

A Proofs and Discussions of Theoretical Results

A.1 The proof and discussion of Proposition 3.1

The proof is as follows:

Proof. We first compute the advantage in one group. For query \mathbf{q} with n correct outputs and $N - n$ wrong outputs ($0 < n < N$), the mean and standard deviation of rewards are n/N and $\sqrt{n(N-n)}/N$ respectively. Accordingly, the advantages of positive and negative samples are

$$\begin{cases} A(\mathbf{q}, \mathbf{o}_+) &= \frac{1-n/N}{\sqrt{n(N-n)}/N} = \sqrt{\frac{N-n}{n}} \\ A(\mathbf{q}, \mathbf{o}_-) &= \frac{0-n/N}{\sqrt{n(N-n)}/N} = -\sqrt{\frac{n}{N-n}} \end{cases}, \quad (6)$$

Note that the advantage will be all 0 if $n = 0$ or $n = N$.

Then we consider the GRPO objective in Eq. (3), when the RL training is on policy, the importance ratio $\pi_\theta(\mathbf{o}|\mathbf{q})/\pi_{\text{old}}(\mathbf{o}|\mathbf{q}) = 1$ and thus we can remove the corresponding clip terms. Meanwhile, we ignore the KL divergence regularization when the coefficient β is small. Consequently, the gradient of simplified GRPO objective on query \mathbf{q} is

$$\nabla_\theta \mathcal{J}(\mathbf{q}; \theta) \quad (7)$$

$$= \mathbb{E}_{\{\mathbf{o}_i\} \sim \pi_\theta} \frac{1}{N} \sum_{i=1}^N \left[\frac{\nabla_\theta \pi_\theta(\mathbf{o}_i|\mathbf{q})}{\pi_{\text{old}}(\mathbf{o}_i|\mathbf{q})} A_i \right] \quad (8)$$

$$= \mathbb{E}_{\{\mathbf{o}_i\} \sim \pi_\theta} \frac{1}{N} \sum_{i=1}^N \left[\frac{\pi_\theta(\mathbf{o}_i|\mathbf{q})}{\pi_{\text{old}}(\mathbf{o}_i|\mathbf{q})} \nabla_\theta \log \pi_\theta(\mathbf{o}_i|\mathbf{q}) A_i \right] \quad (9)$$

$$= \mathbb{E}_{\{\mathbf{o}_i\} \sim \pi_\theta} \frac{1}{N} \left[\sum_{i=1}^n \nabla_\theta \log \pi_\theta(\mathbf{o}_{i+}|\mathbf{q}) A_{i+} + \sum_{j=1}^{N-n} \nabla_\theta \log \pi_\theta(\mathbf{o}_{j-}|\mathbf{q}) A_{j-} \right] \quad (10)$$

$$= \mathbb{E}_{\{\mathbf{o}_i\} \sim \pi_\theta} \frac{\sqrt{n(N-n)}}{N} \left[\frac{1}{n} \sum_{i=1}^n \nabla_\theta \log \pi_\theta(\mathbf{o}_{i+}|\mathbf{q}) - \frac{1}{N-n} \sum_{j=1}^{N-n} \nabla_\theta \log \pi_\theta(\mathbf{o}_{j-}|\mathbf{q}) \right] \quad (11)$$

$$= \mathbb{E}_{\{\mathbf{o}_i\} \sim \pi_\theta} \sqrt{\alpha(1-\alpha)} \left[\frac{1}{n} \sum_{i=1}^n \nabla_\theta \log \pi_\theta(\mathbf{o}_{i+}|\mathbf{q}) - \frac{1}{N-n} \sum_{j=1}^{N-n} \nabla_\theta \log \pi_\theta(\mathbf{o}_{j-}|\mathbf{q}) \right] \quad (12)$$

where $\alpha = n/N$ and we use the policy gradient $\nabla_\theta \pi_\theta(\mathbf{q}|\mathbf{o}) = \pi_\theta(\mathbf{q}|\mathbf{o}) \nabla_\theta \log \pi_\theta(\mathbf{q}|\mathbf{o})$ in derivation.

Similarly, if we do not expand the outputs to correct and incorrect ones in one group, the gradient of GRPO objective on the whole query set is

$$\nabla_\theta \mathcal{J}(Q; \theta) = \mathbb{E}_{\mathbf{q}' \sim Q, \{\mathbf{o}'\} \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(\mathbf{o}'|\mathbf{q}') A_i] \quad (13)$$

Therefore, we can obtain the per-step influence by the inner product of $\nabla_\theta \mathcal{J}(\mathbf{q}; \theta)$ and $\nabla_\theta \mathcal{J}(Q; \theta)$.

□

Remark A.1 (Discussions on the assumptions). Our derivation of per-step influence relies on the on-policy training and small KL regularization coefficient assumptions, both of which approximately hold in practical implementation. The on policy training holds when one rollout step corresponds to one optimization step (i.e., one gradient descent step), which is consistent with our implementation (see Appendix B.3), other framework such as open-R1 [96], or DeepSeekMath [23] where each generation is trained only for once. Moreover, the clip term further helps reduce the gap between π_θ and π_{old} even when strictly on-policy training is not guaranteed. For the KL regularization, we observe that most implementations set a very small coefficient β (e.g., 0.01 or smaller), consistent with our assumption. Overall, the per-step influence is mainly dominated by the data co-influence as derived in Proposition 3.1.

983 *Remark A.2* (Relation of data co-influence to neural tangent kernel (NTK)). The data co-influence
 984 \mathcal{K}_θ is also related to NTK [97], a kernel used to describe how neural network evolves during training
 985 via gradient descent. Denote $\chi \doteq (\mathbf{q}, \mathbf{o})$ as the query-output pair, If we regard the language model
 986 as a neural network with χ as input and the logit \mathbf{z} on the whole vocabulary (with size $|V|$) of each
 987 token as output, we have

$$\nabla_\theta \log \pi_\theta(\mathbf{o}|\mathbf{q}) = \frac{1}{T} \sum_{t=1}^T (\pi_\theta^{(t)}(\cdot|\chi) - \mathbf{e}(o_t))^\top \nabla_{\theta \mathbf{z}_t}(\cdot|\chi) \quad (14)$$

988 where T is the length of output \mathbf{o} , o_t denotes the t -th token of the output, $\mathbf{z}_t \in \mathbb{R}^{|V|}$ is the logits of
 989 t -th token. Here $\pi_\theta^{(t)}(\cdot|\chi), \mathbf{e}(o_t) \in \mathbb{R}^{|V|}$. $\pi_\theta^{(t)}(\cdot|\chi)$ is the output distribution of t -th token and $\mathbf{e}(o_t)$
 990 means the one-hot vector. Note that the result is divided by T because there is nothing. Although we
 991 feed the model with the full answer sequence \mathbf{o} in χ , the tokens in later position will not contribute
 992 to computing the output distribution of each token because of the existence of causal mask, which is
 993 natively integrated in auto-regressive language model.

994 Then the data co-influence can be decomposed as

$$\begin{aligned} \mathcal{K}_\theta[\chi, \chi'] &= \langle \nabla_\theta \log \pi_\theta(\mathbf{o}|\mathbf{q}), \nabla_\theta \log \pi_\theta(\mathbf{o}'|\mathbf{q}') \rangle \\ &= \frac{1}{TT'} \sum_{t=1}^T \sum_{\tau=1}^{T'} (\pi_\theta^{(t)}(\cdot|\chi) - \mathbf{e}(o_t))^\top \langle \nabla_{\theta \mathbf{z}_t}(\cdot|\chi), \nabla_{\theta \mathbf{z}_\tau}(\cdot|\chi') \rangle (\pi_\theta^{(\tau)}(\cdot|\chi') - \mathbf{e}(o'_\tau)) \end{aligned} \quad (15)$$

$$(16)$$

995 where the middle term $\langle \nabla_{\theta \mathbf{z}_t}(\cdot|\chi), \nabla_{\theta \mathbf{z}_\tau}(\cdot|\chi') \rangle$ is empirical NTK of the language model [54].

B More Details of Experiments

B.1 Evaluation Benchmark

We provide a query along with a vanilla demonstration answer for iGSM and PromptBench tasks. Meanwhile, we provide the corresponding DAG representation for each query for easy understanding.

iGSM. In iGSM, we construct a DAG by first generating the nodes, which are with the form of "each X's Y". There are two types of nodes – abstract node and instance node – based on the type of "Y". If "Y" is a class name, the node will be classified as an abstract node; if "Y" is an instance of the class, the node will be classified as an instance node. For example, consider a class "Classroom" with its two instances "Painting room", "Computer room", the "each Oakwood Middle School's Classroom" is an abstract node while "each Oakwood Middle School's Computer room" is an instance node. After generating the nodes, we then generate the dependency of instance nodes randomly as edges to construct a DAG (we do not need to add dependency to abstract nodes because it has implicit dependency, e.g., "each Oakwood Middle School's Classroom" = "each Oakwood Middle School's Painting room" + "each Oakwood Middle School's Computer room"). We can then generate query and answer based on the DAG. In query, we only list the explicit dependencies on instance nodes which may include the conditions for redundant nodes; in answer, the LLM should be able to unlock all instance and abstract nodes to the final target node with a topological order. More implementation details (e.g., how to generate random nodes and random edges, how to set the name of nodes) are systematically introduced in the original paper [74].

To improve the training stability, we modify the original iGSM. Here are the main modifications: (1) we remove the modulo operation in all computation steps; (2) we filter out the queries whose ground-truth answers are larger than 1000 or smaller than -1000 ; (3) the original answer template is "Define [node name] as [a random letter], then [the random letter] = [... computation equations]". We rewrite it to increase the answer diversity while keeping the logic of problem-solving (i.e., the order of node selection) and computation details. See below for an example of the new answer template.

One example with operation number $op = 10$ is shown in the following text boxes. The DAG representation of this question is visualized in Figure 7.

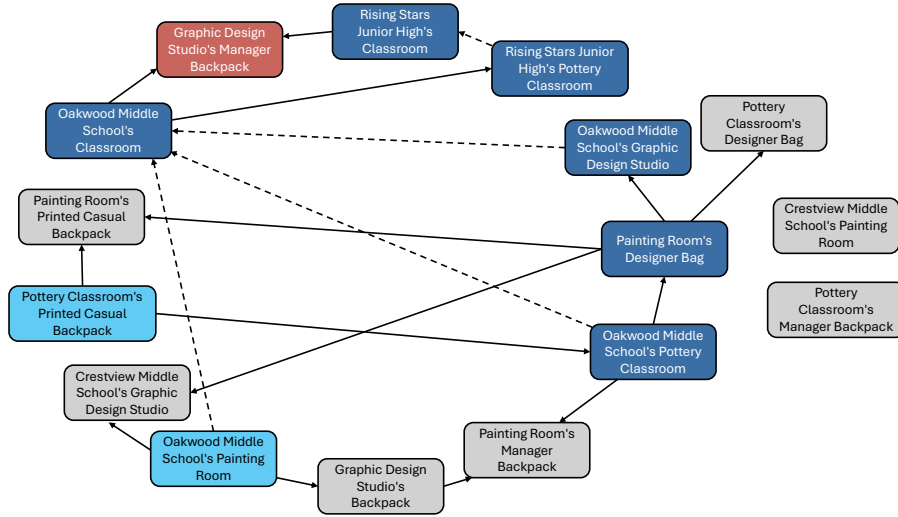


Figure 7: DAG representation of an iGSM task (number of operations = 10). The red, light blue, dark blue, and gray rectangles mean the target, leaf, intermediate, and redundant nodes, respectively. The solid arrow indicates the explicit dependency which is explicitly given in query while the dash arrow means the implicit dependency which needs to be inferred by LLM. For example, it is not directly given how to compute the value of each "Rising Stars Junior High's Classroom" in the query. The LLM is required to understand its semantic meaning and then infer that it equals to "Rising Stars Junior High's Classroom".

Note that the RL training data (15 ~ 20 operations) and OOD test set (25 operations) in practical experiments are much harder than the example above.

Query:

The number of each Graphic Design Studio's Manager Backpack equals the difference of each Rising Stars Junior High's Classroom and each Oakwood Middle School's Classroom.
 The number of each Painting Room's Designer Bag equals 0 more than each Oakwood Middle School's Pottery Classroom.
 The number of each Painting Room's Printed Casual Backpack equals 9 more than the sum of each Painting Room's Designer Bag and each Pottery Classroom's Printed Casual Backpack.
 The number of each Crestview Middle School's Graphic Design Studio equals the difference of each Painting Room's Designer Bag and each Oakwood Middle School's Painting Room.
 The number of each Pottery Classroom's Designer Bag equals each Painting Room's Designer Bag.
 The number of each Pottery Classroom's Manager Backpack equals 0.
 The number of each Pottery Classroom's Printed Casual Backpack equals 1.
 The number of each Rising Stars Junior High's Pottery Classroom equals 2 times each Oakwood Middle School's Classroom.
 The number of each Oakwood Middle School's Painting Room equals 6.
 The number of each Crestview Middle School's Painting Room equals 5.
 The number of each Painting Room's Manager Backpack equals 0 more than the sum of each Graphic Design Studio's Backpack, each Oakwood Middle School's Pottery Classroom and each Oakwood Middle School's Painting Room.
 The number of each Oakwood Middle School's Pottery Classroom equals 5 times each Pottery Classroom's Printed Casual Backpack.
 The number of each Oakwood Middle School's Graphic Design Studio equals 8 more than each Painting Room's Designer Bag.
 How many Manager Backpack does each Graphic Design Studio have?

1025

Vanilla CoT:

Let's compute the answer step by step.

- According to the information given, the number of each Pottery Classroom's Printed Casual Backpack is 1. Let's denote it as S . So $S = 1$.
- Next, let Q represent the number of each Oakwood Middle School's Pottery Classroom. Then $Q = 5 * S = 5 * 1 = 5$.
- Now, we can find the number of each Painting Room's Designer Bag. Let's denote it as U . Then $U = 0 + Q = 0 + 5 = 5$.
- We can then calculate the number of each Oakwood Middle School's Graphic Design Studio. Let it be m . Then $m = 8 + U = 8 + 5 = 13$.
- The number of each Oakwood Middle School's Painting Room is 6. Let's denote it as W . So $W = 6$.
- Then, let's denote the number of each Oakwood Middle School's Classroom as v . Then $v = m + Q + W = 13 + 5 + 6 = 24$.
- We can then calculate the number of each Rising Stars Junior High's Pottery Classroom. Let it be B . Then $B = 2 * v = 2 * 24 = 48$.
- Now, we can find the number of each Rising Stars Junior High's Classroom. Let it be p . Then $p = B = 48$.
- Next, let y represent the number of each Graphic Design Studio's Manager Backpack. Then $y = p - v = 48 - 24 = 24$.

Thus, the answer is 24.

1026

1027 **PromptBench.** PromptBench generates data in two stages: (1) DAG construction and (2) Natural
 1028 language description of the DAG. For DAG construction, we first generate a directed acyclic graph
 1029 (DAG) with a specified depth and number of redundancies. The DAG is constructed top-down: we
 1030 begin by generating the root node and then recursively sampling dependencies between each node and
 1031 its parent node(s). If a node uses a binary operator, it has two parent nodes; if it uses a unary operator,
 1032 it has one parent node. This process continues recursively until the desired depth is reached. During
 1033 DAG construction, each node is assigned a unique name generated by a random string generator.
 1034 Once the DAG structure is complete, we sample values for all leaf nodes from a predefined set and
 1035 compute the values of internal nodes in a bottom-up manner. For the natural language description
 1036 stage, we describe the constructed DAG and its associated computation using predefined templates
 1037 to generate a textual problem description. To improve the training stability, we also filter out the
 1038 queries whose ground-truth answers are larger than 1000 or smaller than -1000 . Full details of the
 1039 generation process can be found in the original PromptBench paper [87].

One example with depth = 4, number of redundancy = 2 is shown in the following text boxes. The DAG representation of this question is visualized in Figure 8

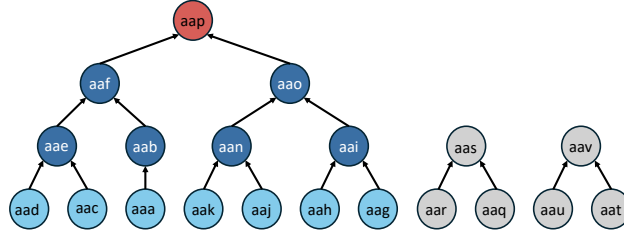


Figure 8: DAG representation of a PromptBench task. The red, light blue, dark blue, and gray circles mean the target, leaf, intermediate, and redundant nodes, respectively.

Query:

The value of aac is 5.
aaf gets its value by adding together the value of aab and aae.
The value of aaj is 5.
The value of aak is 9.
The value of aar is 2.
The value of aaa is 9.
The value of aat is 3.
aao gets its value by adding together the value of aai and aan.
The value of aad is 5.
aas gets its value by multiplying together the value of aaq and aar.
The value of aaq is 5.
aan gets its value by adding together the value of aak and aaj.
The value of aaq is 10.
aai gets its value by multiplying together the value of aah and aag.
aap gets its value by subtracting the value of aao from the value of aaf.
The value of aah is 2.
aav gets its value by multiplying together the value of aat and aaq.
aab gets its value by squaring the value that aaa has.
The value of aag is 6.
aae gets its value by subtracting the value of aac from the value of aad.
What is the value of aap?

Vanilla CoT: Let's compute the answer step by step.

Let's solve aaa, aaa is 9
Let's solve aab, $aab = aaa^2 = 81$
Let's solve aac, aac is 5
Let's solve aad, aad is 5
Let's solve aae, $aae = aac - aad = 0$
Let's solve aaf, $aaf = aab + aae = 81$
Let's solve aah, aah is 2
Let's solve aaj, aaj is 5
Let's solve aag, aag is 6
Let's solve aai, $aai = aag * aah = 12$
Let's solve aak, aak is 9
Let's solve aan, $aan = aaj + aak = 14$
Let's solve aao, $aao = aai + aan = 26$
Let's solve aap, $aap = aao - aaf = 55$
Thus, the answer is 55.

Why are iGSM and Promptbench good testbeds for LLM reasoning evaluation? We can observe that both tasks require the problem-solving capabilities of LLM from multiple perspectives, e.g., basic arithmetic calculation, search and planning, which mirrors reasoning tasks in realistic scenarios. Meanwhile, these tasks require few priors (it only needs very basic concept understanding in iGSM task) and focus on the **abilities** instead of **knowledge**. Moreover, all the data are synthetic and the queries are almost infinite (see estimation in [74]). Therefore, it avoiding the data contamination,

1050 which can significantly confound the experiment results, while keeping substantial diversity of the
1051 query set.

1052 B.2 Examples of Injected Behaviors

1053 In this section, we give an example of injected behaviors on iGSM task.

Injected behaviors in iGSM:

Let's compute the answer step by step.

...

- The number of each Oakwood Middle School's Painting Room is 6. Let's denote it as W . So $W = 6$.

- Then, let's denote the number of each Oakwood Middle School's Classroom as v . Then $v = m + Q + W = 13 + 5 + 6 = 18 + 6 = 24$.

...

- Then, let's denote the number of each Graphic Design Studio's Manager Backpack as y . But we haven't calculated the number of each Rising Stars Junior High's Classroom yet, thus the value of y is still unknown.

...

- Now, we can find the number of each Graphic Design Studio's Manager Backpack. Remember that it has been denoted as y . We know that it equals the difference between each Rising Stars Junior High's Classroom and each Oakwood Middle School's Classroom. Then $y = p - v = 48 - 24 = 24$.

Thus, the answer is 24.

1054

1055 The blue, violet, and cyan parts correspond to the subgoal computation, reflection, and information
1056 analysis behaviors. In subgoal computation, we enforce LLM to conduct only one operation (i.e.,
1057 $+$, $-$ or \times) in every derivation and keep all intermediate computations for complex equations. In
1058 reflection, the LLM explores a locked (not solvable yet) node "each Graphic Design Studio's Manager
1059 Backpack" and then goes back. In information analysis, the LLM integrates the information gathered
1060 from the query for better derivation of computation, which is also an exploitative behavior.

1061 B.3 Training Details

1062 **SFT dataset and baseline implementations.** For the iGSM task, we use 2000 SFT data for Vanilla
1063 and BRIDGE. The PP-Aug and RC-Aug augment each data for additional three times so they have
1064 8000 SFT data in total. For the PromptBench task, we use 5000 SFT data for Vanilla and BRIDGE.
1065 The PP-Aug and RC-Aug augment each data for an additional once so they have 10000 SFT data in
1066 total.

1067 **SFT training.** We first train the LLMs by SFT to narrow the domain gap between the pretraining
1068 corpus and evaluation tasks and enforce the model to answer the question with the demonstrated
1069 template, where the final reward is easy to extract and verify. The other training configurations are
1070 attached below, which are shared by all base models on both the iGSM and PromptBench tasks.

Table 3: Configurations of SFT training

Configurations	value
training epoch	5
batch size	128
learning rate	5×10^{-6}
learning rate scheduler	constant

1071 The system prompt along with query and answer templates are shown as follows:

System prompt:

A conversation that the assistant solves the user's problem. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process

1072

and answer are enclosed within `<think>` `</think>` and `<answer>` `</answer>` tags, respectively, i.e.,
`<think>` all the reasoning process here `</think>` `<answer>` final answer here `</answer>`.

SFT data template (take Qwen tokenizer for example)
`<lim_start>system<lim_end>`
`{system prompt}`
`<lim_start>user<lim_end>`
`{query}`
`<lim_start>assistant<lim_end>`
`<think> {CoT answer} </think>`
`<answer> The final answer is {final answer} </answer>`

RL training. After the SFT stage, we apply RL to further fine-tune the SFT models. Our implementation is based on VeRL [98]. When computing KL divergence, we use the low variance implementation [99], aligning with the GRPO implementation [23]. The shared parameters are listed in Table 4.

Table 4: Configurations of RL training

Configurations	value
batch size	256
learning rate	1×10^{-6}
learning rate scheduler	constant
rollout number per query N	32
rollout temperature	1.0
rollout backend	vllm [100]
KL coefficient β	0.001

In rollout, we set the maximum generation length as 2560 for iGSM and 1536 for PromptBench. We use top $p = 1.0$ for sampling in generation. Besides, we observe that Llama3.2-1B may deviate from answer template so we add a format reward to it, i.e., it will obtain a 0.05 reward if it strictly follows the given answer template (i.e., "`<think>` ... `</think>` `<answer>` ... `</answer>`") addition to the correctness reward.

B.4 More Experiment Results

We attach the training curves of main experiments in Fig. 9&10.

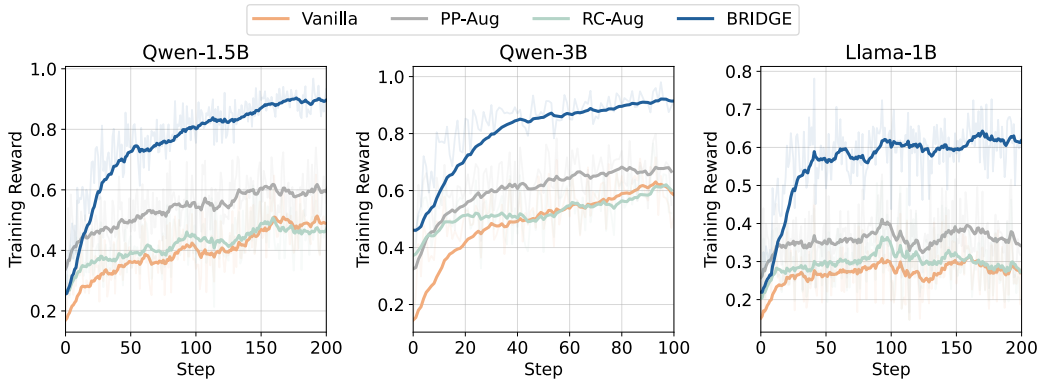


Figure 9: The training curve of experiments in the iGSM task.

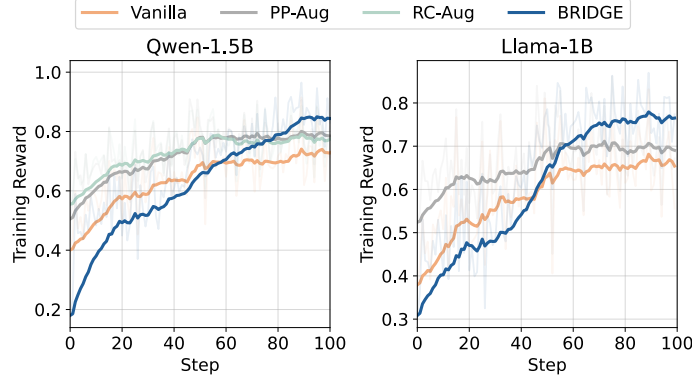


Figure 10: The training curve of experiments in the PromptBench task.

1086 B.5 Comparison between BRIDGE and Accuracy-based Rejection Sampling

1087 Based on the analysis of two factors affecting the performance in RFT in Proposition 3.1, there is an
 1088 intuitive idea that filters out queries with excessively high or low rollout accuracy (close to 1 or 0) as
 1089 discussed in the beginning of sec. 3.3. In this section, we compare the performance of our method
 1090 with the rejection-sampling-based RFT. Specifically, we roll out the SFT models (pre-RL models)
 1091 for 8 times on a larger dataset that shares the same distribution as the original RL training set. We
 1092 then retain only the queries with 1 to 7 correct answers, resulting in a filtered RL training dataset
 1093 with a medium accuracy ratio. Note that the new dataset has the same size as previous RFT training
 1094 dataset. Then we conduct RFT on the rejection sampled dataset starting from the same SFT model.
 1095 The results are shown in Fig. 11.

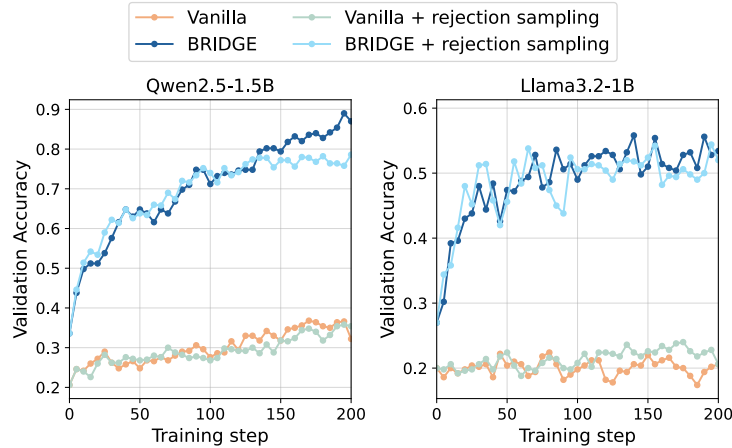


Figure 11: The comparison of BRIDGE with rejection-sampling-based filtering.

1096 We can observe that the rejection sampling does not improve the RFT performance over Vanilla
 1097 baseline and is significantly worse than BRIDGE. Meanwhile, when applying rejection sampling
 1098 to the BRIDGE, the difference is very minor, and it even degrades the final performance on the
 1099 Qwen-1.5B model. The reason for its poor performance is that rejection sampling can only shape
 1100 the accuracy distribution of RL rollouts at the initial stage. As training progresses, the distribution
 1101 may shift and is not guaranteed to remain within the medium-accuracy range. More importantly, this
 1102 method does not improve the data co-influence. Based on the above results, we can conclude that it
 1103 is more effective to improve the data co-influence when preparing LLM for RFT, which relies on
 1104 behavior injection, rather than simply adjusting rollout accuracy by distorting the query distribution.

1105 B.6 Details of Data Co-influence and Per-step Influence Estimation

1106 We mainly follow LESS [82] in data co-influence estimation. Specifically, there are two main
1107 steps reducing the dimensionality of $\nabla_{\theta} \log \pi_{\theta}(\mathbf{o}|\mathbf{q})$: (1) we first compute the LoRA gradient
1108 $\nabla_{\theta_{\text{LoRA}}} \log \pi_{\theta_{\text{LoRA}}}(\mathbf{o}|\mathbf{q})$ rather than the full-parameter gradient by backpropagating the likelihood loss
1109 to the LoRA modules. (2) Then we apply random projection on the LoRA gradient and get a vector
1110 with a smaller dimension, which preserves the inner product of two gradients [101]. For the LoRA
1111 module, we specified a rank of 64, an α value of 128, a dropout rate of 0.1, and learned LoRA
1112 matrices for all attention matrices. For the random projection, the final dimension of the projection
1113 output is 8192. Then we can estimate the data co-influence coefficient $\mathcal{K}_{\theta}[\cdot, \cdot]$ by the inner product
1114 between two query-output samples.

1115 After obtaining the data co-influence coefficient, we can compute the per-step influence $\Delta\mathcal{J}$ by
1116 plugging the data co-influence and advantages. **Note that we do not multiple the learning rate η**
1117 **when computing the results in Fig. 3.**

1118 B.7 Computation Overhead

1119 All experiments can be run on a server with $2 \times \text{A100}$ (80G). Each SFT experiment takes less than 1h.
1120 Regarding the RL experiments, it takes $\sim 8\text{h}$ for Qwen-1.5B and Llama-1B to complete 200-step
1121 RFT and $\sim 6\text{h}$ for Qwen-3B to run 100-step RFT on iGSM while it spends $\sim 2\text{h}$ to run RFT on
1122 PromptBench (100 steps) for Qwen-1.5B and Llama-1B due to shorter rollout length.