VerifierQ: Enhancing LLM Test Time Compute with Q-Learning-based Verifiers

Jianing Qi [1] **Hao Tang** [1 2] **Zhigang Zhu** [1 3]

Abstract

Recent test time compute approaches with verifier models have significantly enhanced the reasoning capabilities of Large Language Models (LLMs). While this kind of generator-verifier approach closely resembles the actor-critic framework in reinforcement learning (RL), current verifiers typically rely on supervised fine-tuning rather than temporal-difference methods. We propose VerifierQ, a novel approach that integrates Offline Q-learning into LLM verifier models to address three well-known challenges of LLMs: utterance-level Markov Decision Processes (MDPs), large action spaces, and overestimation bias. VerifierQ introduces a modified bounded Bellman update to keep Q-values in $[0, 1]$, incorporates Implicit Q-learning (IQL) for efficient approximation of the $\max Q$ over large action space, and introduces an adjustable two-expectile Conservative Q-learning (CQL). Our method is among the first attempts to integrate offline Q-learning to LLM verifiers, and adjustable conservatism can tighten Q-values around the true $\max Q$ more flexibly than IQL alone. Experimental results on mathematical reasoning tasks demonstrate VerifierQ's superior performance compared to supervised fine-tuning approaches.

1. Introduction

Despite Large Language Models (LLMs) ability in generating coherent text, LLMs face significant challenges in sustained, multi-step logical reasoning (Lightman et al., 2024). Overcoming these challenges is critical for enabling the next level of agent capabilities. A promising direction is the concept of test-time compute, which adds a verifier to evaluate and select the best answers from a set of generated solutions (Snell et al., 2024; Cobbe et al., 2021). It typically employs two main components: a **generator** and a **verifier** (Lightman et al., 2024; Uesato et al., 2022; Cobbe et al., 2021). The generator produces potential solutions, while the verifier evaluates their correctness. This setup is analogous to the actor-critic framework in Reinforcement Learning (RL) (Konda & Tsitsiklis, 1999). However, unlike RL critics that use temporal-difference (TD) updates for long-term credit assignments, current verifiers in multi-step reasoning are often trained using supervised fine-tuning (SFT) akin to Behavioral Cloning. This limitation might hinder the verifier's ability to guide the generator toward better long-term outcomes, particularly in complex reasoning tasks without enough data(Kumar et al., 2022).

In contrast, the RL critic in actor-critic methods applies temporal-difference learning to capture long-term credit assignment. Motivated by this gap between SFT and RL in verifier, we propose VerifierQ: the first approach to apply offline reinforcement learning (via IQL and CQL) specifically to the verifier model. A key challenge is that verifier actions exist at the utterance level, creating an extremely large action space. Prior works on offline RL in LLM focus on token-level generation or omit CQL at utterance level (Snell et al., 2023; Zhou et al., 2024), but we combine IQL and CQL in a fully utterance level setting for the verifier. Through careful design, VerifierQ addresses both this large action space and the overestimation problems that naturally arise in offline Q-learning.

Previous works focused on improving generators using methods like Monte Carlo Tree Search (MCTS) (Chen et al., 2024; Wang et al., 2024b;a; Zhang et al., 2024a). However, less attention has been given to applying RL to verifiers, and most works still use SFT approaches for verifier. In VerifierQ, we adapt classic offline Q-learning techniques—particularly Implicit Q-learning (IQL) and Conservative Q-learning (CQL)—to enable long-horizon reasoning, prevent overestimation, and handle large-scale utterance-level actions within LLMs. Our contributions are:

- **Parallel Utterances-Level MDP:** We formulate a flexible MDP for the verifier to operate on the full solution

[1]Department of Computer Science, CUNY Graduate Center, New York, NY, USA [2]Department of Computer Science, Borough of Manhattan Community College, New York, NY, USA [3]Department of Computer Science, The City College of New York, New York, NY, USA. Correspondence to: Jianing Qi <jqi@gradcenter.cuny.edu>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.

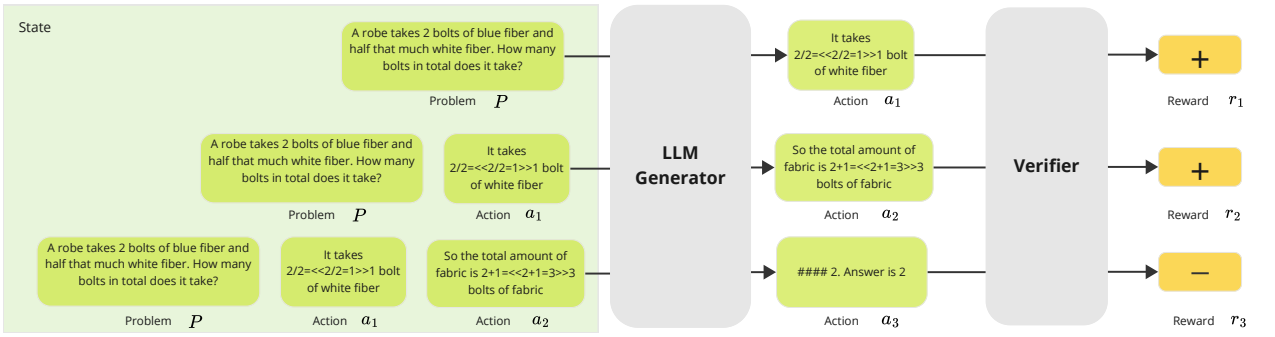

Figure 1: Illustration of State, Action (green), and Reward (orange) in a Math Problem. + denotes correct (1) and − denotes incorrect (0). A state generator produces an action (i.e., the next solution step). For example, a_2 is generated from $[P, a_1]$, and a_3 is generated from $[P, a_1, a_2]$. The verifier assesses the current state and action and outputs a correctness probability.

steps rather than individual step. This design enables us to compute Q-values for multiple steps in a single forward pass.

- **Two-Expectile Offline Q-Learning:** We introduce a novel combination of IQL and CQL, enabling adjustable optimism that can go beyond the approximate $\max Q$ typical of standard IQL.
- **Empirical Validation:** We evaluate VerifierQ on GSM8K and MATH with a 1B-parameter model, demonstrating consistent gains over strong verifier baselines and showcasing the practical viability of integrating RL principles into LLM verifiers.

2. Background

In reinforcement learning (RL), tasks are often modeled as Markov Decision Processes (MDPs). A MDP consists of states $s \in S$, actions $a \in A$, a reward function $r(s, a)$, a transition function $P(s'|s, a)$ from state s to state s' with action a, and a discount factor γ. The goal is to find an optimal policy π^* that maximizes the expected cumulative reward: $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$

Q-learning estimates the optimal state action value Q by iteratively updating expected cumulative rewards, while the V function is similar to Q but just estimates from states without need of actions. Q-learning is a model-free RL algorithm commonly used to solve MDPs by minimizing the temporal difference (TD) error:

$$L_{TD}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right)^2\right]. \quad (1)$$

We can formulate the reasoning process as an utterance-level MDP. Given a problem statement p, the generator produces a solution based on the problem and previous action steps, where $[p, a_1, ..., a_{i-1}]$ is the state and a_i is an action. Each action is a complete sentence as shown in Figure 1. This differs from token-level approaches where actions would be individual tokens from the vocabulary, which can be a word or a subword. The state at step i is the dialogue history that consists of the problem statement and all previous complete utterances generated up to that point: $s_i = [p, a_1, a_2, \ldots, a_i]$. Rewards are given at each step, with a reward of 1 for a correct step and 0 for an incorrect one. We can see the illustrated example in Figure 1, where + indicates "correct" and − indicates "incorrect".

In classical RL, the critic model is trained to estimate the Q value (Konda & Tsitsiklis, 1999). In LLM, the verifier model is trained to estimate the Q value (Snell et al., 2024). Given a problem statement, the generator produces a sequence of steps as actions. The verifier's inputs are the problem and solution steps, and it outputs correctness scores. We find equivalence between actor-critic and generator-verifier framework.

3. Related Work

RL for Verifiers: Multi-step reasoning in LLMs often follows a generator-verifier framework, in which the generator proposes intermediate steps and the verifier checks their correctness (Lightman et al., 2024; Uesato et al., 2022; Cobbe et al., 2021). Process Reward Modeling (PRM) assigns rewards at each step rather than only at the final output—outperforming Object Reward Modeling (ORM) on math tasks (Lightman et al., 2024), but it requires significant manual labels. It is similar to imitation learning in RL. Recent efforts reduce this labeling cost using Monte Carlo Tree Search (MCTS) rollouts (Chen et al., 2024; Wang et al., 2024b;a), though the resulting step-level data can be noisy (many incorrect steps).

In principle, offline RL methods such as Q-learning could stitch together these suboptimal partial solutions more ef-

fectively than supervised (imitation learning) in a fixed dataset(Kumar et al., 2022). However, current verifiers typically rely on supervised fine-tuning (SFT) rather than offline RL, and this imitation approach might not be optimal compared to the offline RL. Our work is the first to implement offline RL with temporal difference updates in a verifier model, bridging the gap between standard RL critics and LLM verifiers.

Utterance Level Actions: A key challenge in language-based RL is deciding whether to treat each token as an action or to treat entire utterances (sentences) as single actions. Existing work in offline RL uses token-level approaches (e.g., Snell et al., 2023). It has a comparatively smaller action space (equal to the vocabulary size) but can be cumbersome for longer reasoning tasks, and utterance-level methods can better perform on longer reasoning steps but introduce a far larger action space (Zhou et al., 2024; Wang et al., 2024a). Prior works using utterance level offline RL generally estimate one utterance-level Q-value per forward pass (Snell et al., 2023; Zhou et al., 2024). Our approach builds on these ideas by computing multiple utterance-level Q-values in a single forward pass, substantially improving computational efficiency.

Large Action Space: Because each utterance may contain multiple tokens, the space of possible actions explodes exponentially. In principle, accurately estimating the maximum Q-value requires sampling many candidate utterances for each partial solution (Chen et al., 2024; Wang et al., 2024a). Some prior approaches use an actor (e.g., MCTS) to sample from this space while holding earlier steps fixed. By contrast, our method sidesteps the need for explicit sampling of next-step utterances for each partial solution instead relies on binary labels (correct vs. incorrect) for each step from Wang et al. (2024b). Our method handles large action space through regression aspect of IQL implicitly.

Overestimation in Offline RL: Overestimation is a known pitfall of Q-learning, particularly in offline settings where the dataset may be limited or unrepresentative (Kumar et al., 2020). This challenge is magnified in language tasks, where arbitrary text sequences can lie outside the training distribution (Verma et al., 2022; Zhou et al., 2024). Conservative Q-learning (CQL) and Implicit Q-learning (IQL) help mitigate overestimation by penalizing Q-values for out-of-distribution actions. While token-level LLM work has used IQL (Snell et al., 2023), utterance-level CQL remains largely unexplored: Snell et al. and Zhou et al. both omit the CQL term when extending to utterance-level. In contrast, our work combines CQL and IQL for full utterance-level verifiers, thereby addressing overestimation without reverting to token-level granularity.

4. Verifier with Q-Learning (VerifierQ)

We now present VerifierQ, an approach that combines Offline Q-learning methods—specifically IQL and CQL—to train a verifier in Large Language Models (LLMs). Figure 2 provides an overview of our architecture. The overarching goal is to enable the verifier to efficiently handle utterance-level MDPs and large action spaces, while mitigating overestimation in offline settings.

Modified Bellman Update for Utterance-Level MDPs: To adapt Q-learning for utterance-level reasoning, we first represent correctness as two special tokens: $+$ (correct) and $-$ (incorrect). Each step a_i in the solution is followed by a "tag" token that triggers a Q-value estimate. By predicting the probability of + (vs. -), we interpret the output as a Q-value in [0,1], consistent with per-step rewards in $[0, 1]$.

To address the bounded nature of outputs (0 to 1) compared to traditional Q-values, we modify the Q-learning algorithm to operate within these constraints. We propose using the $\frac{1}{2}$ of the traditional Bellman update target as the target value instead of Equation 1:

$$Q^*(s, a) = \frac{1}{2}(r(s, a) + \gamma \max_{a'} Q^*(s', a')) \tag{2}$$

where $Q^*(s, a)$ is the optimal Q-value for state s and action a, $r(s, a)$ is the immediate reward, γ is the discount factor, and $\max_{a'} Q^*(s', a')$ is the maximum Q-value for the next state s' taking action a'. More details are provided in the supplementary material (Theorem A.1 of Appendix A.1).

Here we need to ensure that Q^* is bounded. Since $r, Q \in [0, 1]$, we have $0 \leq r + \gamma \max Q \leq 2$ for $\gamma \in [0, 1]$. Thus we can bound $0 \leq Q^*(s, a) = \frac{1}{2}(r(s, a) + \gamma \max_{a'} Q^*(s', a')) \leq 1$. Therefore $Q^*(s, a) \in [0, 1]$, and it aligns naturally with the model's probabilistic outputs.

In practice, we insert the tag token at the end of each step $[p, a_1, tag, a_2, tag, \ldots, a_n, tag]$ and estimate multiple Q-values in a single forward pass (Fig 2, left). This parallel computation significantly improves efficiency for multi-step verification.

Handling Large Action Spaces via IQL: A classical challenge in utterance-level MDPs is the enormous action space (V^n, where V is the vocabulary size and n the token length). Implicit Q-learning (IQL) (Kostrikov et al., 2022) addresses this by regressing onto actions seen in the offline dataset, thus avoiding explicit enumeration or sampling of all possible utterances. We follow Snell et al. (2023) for the IQL framework to our setting, using the expectile of the Q-value to approximate the value function V. Formally, we use expectile regression:

$$\mathcal{L}V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau\left(Q_\theta(s, a) - V_\psi(s)\right)\right] \tag{3}$$

where $L_2^\tau(u) = |\tau - \mathbb{1}(\mathrm{u} < 0)|u^2, \tau \in (0, 1)$ is the quantile

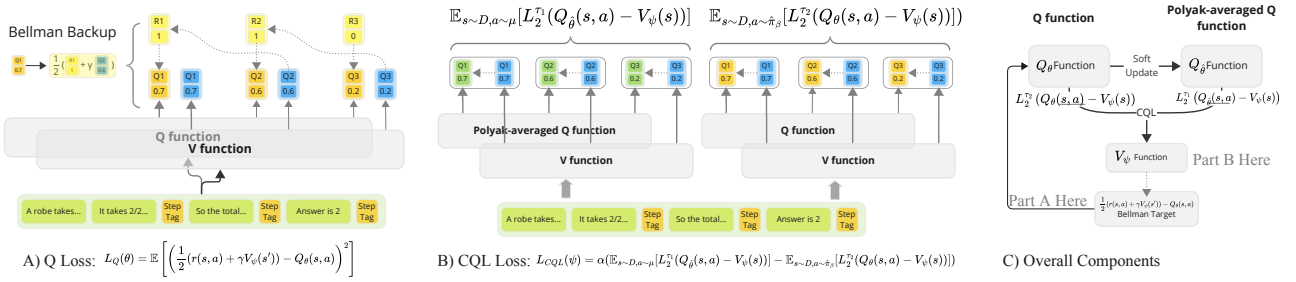

Figure 2: Illustration of the VerifierQ architecture and modified Bellman update. Left: Bellman update, where Q_θ is updated via the TD target with V. Middle: CQL Loss component. The main goal is to have the lower bound V_ψ as the target policy distribution while the upper bound as the data distribution. Right: Relationships among Q_θ, $Q_{\hat{\theta}}$, and V_ψ. Q_θ updates through the Bellman equation as shown in (A), V_ψ is updated through CQL as shown in (B), and $Q_{\hat{\theta}}$ is updated via soft update. The key intuition here is to leverage IQL's regression to overcome the large action space problem while CQL keeps estimation in check.

level, $\mathcal{D}$ is the offline dataset, Q_θ is the learned Q-function, and V_ψ is the approximated value function.

Intuitively, IQL approximates $\max_a Q(s,a)$ without explicitly computing it over $\mathcal{O}(V^n)$ actions. This is especially valuable for LLM verifiers, where an utterance can easily contain dozens of tokens, resulting in a combinatorial explosion. By focusing on regression against the offline data, IQL remains efficient even for large language action spaces. As τ approaches 1, we have $\lim_{\tau\to 1} V_\psi(s) = \max_a Q^*_\theta(s,a)$ Kostrikov et al. (2022), ensuring that our IQL-based approach can asymptotically recover the optimal value function, even with large action spaces, given sufficient coverage in the offline dataset.

Our approach leverages regression aspect to solve the large action space problem. By expectile regressing the reward over utterances, we can avoid explicitly sampling the action or iterate through all the combinations of tokens. The approximation of minimum through τ value is shown in Theorem A.2 of Appendix A.1 .

Mitigating Overestimation via a Two-Expectile CQL Formulation.

Although IQL handles large action spaces by avoiding explicit maximization, offline Q-learning remains prone to overestimation, particularly when the dataset is sparse or underrepresented. In essence, IQL alone can approximate the maximum Q-value well within the data distribution but lacks a mechanism to explicitly penalize out-of-distribution actions.

To remedy this, we introduce a Conservative Q-learning (CQL) penalty (Kumar et al., 2020) on top of IQL, yielding a novel two-expectile objective. Specifically, we enforce a conservative bound on actions not well-supported by the dataset:

$$\arg\min_Q \alpha(\mathbb{E}_{s\sim D, a\sim\mu}\left[Q(s,a)\right] - \mathbb{E}_{s\sim D, a\sim\hat{\pi}_\beta}\left[Q(s,a)\right]) \tag{4}$$

Here μ is the target policy distribution and $\hat{\pi}_\beta$ is the data distribution. This objective effectively pushes down overestimated Q-values. Unlike token-level approaches that skip CQL for large utterances (Snell et al., 2023; Zhou et al., 2024), we combine CQL and IQL for utterance-level verification, thus controlling overestimation while still approximating $\max Q$ implicitly.

Moreover, to allow adjustable optimism, we add separate expectile levels τ_1, τ_2 in the CQL term:

$$\begin{aligned} L_{CQL}(\psi) = \alpha(&\mathbb{E}_{s\sim D, a\sim\mu}\left[L_2^{\tau_1}\left(Q_{\hat{\theta}}(s,a) - V_\psi(s)\right)\right] \\ &- \mathbb{E}_{s\sim D, a\sim\hat{\pi}_\beta}\left[L_2^{\tau_2}\left(Q_\theta(s,a) - V_\psi(s)\right)\right]) \end{aligned} \tag{5}$$

Here, $\tau_1 \approx 0$ and $\tau_2 \approx 1$ let us independently modulate conservatism for out-of-distribution actions while maintaining optimism on in-distribution actions. Intuitively, the "lower expectile" side gently pushes down $Q_{\hat{\theta}}$ for high-risk (i.e., underexplored) actions, whereas the "upper expectile" side ensures that plausible actions from the dataset remain sufficiently valued. For more details on the explanations, see Appendix A.5.

This design goes beyond standard single-expectile CQL or IQL alone. By merging their strengths, our two-expectile approach can tune the degree of optimism vs. conservatism, even with massive utterance-level action spaces. It makes our formulation the first that unifies both IQL-based max-Q approximation and adjustable conservative penalties for an LLM verifier.

Overall Objective: Bringing it all together, VerifierQ minimizes a sum of TD Loss and a CQL Term.

1. TD Loss Like previous works Snell et al. (2023), we

use a separate value function $V_\psi(s')$ to approximate the Q-value $\max_{a'} Q^*(s', a')$. With our adaptation to the Bellman Update (Equation 1), the TD error is:

$$L_Q(\theta) = \mathbb{E}\left[\left(\frac{1}{2}(r(s,a) + \gamma V_\psi(s')) - Q_\theta(s,a)\right)^2\right] \quad (6)$$

2. CQL Term We add a CQL term to achieve an adjustable conservatism with Equation 5.

The comprehensive objective function of VerifierQ is thus the sum of the Bellman error (Equation 6) and the CQL term (Equation 5):

$$L(\theta, \psi) = L_Q(\theta) + L_{CQL}(\psi) \quad (7)$$

To enhance training stability, we employ a Polyak-averaged version of $Q_{\hat{\theta}}$ (Polyak & Juditsky, 1992). The hyperparameter α is set to 1 in our experiments, balancing the influence of the CQL term. The overall objective is shown in the Figure 2.

This formulation allows VerifierQ to benefit from the conservative nature of CQL while maintaining an optimistic outlook, crucial for effective Q-value estimation in large action spaces characteristic of language models. The expectile regression provides flexibility to adjust τ_1, τ_2 values as preferred. By integrating these components, VerifierQ addresses the challenges of overestimation and large action spaces in utterance-level MDPs, providing a robust framework for multi-step reasoning tasks.

5. Experiments and Results

We evaluate VerifierQ on two standard mathematical reasoning benchmarks, GSM8K and MATH (Cobbe et al., 2021; Hendrycks et al., 2021). These datasets are among the most commonly used for multi-step math reasoning in recent work (Lightman et al., 2024; Wang et al., 2024b;a; Snell et al., 2024; Yu et al., 2024; Chen et al., 2024).

5.1. Experimental Setup

We compare VerifierQ with a well-established verifier baseline method Process Reward Model (PRM) method Lightman et al. (2024), and we use the same dataset as Wang et al. (2024b); Snell et al. (2024) for training. We do not include Object Reward Model (ORM) since Wang et al. (2024b); Snell et al. (2024); Lightman et al. (2024) already validated PRM's effectiveness over ORM. Due to computational constraints, we use the TinyLlama-1.1B model (Zhang et al., 2024b).

Dataset: We generate a test time compute set using a generator trained on MetaMath (Yu et al., 2024). The generator is finetuned on MetaMath for 2 epochs with a learning rate of 2e-5. We then use LoRA finetuning for 1 epoch to adjust the format of answer style so it can be easier to extract the answers (Hu et al., 2022). For each question in the full GSM8K test set and a 500-question subset of MATH (following Lightman et al. (2024)), we generate 256 answers. The verifier is trained on the MathShepherd dataset Wang et al. (2024b), which uses MCTS-generated data with binary rewards (1 for correct, 0 for incorrect).

Model Architecture: Our verifier model consists of a Q-network and a separate value network to prevent single sample overestimation. We employ soft updates to stabilize training with rate 0.01.

Training: We initialize our verifier model with MetaMath pretraining model, then train with PRM on MathShepherd for 1 epoch, followed by VerifierQ training. Here are key hyperparameters. Learning rate: 2e-5 for all training phases. Batch size: 64 (crucial for Q-learning stability). Q-learning parameters: $\gamma = 0.99$, $\alpha = 1$ for the CQL term. For PRM, we continued training on MathShepherd from 1 epoch to 2 epochs. Majority Voting uses the raw output from the generator.

Evaluation Metrics: We evaluate the verifier against PRM and Majority Voting using accuracy. Following Snell et al. (2024); Lightman et al. (2024), we use minimum evaluation metrics.

5.2. Results

We evaluate VerifierQ against PRM (for one epoch and two epochs) and Majority Voting on both GSM8K and MATH datasets using minimum evaluation. For VerifierQ, we use $\tau_1 = 0.3$ for GSM8K and $\tau_1 = 0.5$ for MATH. Figure3 compares VerifierQ, PRM, PRM (2nd epoch), and Majority Voting on GSM8K and MATH as the number of solutions per problem grows. VerifierQ consistently outperforms PRM.

We notice PRM 2nd epoch perform worse than PRM 1st epoch. We hypothesized that the model might overfit the dataset. Hence for PRM we will use first epoch PRM baseline in subsequent comparisons. On GSM8K, VerifierQ's performance improves with an increase in the number of solutions per problem, aligning with trends observed in previous studies (Snell et al., 2024; Lightman et al., 2024; Wang et al., 2024b). While the absolute improvement margins may appear modest, a Wilcoxon signed-rank test demonstrates that VerifierQ's improvements over PRM are statistically significant (GSM8K:$W = 1500.0, p < 1.6410^{-35}$ MATH:$W = 156.5, p < 4.7010^{-43}$). This extremely low p-value indicates that the performance differences are highly unlikely to occur by chance, suggesting that VerifierQ provides consistent improvements over baseline methods. This aligns with our theoretical expectations of improvements in

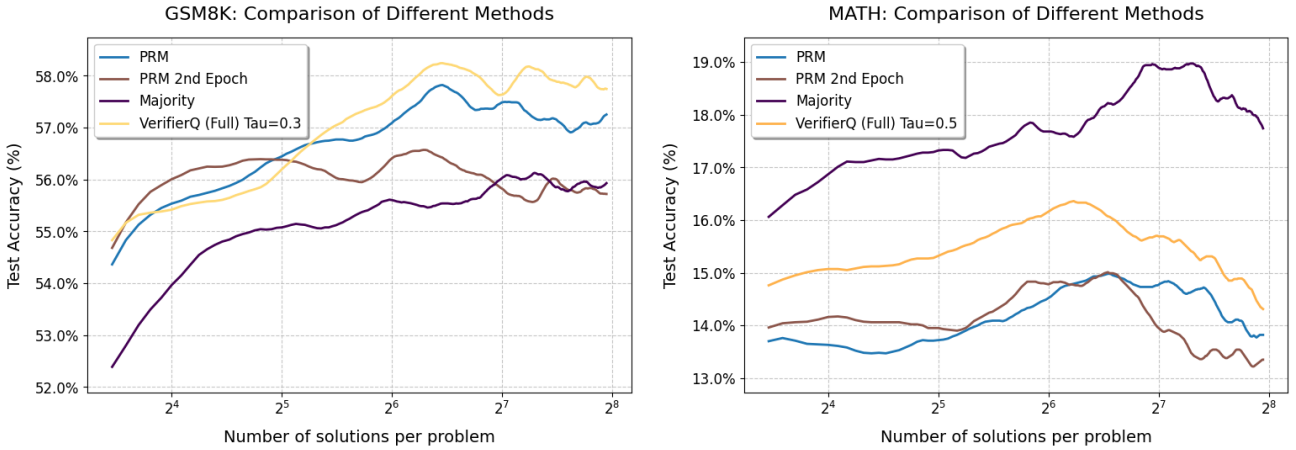

Figure 3: Comparison of different methods on GSM8K (left) and MATH (right) using minimum evaluation. Rolling average over 20 steps. For VerifierQ we use $\tau_1 = 0.3$ (left) and $\tau_1 = 0.5$ (right). VerifierQ (Yellow) outperforms PRM (Blue) on both datasets.

verifier accuracy. We also want to point out that VerifierQ's advantage emerges as it better leverages multiple solutions for value estimation. Previous work showed that PRM's accuracy increases more noticeably as the number of candidate solutions grows compared to majority voteLightman et al. (2024); Wang et al. (2024b); Snell et al. (2024). We observe that VerifierQ's accuracy increases even higher than PRM as the number of solutions grows in GSM8K.

We note that for MATH, all methods underperform compared to Majority Vote. We hypothesize that it is due to the large shift of the language modeling objective to value estimation objective. The small model size (1.1B) is the cause of discrepancy. Previous works use 7B size model as the minimum baseline, sometimes 70B. Due to practical reasons, we do not have the resources to train the model larger than 1B. Larger models are more sample efficient than smaller models Kaplan et al. (2020), and in our tasks, it means larger LLM shall learn better and have better performance given the same dataset size. However, VerifierQ still outperform all other methods among learned verifiers. We observe VerifierQ consistently outperform to PRM method. Furthermore, VerifierQ method also has a better performance compared to other Q learning methods, and we will discuss it in ablation study in Section 6.3.

In Figure 3, VerifierQ achieves the higher accuracy both on GSM8K and on MATH with different τ_1 values compared to PRM (see Figure 4 in Section 6), and it also outperforms other variations (see Figure 5 in Section 6).

Overall, these results confirm that offline RL can provide a systematic advantage over purely supervised verifiers, even at smaller model scales. While the absolute improvements may appear modest, the statistical significance underscores VerifierQ's robustness. We further analyze the impact of various components and hyperparameters in the ablation studies (Sections6–6.3).

6. Ablation Study

Our ablation study addresses the challenges of large action spaces, computational efficiency, and the impact of key components in VerifierQ. We investigate the computational efficiency of our parallel utterance-level design versus other architectures, the impact of CQL hyperparameters (especially τ_1) on performance and overestimation, and the impact of IQL CQL components on the performance.

6.1. Multi-Utterances-level Architecture for Computational Efficiency

A key advantage of VerifierQ's parallel sentence-level architecture is that it estimates all Q-values for an entire sequence in a single forward pass. Previous "utterance-level" methods often follow a BERT-style approach, which use [CLS] token to estimate the Q value of one utterance Zhou et al. (2024). This approach require separate forward passes for each step's Q-value estimation, resulting in $O(n^3m^2)$ complexity for n steps with m tokens each. n is the number of reasoning steps, and m is the average tokens per step, and in Mathshepherd dataset $n \approx 6, m \approx 20$. In contrast, VerifierQ's tag-token insertion strategy requires only a single forward pass, reducing complexity to $O(n^2m^2)$.

Concretely, if there are n reasoning steps and each step has m tokens, we avoid repeated re-encoding of shared tokens for each Q-value. Instead, our approach gathers all needed Q-estimates in one pass, resulting in substantial speedups—particularly for longer sequences. Appendix C.1 provides formal proofs and additional comparisons (including sequential decoder baselines).

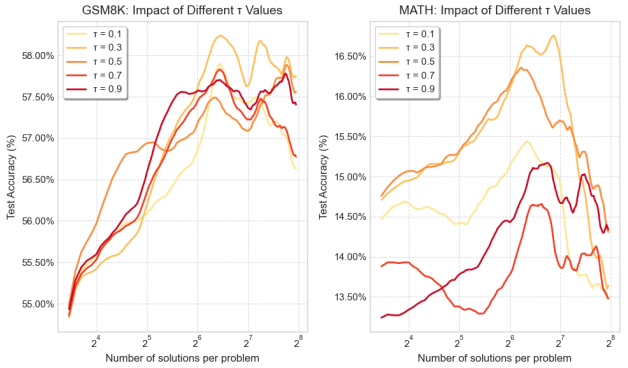

Figure 4: Impact of different τ_1 values on VerifierQ performance. Left: GSM8K dataset. Right: MATH dataset. The colors transition gradually from yellow to red, with the τ_1 value ranging from 0.1 to 0.9. Higher τ_1 values correspond to redder colors.

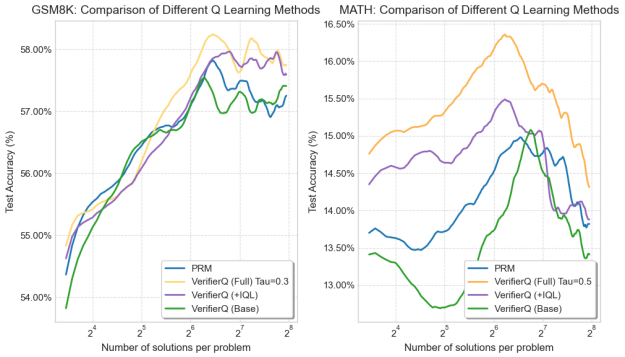

Figure 5: Comparison of different components. PRM is blue. VerifierQ (Base) is green. VerifierQ (+IQL) is purple. VerifierQ (Full) is yellow. Left: GSM8K dataset. Right: MATH dataset.

6.2. Impact of Adjustable CQL Term on Overestimation

Another focal point is how Conservative Q-Learning (CQL) hyperparameters govern the trade-off between optimism (higher Q-values) and conservatism (avoiding overestimation). We focus primarily on τ_1, which controls the lower expectile bound, while fixing τ_2 at 0.9, a choice guided by theoretical considerations of the CQL objective. In essence, τ_2 ensures that out-of-distribution Q-values are pushed down toward the data distribution, whereas τ_1 influences how aggressive or lenient that push is.

Figure 4 illustrates the impact of different τ_1 values on VerifierQ's performance. We examine different levels of optimism by varying τ_1 (0.1, 0.3, 0.5, 0.7, 0.9). As shown in Figure 4, $\tau_1 = 0.3$ generally yields better results, suggesting it approximates the maximum Q-value more effectively than other τ_1 values. MATH dataset shows higher sensitivity to τ_1 values, with $\tau_1 = 0.5$ performing the best. It suggest that a more conservative Q-value estimate is beneficial for the more complex MATH tasks, and it makes intuitive sense since it needs to be more careful for more complex tasks. The difference in optimal τ_1 values between datasets suggests that dataset-specific tuning may be necessary. It also suggests that more complex tasks (MATH) benefit from more conservative estimation.

In summary, while CQL helps mitigate overestimation in offline RL, its precise benefits depend on choosing the right τ_1, τ_2. For simpler tasks, a lower τ_1 may suffice, whereas more difficult reasoning tasks benefit from a more conservative setting.

6.3. Component-wise Analysis on Performance

To understand how each element of VerifierQ contributes to final performance, we perform an ablation by systematically removing key components from the full approach. We compare the variants below to PRM, and summarize results in Table1 and Figure5. VerifierQ Variants:

- **VerifierQ (Base)**: A straightforward Q-learning model without any IQL or CQL modifications. This is essentially SARSA-style, using our bounded Bellman update (Eq.2) but no additional mechanism for handling large action spaces or overestimation.
- **VerifierQ (+IQL)**: Adds Implicit Q-learning (IQL) on top of the Base variant to approximate $\max Q$. It is an Implicit Q-learning (IQL) with $\tau = 0.9$ similar to (Snell et al., 2023). This helps manage large utterance-level action spaces but does not address overestimation with CQL. The objective can be referenced with Equation 6 with an additional $L_2^\tau\left(Q_{\hat{\theta}}(s,a) - V_\psi(s)\right)$ term.
- **VerifierQ (Full)**: Combines both IQL and CQL components. This tackles the exponential action space (via IQL) and overestimation bias (via CQL). The complete loss from Equation7.

Base Performance: VerifierQ (Base) performs on par with PRM for GSM8K but fares worse for MATH. This suggests naive Q-learning, absent specialized solutions for large ac-

Table 1: Final Accuracy (%) of different methods over 256 answers. Higher is better. VerifierQ(Full) with $\tau_1 = 0.3$ (GSM8K) and $\tau_1 = 0.5$ (MATH)

METHOD	GSM8K	MATH
PRM	57.32	13.8
VERIFIERQ(BASE)	57.39	13.2
VERIFIERQ(+IQL)	57.39	14.0
VERIFIERQ(FULL)	57.70	14.4

tion spaces or overestimation, provides only minimal benefits—confirming our hypothesis that a direct application of classic Q-learning is inadequate.

Impact of IQL: VerifierQ (+IQL) significantly improves over the Base variant on GSM8K, highlighting the importance of addressing large action spaces. However, on MATH, its accuracy remains comparable to PRM, indicating that overestimation can still degrade performance in more challenging tasks.

Full VerifierQ: VerifierQ (Full) shows notable and consistent improvements on both GSM8K and MATH datasets. VerifierQ outperforms other methods after 2^5 in GSM8K and all the time in MATH. By integrating CQL, the model more effectively controls overestimation, while IQL addresses the large action space. The adjustable τ-parameters (τ_1, τ_2) allow dataset-specific tuning of conservatism, helping the verifier remain sufficiently optimistic where supported by data.

These findings highlight how each additional component in VerifierQ resolves a specific challenge (i.e., large action spaces or overestimation), culminating in consistent improvements in multi-step math reasoning. For further qualitative insight, we provide a case study in AppendixC.2, illustrating how VerifierQ more accurately identifies correct solution paths than simpler baselines.

7. Discussion, Ethics, and Limitations

VerifierQ introduces a novel way to integrate offline reinforcement learning into verifier models for large language models (LLMs). By drawing parallels with the actor-critic paradigm, it opens avenues for more sophisticated multi-step reasoning and planning strategies—echoing successes from broader AI systems like AlphaGo, which combine learning and search in complex decision spaces.

Ethical Considerations: As VerifierQ (and similar approaches) become more advanced, ethical challenges arise around transparency. As decision-making grows in complexity, ensuring human interpretability of the model's internal Q-value judgments (e.g., how it balances partial correctness, or why it penalizes certain solution paths) becomes essential. Current experiment relies on automatic labeled data. As many cases in RL, the model sometimes might make decisions that we cannot comprehend but are effective. As the model's decision-making process becomes more complex, ensuring transparency and maintaining a human understanding what values represent becomes increasingly challenging but vastly important.

Limitations:

- Model Scale: All experiments use the TinyLlama (1.1B) model due to computational constraints. Larger LLMs (e.g., 7B or 70B parameters) might yield stronger results but require substantially more resources.
- Hyperparameter Sensitivity: We observe that τ_1 tuning in CQL heavily influences performance, and a single setting does not necessarily generalize across tasks (e.g., GSM8K vs. MATH). More extensive hyperparameter sweeps could further stabilize or improve results, but were limited by computational budgets.
- Resource Usage: Compared to straightforward supervised fine-tuning (SFT), VerifierQ maintains three networks $(Q_\theta, V_\psi, Q_{\hat{\theta}})$ for offline RL, increasing VRAM usage by roughly 2.25×. We need to train two models and perform soft updates on one for VerifierQ, while PRM just need to train one model. This overhead could be a barrier to broader adoption, emphasizing the need for memory-efficient architectures.

Overall, VerifierQ broadens the design space for LLM verifiers by harnessing the potential of offline RL. Continued research and engineering advances are needed to refine memory usage, tune hyperparameters effectively, and ensure ethical, transparent deployment in real-world reasoning tasks.

8. Conclusion

We introduced VerifierQ, an offline Q-learning framework that equips Large Language Model (LLM) verifiers with temporal-difference (TD) capabilities. By recasting multi-step reasoning as an utterance-level MDP, VerifierQ avoids many of the standard pitfalls in naive Q-learning (intractable action spaces and overestimation) through a two-expectile design combining IQL and CQL. Concretely:

- **Parallel, Utterance-Level Architecture:** A single forward pass estimates Q-values across all reasoning steps, enabling efficient handling of massive discrete spaces without exhaustive sampling.
- **Adjustable Conservatism:** Our novel integration of IQL (to approximate $\max Q$) with CQL (to penalize out-of-distribution actions) lets us balance optimism and caution.
- **Empirical Validation:** Experimental results on GSM8K and MATH reveal that VerifierQ outperforms purely supervised verifiers (e.g., PRM), especially as the number of generated solutions increases.

By infusing TD learning into the verifier's update rule, VerifierQ offers a bridge between classic RL critics and LLM-based verification. In doing so, it opens the door to fully

actor–critic solutions for natural language tasks, where generators propose candidate solutions and verifiers adaptively refine them using offline data. As the broader field of RL for language continues to mature, we believe VerifierQ provides a flexible foundation for robust multi-step reasoning on the verifier side.

Acknowledgments

The work is supported by the National Science Foundation (NSF) through Awards #2131186 (CISE-MSI), #1827505 (PFI), and the Google CyberNYC Initiative. The work is also supported by a College-wide Research Vision (CRV) Fund from the CCNY Provost's Office, and the ODNI Intelligence Community Center for Academic Excellence (IC CAE) at Rutgers University (#HHM402-19-1-0003 and #HHM402-18-1-0007).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. Reinforcement learning critics can be challenging to interpret, potentially obscuring the reasons for verification decisions. Ensuring transparency and explainability remains crucial.

References

Chen, G., Liao, M., Li, C., and Fan, K. Alphamath almost zero: Process supervision without process. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VaXnxQ3UKo.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Neural Information Processing Systems*, 1999. URL https://api.semanticscholar.org/CorpusID:207779694.

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1179–1191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

Kumar, A., Hong, J., Singh, A., and Levine, S. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=AP1MKT37rJ.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. doi: 10.1137/0330046. URL https://doi.org/10.1137/0330046.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Snell, C. V., Kostrikov, I., Su, Y., Yang, S., and Levine, S. Offline RL for natural language generation with implicit language q learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=aBH_DydEvoH.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback, 2022.

Verma, S., Fu, J., Yang, M., and Levine, S. Chai: A chatbot ai for task-oriented dialogue with offline reinforcement learning, 2022. URL https://arxiv.org/abs/2204.08426.

Wang, C., Deng, Y., Lyu, Z., Zeng, L., He, J., Yan, S., and An, B. Q*: Improving multi-step reasoning for llms with deliberative planning, 2024a. URL https://arxiv.org/abs/2406.14283.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510.

Yu, L., Jiang, W., Shi, H., YU, J., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Zhang, D., Huang, X., Zhou, D., Li, Y., and Ouyang, W. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b, 2024a. URL https://arxiv.org/abs/2406.07394.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model, 2024b. URL https://arxiv.org/abs/2401.02385.

Zhou, Y., Zanette, A., Pan, J., Levine, S., and Kumar, A. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=b6rA0kAHT1.

A. Reproducibility Statement

To ensure reproducibility of our results, we provide the following details:

Implementation Details: The complete implementation details are available in Appendix B.1.

Hardware Requirements:

- VerifierQ experiments: Conducted on a single NVIDIA A100 GPU with 40GB memory.
- Other models (Q Learning and PRM): Can be trained on an NVIDIA RTX 4090 GPU.

Training Time:

- VerifierQ: Approximately 10 hours for 1 epoch.
- PRM: Approximately 4 hours for 1 epoch.

Datasets: We use the following publicly available datasets:

- MetaMath: https://huggingface.co/datasets/meta-math/MetaMathQA
- GSM8K: https://huggingface.co/datasets/openai/gsm8k
- MathShepherd: https://huggingface.co/datasets/peiyi9979/Math-Shepherd
- MATH (test subset): We use the same dataset as Lightman et al. (2024), available at https://github.com/openai/prm800k

Model: All experiments were conducted using the TinyLlama-1.1B model.

Hyperparameters: Key hyperparameters include:

- Learning rate: 2e-5 (constant for all training phases)
- Batch size: 64
- Discount factor (γ): 0.99
- CQL coefficient (α): 1
- Soft update coefficient (α_{soft}): 0.01
- IQL coefficients:
  - For GSM8K: $\tau_1 = 0.3$, $\tau_2 = 0.9$
  - For MATH: $\tau_1 = 0.5$, $\tau_2 = 0.9$

Training Process: The model is initialized with MetaMath pretraining, followed by 1 epoch of PRM training before VerifierQ training begins.

For any additional details or clarifications needed to reproduce our results, please refer to the code and documentation that will be made available upon acceptance.

A.1. Architecture Details

To apply Offline Q-learning to LLMs at the utterance level, we propose a flexible architecture that integrates with language modeling tasks. Following Wang et al. (2024b) and Lightman et al. (2024), we utilize two tokens $+$ and $-$ to represent correct and incorrect states, with a tag token indicating estimation. The probability of the correct token can be interpreted as Q-values ranging from 0 to 1, aligning with the reward structure in the MCTS-generated dataset from Wang et al. (2024b). We compute the Q-value for each step as:

$$Q(s,a) = p(+) = \text{softmax}(\text{logit}_+) = \text{sigmoid}(\text{logit}_+ - \text{logit}_-) \tag{8}$$

It is flexible to choose either softmax or sigmoid function to compute the Q-value. We use the sigmoid function in our experiments for more effficiency. The Q-value is computed for each step in the solution sequence, estimating a numerical value in the range of (0, 1).

This formulation offers several advantages:

1. It allows flexible integration for Q-value estimation of utterances of arbitrary length since we can insert the step tag anywhere in the sequence.
2. It enables parallel estimation of multiple Q-values for multiple steps in a single forward pass, significantly reducing computation time.
3. This approach seamlessly integrates with existing language modeling tasks.

A.2. Convergence of Modified Bellman Update

We first prove that our modified Bellman update converges to a fixed point.

Theorem A.1 (Convergence of Modified Bellman Update)**.** *Let Q^* be the optimal Q-function. The modified Bellman update*

$$Q^*(s,a) = \frac{1}{2}(r(s,a) + \gamma \max_{a'} Q^*(s',a')) \tag{9}$$

converges to a unique fixed point.

Proof. Let $\mathcal{T}$ be the operator defined by our modified Bellman equation:

$$\mathcal{T}Q(s,a) = \frac{1}{2}(r(s,a) + \gamma \max_{a'} Q(s',a')) \tag{10}$$

We need to show that $\mathcal{T}$ is a contraction mapping in the sup-norm $\|\cdot\|_\infty$. For any two Q-functions Q_1 and Q_2:

$$\begin{aligned}
\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty &= \sup_{s,a} |\mathcal{T}Q_1(s,a) - \mathcal{T}Q_2(s,a)| && (11)\\
&= \sup_{s,a} \left| \frac{1}{2}\gamma \left(\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right) \right| && (12)\\
&\le \frac{1}{2}\gamma \sup_{s,a} \left| \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right| && (13)\\
&\le \frac{1}{2}\gamma \sup_{s',a'} |Q_1(s',a') - Q_2(s',a')| && (14)\\
&= \frac{1}{2}\gamma \|Q_1 - Q_2\|_\infty && (15)
\end{aligned}$$

Since $0 < \gamma < 1$, it follows that $0 < \frac{1}{2}\gamma < 1$. Therefore, $\mathcal{T}$ is a contraction mapping with contraction factor $L = \frac{1}{2}\gamma$. By the Banach fixed-point theorem, $\mathcal{T}$ has a unique fixed point, and the Q-learning algorithm will converge to this fixed point. □

A.3. Optimality of IQL in Large Action Spaces

Next, we prove that IQL can effectively approximate the maximum and minimum Q-value in large action spaces.

Theorem A.2 (IQL Optimality)**.** *We can directly get the following result from the proof in (Kostrikov et al., 2022). As the quantile level τ approaches 1, the IQL value function V_ψ converges to the maximum Q-value:*

$$\lim_{\tau \to 1} V_\psi(s) = \max_{a \in \mathcal{A}, \pi_\beta(a|s) > 0} Q^*(s, a) \tag{16}$$

Additionally, as $\tau \to 0$, the IQL value function V_ψ converges to the minimum Q-value:

$$\lim_{\tau \to 0} V_\tau(s) = \min_{a \in \mathcal{A}} Q^*(s, a) \tag{17}$$

Proof Sketch. Following Lemma 1 of Kostrikov et al. (2022) we can show a modified Lemma:

Lemma A.3. *Let X be a real-valued random variable with bounded support and infimum x^*. Since X is bounded below and m_τ approaches the lower bound as $\tau \to 0$, we have:*

$$\lim_{\tau \to 0} m_\tau = \inf\{x : F_X(x) > 0\} = x^* \tag{18}$$

For all τ_1 and τ_2 such that $0 < \tau_1 < \tau_2 < 1$, we can get $m_{\tau_1} \leq m_{\tau_2}$. Therefore, as $\tau \to 0$, the limit of m_τ converges to the infimum of the random variable X.

In addition, using Lemma 2 of Kostrikov et al. (2022), we can show that the IQL value function V_ψ converges to the minimum Q-value as $\tau \to 0$:

Lemma A.4. *For all s, τ_1 and τ_2 such that $0 < \tau_1 < \tau_2 < 1$, we have $V_{\tau_1}(s) \leq V_{\tau_2}(s)$.*

Since $Q^*(s, a)$ is bounded below, the minimum Q-value exists and is finite. Therefore, as $\tau \to 0$, the IQL value function V_ψ converges to the minimum Q-value:

$$\lim_{\tau \to 0} V_\tau(s) = \inf_{a \in \text{supp}(\pi_\beta)} Q^*(s, a) \tag{19}$$

So we have:

$$\lim_{\tau \to 0} V_\tau(s) = \min_{a \in \mathcal{A}, \pi_\beta(a|s) > 0} Q^*(s, a) \tag{20}$$

□

A.4. Adjustable Conservative Q-values with Modified CQL

Finally, we present a proposition about our modified CQL approach and its potential to lead to adjustable conservatism.

Proposition A.5 (Modified CQL Bounds)**.** *The modified CQL objective with expectile levels τ_1 (close to 0) and τ_2 (close to 1) aims to provide both lower and upper bounds on the true Q-function $Q^*(s, a)$:*

$$\max_{a \sim \hat{\pi}_\beta} Q_\theta(s, a) \lesssim Q^*(s, a) \lesssim \min_{a \sim \mu} Q_{\hat{\theta}}(s, a) \tag{21}$$

where $\lesssim$ denotes "approximately less than or equal to".

Remark A.6 (Supporting Arguments and Intuitions). The original CQL objective is:

$$\arg\min_Q \alpha(\mathbb{E}_{s \sim D, a \sim \mu}[Q(s, a)] - \mathbb{E}_{s \sim D, a \sim \hat{\pi}_\beta}[Q(s, a)]) \tag{22}$$

Where μ is the target policy distribution and $\hat{\pi}_\beta$ is the data distribution. Intuitively, this term finds the maximum Q-value under the target policy distribution $\mathbb{E}_{s \sim D, a \sim \mu}[Q(s, a)]$ and minimizes it since it is usually overestimated. To get a tighter bound, it pushes the Q-value up under the data distribution $\mathbb{E}_{s \sim D, a \sim \hat{\pi}_\beta}[Q(s, a)]$.

For large action spaces, CQL typically uses importance sampling to estimate $\mathbb{E}_{s\sim D,a\sim\mu}[Q(s,a)]$ with $\log\sum a\exp(Q(s,a))$ at every state (Kumar et al., 2020). However, unlike token-level approaches, we leverage IQL to approximate Q-values in the large action space. This mitigates the requirement to sample a set number of actions for each state and allows for more efficient Q-value estimation for longer sequences.

We propose a novel formulation that directly approximates both the lower bound Q-function and the upper bound of the data distribution using IQL with different τ values for each term in CQL objective. The goal remains the same: finding the overestimated Q-value under the target policy to minimize it and tighten the bound with the data distribution. However we want to give control on the level of the tightening of the bound.

Our modified CQL objective is:

$$\begin{aligned} L_{CQL}(\psi) = \alpha(&\mathbb{E}_{s\sim D,a\sim\mu}[L_2^{\tau_1}(Q_{\hat{\theta}}(s,a) - V_\psi(s))] \\ &- \mathbb{E}_{s\sim D,a\sim\hat{\pi}_\beta}[L_2^{\tau_2}(Q_\theta(s,a) - V_\psi(s))]) \end{aligned} \tag{23}$$

The first term, with τ_1 close to 0, approximates the lower bound of $Q_{\hat{\theta}}$. It acts as an upper bound on the target policy which is typically overestimated. This suggests:

$$V_\psi(s) \lesssim \min_{a\sim\mu} Q_{\hat{\theta}}(s,a) \tag{24}$$

The second term, with τ_2 close to 1, approximates an upper bound on Q_θ. It acts as a lower bound on the data distribution, indicating:

$$V_\psi(s) \gtrsim \max_{a\sim\hat{\pi}_\beta} Q_\theta(s,a) \tag{25}$$

This approach allows for a more optimistic Q-value estimation within the CQL framework. The lower bound of the target policy μ pushes the Q-value down less aggressively, while the upper bound of the data distribution $\hat{\pi}_\beta$ elevates the Q-value more, resulting in a more optimistic Q-value under the CQL term. This approach maintains the benefits of CQL's conservatism while allowing for adaptable optimism through the adjustment of τ_1 and τ_2.

The modified CQL objective aims to minimize the difference between the lower bound of the overestimated Q-values ($Q_{\hat{\theta}}$) and the upper bound of the true Q-values (Q_θ). Minimizing this difference may lead to a more accurate estimation of $Q^*(s,a)$. We can express this as:

$$L_{CQL}(\psi) \approx \min_{a\sim\mu} Q_{\hat{\theta}}(s,a) - \max_{a\sim\hat{\pi}_\beta} Q_\theta(s,a) \tag{26}$$

As this difference approaches zero, it suggests that the the lower bound of the overestimation of Q-values is being reduced to the extent supported by the data, and we should have $\max_{a\sim\hat{\pi}_\beta} Q_\theta(s,a) \lesssim \min_{a\sim\mu} Q_{\hat{\theta}}(s,a)$. Adjusting τ_1 we could have $Q_{\hat{\theta}}(s,a)$ approximately close to the optimal maximum.

This formulation allows us to balance conservatism with optimism in Q-value estimation. The lower bound of the Q-value is pushed down less aggressively, while the upper bound is elevated more, resulting in Q-values that approach the maximum Q-value under the data distribution more closely. We can adjust τ_1 and τ_2 to fine-tune this balance, allowing for more adaptable Q-values under the CQL term.

It is important to note that this difference can potentially become negative. A negative value would imply that the estimated lower bound of $Q_{\hat{\theta}}$ is smaller than the estimated upper bound of Q_θ for some state-action pairs. While this might seem counterintuitive given the general overestimation tendency of $Q_{\hat{\theta}}$, it can occur due to the approximations introduced by the L_2^τ loss functions or other factors in the learning process. This suggests that the value function might be correctly valuing the in-distribution actions more highly, which is desirable, although it might introduce some pessimism in the value estimates.

This intuition provides insight into why our modified CQL approach might lead to a bit more optimistic Q-values. However, a rigorous mathematical proof would require further development and analysis.

Figure 6 illustrates the intuition. In the original CQL term, an overestimated Q-value would be pushed down to the data distribution. In our formulation, the lower bound of the Q-value is pushed down less aggressively, and the upper bound is elevated more, resulting in a more optimistic Q-value that approaches the maximum Q-value under the data distribution more closely. The advantage of this approach is that τ value can be adjusted to balance conservatism with optimism.

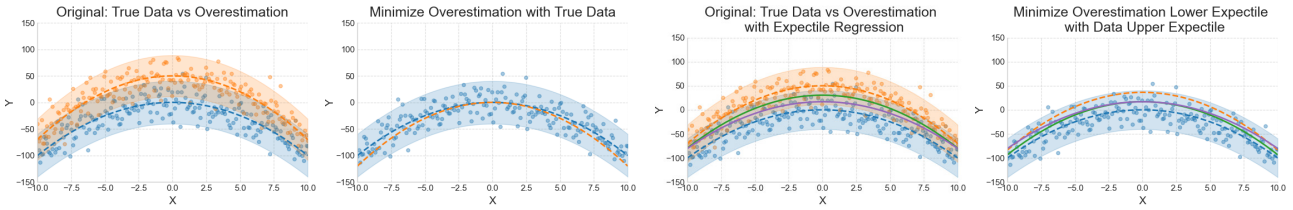

Figure 6: Illustration of our approach. Left two graphs: Orange line represents the overestimated Q-value $Q_{\hat{\theta}}$. The blue line indicates the data distribution Q_θ. Minimizing the overestimation term brings the orange line down to the mean of data distribution. Right two graphs: The green line shows the lower expectile of the overestimated Q-value and the purple line shows the upper expectile of the data Q-value. Minimizing those two can make the orange line approach the maximum Q-value under the data distribution.

B. Appendix

B.1. Algorithm and Implementation Details

Algorithm 1 VerifierQ

Input: Dataset D, Q-network Q_θ, target Polyak-averaged Q-network $Q_{\hat{\theta}}$ with α_{soft}, value network V_ψ, IQL coefficients τ_1 and τ_2, CQL coefficient α
Initialize Q-network Q_θ, target Q-network $Q_{\hat{\theta}}$, value network V_ψ
Initialize target Q-network parameters $\hat{\theta} \leftarrow \theta$
for each training step **do**
  Sample batch of state-action pairs $S = (s_1, s_2, s_3) \sim D$ and rewards $R = (r_1, r_2, r_3) \sim D$
  # TD Target.
  Compute target Q-values in parallel: $y = \frac{1}{2}(r + \gamma V_\psi(S'))$ {Equation 2}
  TD Loss: $L_Q(\theta) = \frac{1}{2}(Q_\theta(S) - y)^2$ {Equation 6}
  # CQL Term.
  Compute CQL μ with IQL: $L_\mu = L_2^{\tau_1}(Q_{\hat{\theta}}(S) - V_\psi(S))$ {Equation 3}
  Compute CQL $\hat{\pi}_\beta$ with IQL: $L_{\hat{\pi}} = L_2^{\tau_2}(Q_\theta(S) - V_\psi(S))$ {Equation 3}
  CQL Loss: $L_{CQL}(\psi) = \alpha(L_\mu - L_{\hat{\pi}})$ {Equation 5}
  # Update networks
  Update Q-network: $\theta \leftarrow \theta - \nabla_\theta L_Q(\theta)$
  Update value network: $\psi \leftarrow \psi - \nabla_\psi L_{CQL}(\psi)$
  Update target Q-network: $\hat{\theta} \leftarrow (1 - \alpha_{\text{soft}})\hat{\theta} + \alpha_{\text{soft}}\theta$
end for

As described in Section 2, the state at step i is the concatenation of the problem statement and all tokens generated up to that point: $s_i = [p, a_1, a_2, \ldots, a_i]$. As illustrated in Figure 1, s_1 consists of p and a_1, s_2 consists of p, a_1, and a_2, and so on. The reward r_i is 1 if the token a_i is correct and 0 otherwise. This approach leverages the decoder architecture's ability to generate the next token based on the previous tokens.

For the hyperparameters, we use the following settings:

- Discount factor: $\gamma = 0.99$
- CQL coefficient: $\alpha = 1$
- Soft update coefficient: $\alpha_{\text{soft}} = 0.01$
- Batch size: 64
- Optimizer: AdamW with a constant learning rate of $2e-5$ for all training phases
- IQL coefficients:

- For GSM8K: $\tau_1 = 0.3$, $\tau_2 = 0.9$
- For MATH: $\tau_1 = 0.5$, $\tau_2 = 0.9$

We initialize the model with MetaMath pretraining and train it with PRM for 1 epoch before starting VerifierQ training. All experiments are conducted using the TinyLlama-1.1B model.

C. Appendix

C.1. Computational Efficiency Analysis

VerifierQ's architecture offers significant computational advantages over both BERT-style encoder and traditional sequential decoder approaches.

For a solution sequence with n steps, where each step contains m tokens on average, let $C(m)$ denote the computational cost of a forward pass through m tokens. The self attention in transformer in general has Big-O of $O(L^2d + Ld^2)$ where as L is the sequence Length and d is depth. Since we have depth to be the same and varies of sequence length, we can view it in our case as $O(L^2)$. The traditional sequential approach computes:

$$Q_{\text{seq}}(s_i, a_i) = f_\theta(\text{concat}(s_i, a_i)) \tag{27}$$

where f_θ represents the model's forward pass.

For example, consider a three-step solution:

Problem: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?

Step 1: It takes $2/2 =<< 2/2 = 1 >> 1$ *bolt of white fiber* $< tag > +$

Step 2: So the total amount of fabric is $2 + 1 =<< 2 + 1 = 3 >> 3$ *bolts of fabric* $< tag > +$

Step 3: The Answer is: $3 < tag > +$

BERT-style Encoder Approach: For encoder architectures like BERT, Q-values are typically estimated through a [CLS] token. As shown in Zhou et al. (2024), they are using RoBERTa based model to estimate the utterance level with "[CLS]" token:

$$Q_{\text{encoder}}(s_i, a_i) = f_\theta([\text{CLS}] + \text{concat}(s_i, a_i)) \tag{28}$$

where the [CLS] token representation is used for value estimation.

As an example would estimate like this:

- Pass 1: $Q(s_1, a_1) = f_\theta([\text{CLS}] + [p, a_1])$
- Pass 2: $Q(s_2, a_2) = f_\theta([\text{CLS}] + [p, a_1, a_2])$
- Pass 3: $Q(s_3, a_3) = f_\theta([\text{CLS}] + [p, a_1, a_2, a_3])$

This requires separate encoding for each step since we can have only one [CLS] each time. Since we have n steps, each step contains m tokens on average, and $C(m) = O(L^2)$, we can have following:

$$\text{Cost}_{\text{encoder}} = \sum_{i=1}^{n} C(im) = \sum_{i=1}^{n} (im)^2 = m^2 \sum_{i=1}^{n} i^2 = m^2 \frac{n(n+1)(2n+1)}{6} = O(m^2 n^3) \tag{29}$$

Sequential Decoder Approach: Traditional decoder architectures estimate Q-values at sequence endpoints by predicting the last embedding:

$$Q_{\text{seq}}(s_i, a_i) = f_\theta(\text{concat}(s_i, a_i)) \tag{30}$$

An example would be like this:

Sequential Decoder:

- Pass 1: $Q(s_1, a_1) = f_\theta([p, a_1])$
- Pass 2: $Q(s_2, a_2) = f_\theta([p, a_1, a_2])$
- Pass 3: $Q(s_3, a_3) = f_\theta([p, a_1, a_2, a_3])$

Where as the Q value estimation is from the linear head of the last token embeddings.

Similarly requiring n separate computations:

$$\text{Cost}_{\text{seq}} = \sum_{i=1}^{n} C(im) = \sum_{i=1}^{n} (im)^2 = O(m^2 n^3) \tag{31}$$

VerifierQ's Parallel Approach: Following Wang et al. (2024b), VerifierQ uses decoder architecture with strategically placed tag tokens enabling parallel estimation.

$$[Q_{\text{parallel}}(s_1, a_1), ..., Q_{\text{parallel}}(s_n, a_n)] = f_\theta(\text{concat}(s_1, \text{tag}, a_1, ..., s_n, \text{tag}, a_n)) \tag{32}$$

As an example it would be like following: **VerifierQ:**

$$[Q(s_1, a_1), Q(s_2, a_2), Q(s_3, a_3)] = f_\theta([p, a_1, \text{tag}, a_2, \text{tag}, a_3, \text{tag}]) \tag{33}$$

It would require only a single forward pass:

$$\text{Cost}_{\text{parallel}} = C(nm) = O(n^2 m^2) \tag{34}$$

This parallelization provides several advantages:

- Reduced complexity: From $O(n^3 m^2)$ to $O(n^2 m^2)$
- Parallel computation: Simultaneous Q-value estimation for all steps
- Decoder architecture benefits: Natural alignment with autoregressive generation

To quantify these benefits, consider our preliminary experiments using the MathShepherd dataset (Wang et al., 2024b), where solutions average 6.2 steps per problem. The sequential approach requires:

- Two forward passes per step (current Q and next Q) for Q-learning
- Total passes per problem = 6.2 steps $\times$ 2 passes $= 12.4$

In contrast, VerifierQ computes all Q-values in a single forward pass. This theoretical reduction from approximately 12 passes to 1 aligns with our preliminary observations of approximately 10× reduction in training time. This efficiency gain would become even more pronounced for longer solution sequences where n is large, demonstrating the scalability of our approach.

C.2. Qualitative Overestimation Analysis

We conduct a qualitative study on overestimation between PRM and VerifierQ by replacing important tokens in the solution sequence with incorrect ones. Figure 7 reveals that PRM generally assigns higher Q-values to incorrect tokens, while VerifierQ assigns lower values.

This analysis compares three configurations: 1. PRM (baseline) 2. VerifierQ without CQL (ablation) 3.VerifierQ (full model). We want to see how well the correct value is differentiated from other incorrect values, so we use $\Delta = Correct - mean(Incorrect)$ to measure the difference between the correct answer and the mean of the incorrect answers. Then we want to see how much this value compared to mean to see whether it is a large differentiation or not, and we use Percentage as $\Delta/mean$

Our analysis reveals several key patterns:

- **PRM**: Assigns relatively high Q-values (mean: 0.24) with average discrimination between correct and incorrect tokens (Δ = 0.03). Percentage is 11.9%.
- **VerifierQ without CQL**: Shows higher overall Q-values (mean: 0.39) but slightly better discrimination (Δ = 0.05). Percentage 12.6%.
- **VerifierQ with CQL**: Demonstrates: 1. Lower overall Q-values (mean: 0.20) 2. Stronger discrimination between correct and incorrect tokens (Δ = 0.06) 3. More aggressive penalization of incorrect tokens (Percentage: 30.6%)

The CQL term's impact is twofold:

1. **General Conservative Estimation**: Reduces overall Q-values to mitigate general overestimation. We can see the without CQL term the Q learning has overestimated significantly, and CQL term brought down overestimation. However adding CQL the correct value is about the same as PRM but incorrect ones are lower. It penalizes the incorrect tokens more.
2. **Enhanced Discrimination**: Increases the gap between correct and incorrect token estimates. The Percentage gap of correct tokens has almost doubled compared to the PRM and VerifierQ without CQL.

This analysis suggests that while CQL does lead to generally lower Q-values, its primary benefit is the enhanced ability to distinguish between correct and incorrect solutions. The increased gap between correct and incorrect Q-values (Percentage

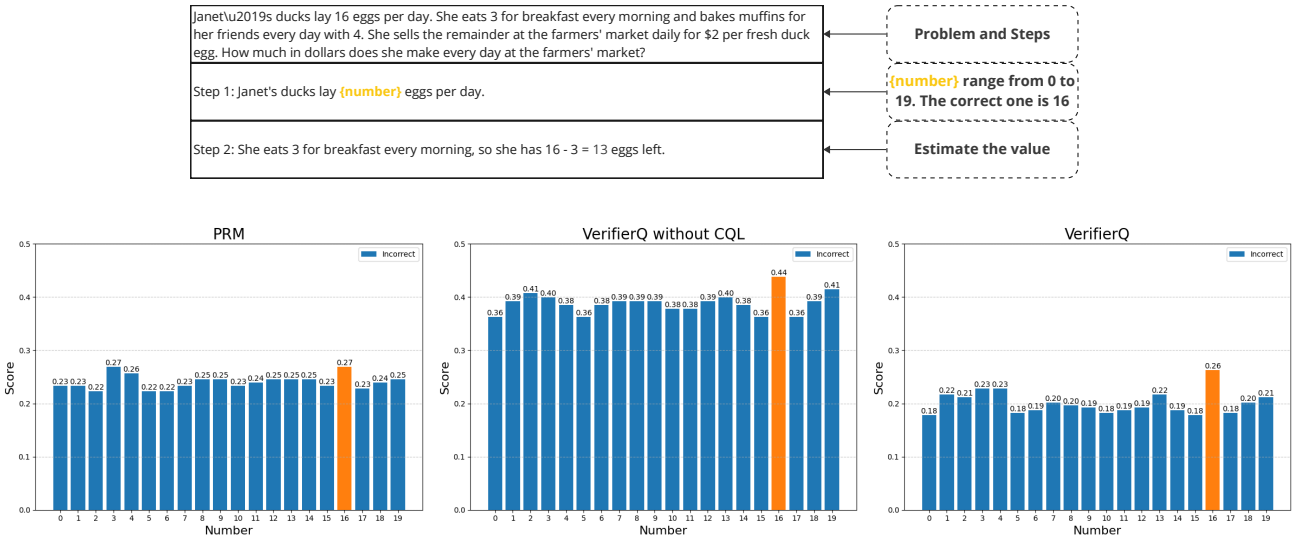

Figure 7: Overestimation case study: PRM (left) vs VerifierQ without CQL (middle) vs VerifierQ (right). Orange indicates correct value, blue indicates incorrect value.

= 30.6% compared to 11.9% and 12.6%) demonstrates that CQL improves the model's discriminative capability while maintaining reasonable estimates for valid solutions.