
Stem | Leaf
2 | 0, 2, 6, 7
3 | 5, 6, 9
4 | 1, 5
5 | 4
6 | 1, 2, 9
7 | 2, 3, 5
Problem: The Clarksville Soup Company recorded the number of rotten tomatoes in each barrel it received. How many barrels had at least 70 rotten tomatoes but less than 80 rotten tomatoes? (Unit: barrels) Solution: Count all the leaves in the row with stem 7. You counted 3 leaves, which are blue in the stem-and-leaf plot above. 3 barrels had at least 70 rotten tomatoes but less than 80 rotten tomatoes. Final Answer: 3</td><td>Table: [TITLE]: Pairs of shoes per store
Stem | Leaf
3 | 5
4 | 2, 6
5 | 7
6 | 0, 4
7 |
8 | 4
9 | 0
Problem: Ron counted the number of pairs of shoes for sale at each of the shoe stores in the mall. How many stores have at least 50 pairs of shoes but fewer than 100 pairs of shoes? (Unit: stores) Solution: Count all the leaves in the rows with stems 5, 6, 7, 8, and 9. You counted 5 leaves, which are blue in the stem-and-leaf plot above. 5 stores have at least 50 pairs of shoes but fewer than 100 pairs of shoes. Final Answer: 5

Table: [TITLE]: Pairs of shoes per store
Stem | Leaf
1 | 9
2 | 4
3 | 0, 0, 1
4 | 3
5 | 9
Problem: Emma counted the number of pairs of shoes for sale at each of the shoe stores in the mall. How many stores have at least 24 pairs of shoes but fewer than 46 pairs of shoes? (Unit: stores) Solution: Find the row with stem 2. Count all the leaves greater than or equal to 4. Count all the leaves in the row with stem 3. In the row with stem 4, count all the leaves less than 6. You counted 5 leaves, which are blue in the stem-and-leaf plots above. 5 stores have at least 24 pairs of shoes but fewer than 46 pairs of shoes. Final Answer: 5</td></tr>
<tr><th colspan="2">Generated Solutions</th></tr>
<tr><td>Count all the leaves in the row with stem 3. You counted 2 leaves, which are blue in the stem-and-leaf plot above. 2 stores have at least 30 pairs of shoes but fewer than 40 pairs of shoes. Final Answer: 2 ✓</td><td>Count all the leaves in the rows with stems 3 and 4. You counted 5 leaves, which are blue in the stem-and-leaf plots above. 5 stores have at least 30 pairs of shoes but fewer than 40 pairs of shoes. Final Answer: 5 ✗</td></tr>
</table>

Table 6: Problem from TabMWP with examples and generated solutions from RetICL and kNN, where RetICL examples lead to a correct solution and kNN examples lead to an incorrect solution. While the first RetICL example is seemingly unrelated to the current problem, the second example has the exact same reasoning steps as the current problem, enabling the LLM to copy the solution closely and get the correct answer. On the other hand, while both kNN examples match the current problem semantically, the reasoning steps are slightly different than what are required for the current problem, causing the LLM to get the incorrect answer by following the example reasoning steps too closely.