SELF-PLAY ENHANCEMENT VIA ADVANTAGE-WEIGHTED REFINEMENT IN ONLINE FEDERATED LLM FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent works have advanced feedback-based learning systems, whereby a foundation model is able to intake incoming feedback (e.g., from a user) to self-improve, creating a self-loop system of training. However, existing works are limited in needing to consider an offline setup to allow for such feedback-based methods, requiring privileged ground-truth contexts for training. Moreover, there is limited consideration of federated learning (FL), which is particularly well-suited for incorporating external feedback across large networks of end users, for example, but requires methods to be efficient for training on resource-constrained edge devices. Therefore, we introduce SPEAR (Self-Play Enhancement via Advantage-Weighted Refinement), an efficient online learning algorithm for federated LLM fine-tuning. SPEAR utilizes a *feedback-guided self-play loop* to construct naturally contrastive pairs per prompt which are used to train the model with (i) standard maximum likelihood on satisfactory completions and (ii) confidence-weighted unlikelihood on tail tokens of unsatisfactory completions. SPEAR requires only an interaction-level binary satisfaction signal together with non-answer feedback, without requiring a ground-truth answer to be provided, and does not need expensive group generations, meaning training both online and in a resource-efficient manner is feasible. We validate SPEAR across various benchmark datasets, demonstrating its superior performance in comparison to relevant baselines.

1 INTRODUCTION

Federated Learning (FL) (Konečnỳ et al., 2016; McMahan et al., 2017) has emerged as a promising paradigm for training machine learning (ML) models in a distributed and privacy-preserving manner. In FL, a network of edge devices (clients) collaboratively train a shared model by performing local updates on their private data and transmitting intermediate representations (e.g., model parameters) to a central server for aggregation. This framework eliminates the need to transfer sensitive data while still enabling the model to benefit from diverse data sources. Motivated by these advantages, recent work has begun exploring how to adapt FL for fine-tuning foundation models (Bommasani et al., 2021), such as large language models (LLMs) (Lee et al., 2025b; Ye et al., 2024; Fang et al., 2025).

In parallel, centralized LLM fine-tuning paradigms have increasingly focused on feedback-driven learning, where models iteratively improve by incorporating external feedback signals (Zhao et al., 2026). In addition, reinforcement learning-based fine-tuning and preference optimization, including approaches such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024), have demonstrated strong performance gains by aligning model outputs with desired behaviors.

Challenges: Despite these advances, there remains limited understanding of how feedback-driven training can be effectively integrated into FL settings. Specifically, existing approaches to feedback-based fine-tuning of LLMs are typically designed for centralized, offline pipelines that rely on privileged information (e.g., ground-truth answers or curated preference datasets) (Zhao et al., 2026; Song et al., 2026). In contrast, real-world deployments often require online learning from noisy and imperfect feedback sources, such as end users. Furthermore, while methods like GRPO provide an effective framework for preference-based optimization, they introduce significant computational overhead, making them challenging to deploy on resource-constrained edge devices in FL environ-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ments. These limitations highlight the need for efficient and online feedback-aware LLM fine-tuning strategies tailored to FL.

1.1 CONTRIBUTIONS

Driven by these challenges, we propose a *computationally efficient, online, and novel* learning algorithm suited for federated LLM fine-tuning. Our self-play enhancement via advantage-weighted refinement (SPEAR) algorithm uses a *feedback-guided self-play* loop to construct natural contrasting pairs: satisfactory completions (wins) and unsatisfactory completions (losses). SPEAR executes LLM fine-tuning via (i) a standard maximum-likelihood estimation on the win traces and (ii) a *confidence-gated unlikelihood* to suppress the high confidence tail tokens for unsatisfactory outputs. Overall, our key contributions are outlined as follows:

- *Online, Real-time Feedback-based FL.* We formulate a feedback-guided environment for fine-tuning of LLMs in an FL setting. Without requiring complete ground-truth answers in the feedback, we enable an online learning protocol for LLM fine-tuning.
- *Self-Play Enhancement Refinement Algorithm.* We propose a novel feedback-guided self-play enhanced methodology for LLM fine-tuning, which seeks to separate samples into win and lose traces to create contrasting samples for training. Via hyperparameters that control the confidence and number of tail tokens targeted for unsatisfactory lose traces, SPEAR's unlikelihood auxiliary loss discourages high-confidence tokens associated with unsatisfactory outputs. We show theoretically that a model whose SPEAR loss is bounded by ϵ satisfies a guaranteed per-token separation between win and active lose completions.
- *Experimental Validation.* We conduct extensive experiments on a diverse set of benchmark datasets, comparing SPEAR with existing relevant baselines. Via this, we show the superiority of SPEAR in not only accuracy, but also computational efficiency, meaning that even in an online setting with no privileged context, SPEAR still maintains superior performance.

2 RELATED WORK

Parameter Efficient Fine-tuning (PEFT): For fine-tuning existing foundation models (Bommasani et al., 2021), a common approach is to adopt PEFT techniques that do not require complete re-training of the model, resulting in computational efficiency and naturally suited for FL scenarios (Wang et al., 2019; Lee et al., 2025b; Kuo et al., 2024). Common PEFT methods include prefix-tuning (Li & Liang, 2021) and LoRA (Hu et al., 2022) (which adds and updates two low-rank matrices instead of the original weights), with the latter being the most popular option and the approach we adopt.

Federated LLM Fine-Tuning: A crucial component of FL (Konečnỳ et al., 2016; McMahan et al., 2017) is consideration for resource-constrained edge devices and enhancing computational efficiency (Sattler et al., 2019; Shahid et al., 2021; Imteaj et al., 2021). As a consequence, considerable attention has been given to fine-tuning pre-trained foundation models, particularly LLMs, in a FL setup with PEFT methods (Lee et al., 2025b). For example, Fang et al. (2025) consider how to train LLMs when their transmitted update sizes differ (dependent on each client's available resources) via a sketching mechanism while (Yuan et al., 2025) seek a method to work with LLMs in a fast and efficient manner in a cloud environment.

Feedback-based Learning: Feedback-guided fine-tuning involves the fine-tuning of pre-trained foundation models via an external feedback source, which is utilized to self-improve the model. In this realm, self-distillation (Zhao et al., 2026; Shenfeld et al., 2026) has emerged as an option, whereby the model aims to minimize the KL-divergence (Kullback & Leibler, 1951; Hinton et al., 2015) between the feedback-augmented output and original model output via knowledge distillation (KD) (Zhao et al., 2026). Other works have considered utilizing reinforcement learning (RL) (Kaelbling et al., 1996) in conjunction with a distillation process (Song et al., 2026). However, prior works such as Zhao et al. (2026) rely on privileged access to ground-truth contexts for feedback. In addition, RL-based approaches often require explicitly defined reward functions and can be computationally inefficient (Song et al., 2026), particularly when built upon algorithms such as GRPO due to multiple generations required for group calculation (Shao et al., 2024). Compared to prior work, SPEAR is crafted to operate in *online, imperfect feedback settings* and is capable of *preventing the reinforcement of undesired feedback-based outputs*, which can lead to model collapse (Shumailov et al., 2024).

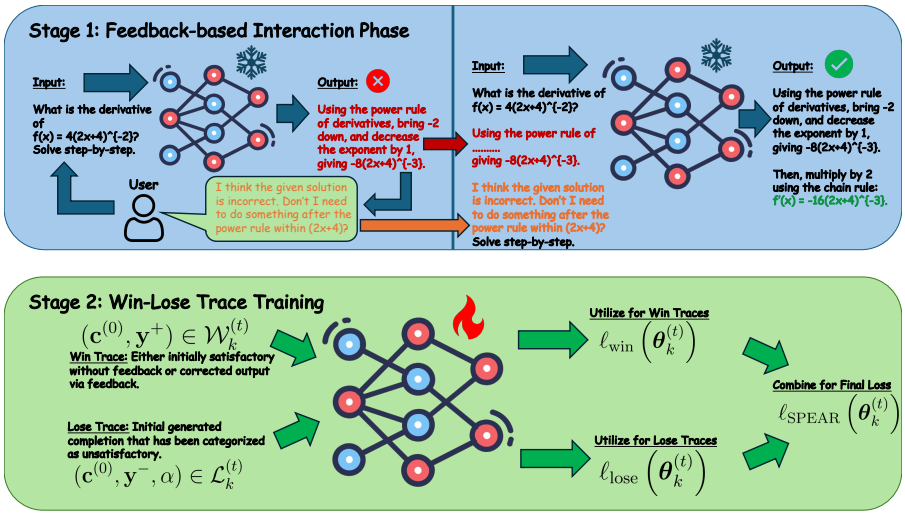


Figure 1: The two phases of the SPEAR algorithm. Firstly, the model interacts with an incoming feedback source (e.g., a user) to correct unsatisfactory generations. After the interaction phase, it categorizes the samples into wins and losses, which are then used to train a standard MLE and unlikelihood objective. This process repeats at each federated round t for each client selected for aggregation.

Unlikelihood Training: Unlikelihood training (Welleck et al., 2019), in contrast with standard supervised fine-tuning (SFT), seeks to minimize the probability that certain tokens are generated by taking the complement of the maximization probability. Unlikelihood training has been shown to be effective in discouraging certain outputs over various use cases, such as overuse of frequent words and excessive repetition (Li et al., 2020a; Lagutin et al., 2021). This naturally makes it suitable for self-play feedback based systems, where outputs can be susceptible to undesirable completions and therefore model collapse (Shumailov et al., 2024). SPEAR, through its confidence-gated, tail-targeted unlikelihood loss, is the first work to consider this dimension.

3 Proposed Methodology

3.1 Preliminaries and Problem Outline

We consider an *online, feedback-driven FL* setting, where a model is deployed across K distributed clients indexed by $k \in \{1, 2, \ldots, K\} = \mathcal{K}$ and interacts with an incoming feedback source (e.g., a human user). Unlike standard supervised or offline preference-based learning, SPEAR does not require ground-truth answers or curated preferred responses. Instead, it uses a binary satisfaction signal and partial, non-answer feedback, allowing multiple distinct responses to be treated as satisfactory.

At each client k, a private, but evolving incoming data stream exists, where this incoming data is considered *off-policy* (i.e., not generated by the current model) and follows a distribution $\mathbf{x}_k \sim \mathcal{D}_k$. For a given prompt $\mathbf{x}_k \sim \mathcal{D}_k$ formed from the incoming data stream, the model first produces an initial completion. Upon observing this completion, an existing (e.g., end user) feedback source provides feedback $\mathbf{f}$, which may contain partial hints or intermediate reasoning, but does not reveal the full ground-truth answer (Stage 1 of Fig. 1). The feedback can be appended to the end of the initial context to form a revised input, enabling iterative refinement.

This process induces a *trajectory* per prompt consisting of an initial generation and subsequent feedback-conditioned revisions. These trajectories form the fundamental training signal in our setting, rather than static labeled pairs. In particular, each trajectory can be categorized into successful (i.e., initially satisfactory or corrected) and unsuccessful generations, which are later used to construct contrasting training objectives. These generations can be formed from the following process: Given a context $\mathbf{c}$ (a token sequence), the model defines

$$p_{\boldsymbol{\theta}}(v \mid \mathbf{c}) = \text{softmax}(z_{\boldsymbol{\theta}}(\mathbf{c}))_v, \quad v \in \mathcal{V},$$

where $z_{\boldsymbol{\theta}}(\mathbf{c}) \in \mathbb{R}^{|\mathcal{V}|}$ are the logits, $\mathcal{V}$ is the vocabulary, and $\boldsymbol{\theta}$ captures the model parameters. Then, for a token sequence $\mathbf{y} = (y_0, \ldots, y_{T-1})$, the likelihood factorizes as:

$$p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c}) = \prod_{i=0}^{T-1} p_{\boldsymbol{\theta}}(y_i \mid \mathbf{c}, \mathbf{y}_{<i}), \tag{1}$$

where $\mathbf{y}_{<i} = (y_0, \ldots, y_{i-1})$.

These trajectory-inducing interactions occur locally on each client within a federated system coordinated by a central server over communication rounds $t \in \{0, 1, \ldots, S\}$. At round t, a subset of clients $\mathcal{K}_t \subseteq \mathcal{K}$ is selected to perform local updates based on their collected interaction trajectories. Each client maintains local model parameters $\boldsymbol{\theta}_k^{(t)}$, initialized from the global model $\boldsymbol{\theta}^{(t)}$. After local training, updated parameters are transmitted to the server and aggregated via Federated Averaging (FedAvg) (Konečnỳ et al., 2016) based on the number of initially satisfactory or corrected generations:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \sum_{k \in \mathcal{K}_t} \frac{|\mathcal{W}_k^{(t)}| \cdot \boldsymbol{\theta}_k^{(t)}}{\sum_{k' \in \mathcal{K}_t} |\mathcal{W}_{k'}^{(t)}|},$$

where $\mathcal{W}_k^{(t)}$ is called the win set for client k at round t (specifics outlined in Sec. 3.2.1). Overall, the goal is to learn model parameters $\boldsymbol{\theta}$ that effectively leverage both successful and unsuccessful interaction trajectories under partial feedback, while remaining computationally efficient for deployment in FL environments. More detailed specifics are outlined in the subsequent sections (Sec. 3.2).

3.2 SPEAR: Self-Play Enhancement via Advantage-Weighted Refinement

As outlined in Sec. 1, since the goal is to fine-tune the model in an online manner efficiently while interacting with an entity such as a user, it is important that feedback $\mathbf{f}$ does not require a complete ground-truth answer during training. Therefore, SPEAR consists of two phases for each client in each round: (i) an *interaction phase* that generates and categorizes successful completions and (ii) a *training phase* that optimizes for both satisfactory and unsatisfactory traces produced from the model.

3.2.1 Phase 1: Interaction Phase

Firstly, during the interaction phase (Stage 1 of Fig. 1) at the start of each FL round, for each prompt $\mathbf{x}_{k,j} \in \mathcal{B}_k$, where $\mathcal{B}_k$ denotes a minibatch on client k and j indexes a sample within a batch, client k will encode $\mathbf{x}_{k,j}$ to form context $\mathbf{c}_{k,j}^{(0)}$ and generate $\mathbf{y}_{k,j}^{(0)} \sim p_{\boldsymbol{\theta}_k}(\cdot|\mathbf{c}_{k,j}^{(0)})$. Then, the user interacting with the model can indicate whether the generated answer is satisfactory, assigned to binary function $\mathsf{IsSatisfied}(\mathbf{y}_{k,j}^{(0)}, \mathbf{x}_{k,j})$. If the user deems the answer as satisfactory, tuple $(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)})$ is added to *win set* $\mathcal{W}_k^{(t)}$. If deemed unsatisfactory, revision is required for those traces. In cases where $\mathbf{y}_{k,j}^{(0)}$ is unsatisfactory, it is assumed that the user may still possess partial knowledge relevant to the task. This can be expressed through informative feedback (e.g., intermediate reasoning steps or the identification of potential errors) even if the user is unable to provide the final step. Therefore, the model receives feedback $\mathbf{f}_{k,j} \leftarrow \mathsf{FB}(\mathbf{x}_{k,j}, \mathbf{y}_{k,j}^{(0)})$, and constructs revision context

$$\mathbf{c}_{k,j}^{(1)} = \mathbf{c}_{k,j}^{(0)} \parallel \mathbf{y}_{k,j}^{(0)} \parallel \mathbf{f}_{k,j}, \tag{2}$$

which is utilized to generate a revised answer $\mathbf{y}_{k,j}^{(1)} \sim p_{\boldsymbol{\theta}_k}(\cdot|\mathbf{c}_{k,j}^{(1)})$.

After revision is conducted N times (defined by each client/user), the revision traces are re-classified. If $\mathsf{IsSatisfied}(\mathbf{y}_{k,j}^{(n)}, \mathbf{x}_{k,j})$ is 1 (where n, $1 \leq n \leq N$, is the number of revisions attempted for a sample so far), then the model has successfully self-corrected given the feedback, and $(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(n)})$ is added to $\mathcal{W}_k^{(t)}$, as we would like the model to learn to produce $\mathbf{y}_{k,j}^{(n)}$ from *original prompt* $\mathbf{c}_{k,j}^{(0)}$. For traces that failed to produce a satisfactory answer, $(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)})$ is added to *lose set* $\mathcal{L}_k^{(t)}$. We further categorize two types of lose traces: (i) ones where the initial generated answer is unsatisfactory but feedback generation is successful and (ii) those where both attempts (initial and revision) fail. In the first case, we are more confident in the lack of satisfaction for the trace, as we have received indication

definitively via the corrected trace. In the second case, while $\mathbf{y}_{k,j}^{(0)}$ is indicated to be unsatisfactory, we have no concrete alternative. Therefore, we define the final construction of lose set $\mathcal{L}_k^{(t)}$ as

$$\mathcal{L}_k^{(t)} = \left\{ \left((\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)}),\ \alpha_{k,j} \right) \;\middle|\; \alpha_{k,j} = \begin{cases} 1 & \text{if } \mathsf{IsSatisfied}(\mathbf{y}_{k,j}^{(n)}, \mathbf{x}_{k,j}) \\ 0.5 & \text{otherwise} \end{cases} \right\}, \tag{3}$$

where $\alpha_{k,j}$ is the confidence weight assigned to each trace in the lose set. In this way, traces whose lack of satisfaction we are more confident about are assigned a higher weight.

3.2.2 Phase 2: Training Contrasting Objectives

After the interaction phase, training on the collected traces begins (Stage 2 of Fig. 1). For a client k at round t, SPEAR first seeks to engage in maximum likelihood estimation (MLE) via the standard negative log-likelihood (NLL) (Shen, 2024) on satisfactory completions from win set $\mathcal{W}_k^{(t)}$, i.e.,

$$\ell_{\text{win}}\left(\boldsymbol{\theta}_k^{(t)}\right) = -\mathbb{E}_{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}) \sim \mathcal{W}_k^{(t)}} \left[\sum_{i=0}^{|\mathbf{y}_{k,j}|-1} \log p_{\boldsymbol{\theta}}(y_{k,j,i} | \mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j,<i}) \right]. \tag{4}$$

While standard supervised fine-tuning (SFT) on only the win traces is important to ensuring that we do not encourage the model to learn to produce unsatisfactory outputs (and thereby degrade performance), it leaves an entire set of samples generated from lose set $\mathcal{L}_k^{(t)}$ that are not utilized. Therefore, to fully exploit the information available to each client, we perform a contrastive separation between satisfactory and unsatisfactory traces, pushing away tokens associated with the latter, similar to metric learning (Chen et al., 2020; Lee et al., 2025a). To accomplish this, for a lose trace $(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{-}, \alpha_{k,j}) \in \mathcal{L}_k^{(t)}$, SPEAR seeks to reduce the probability of generating an unsatisfactory completion via an *unlikelihood* objective (Welleck et al., 2019; Xhonneux et al., 2024). To this end, instead of directly applying the unlikelihood objective to the lose traces, we selectively target tokens using two hyperparameters: a confidence-weighted unlikelihood margin, $\mu \in (0, 1)$, and a tail-token selection parameter, τ. The intuition is twofold. First, issues in unsatisfactory completions tend to concentrate toward the end of the sequence, motivating us to focus on the final τ tokens (we validate this in Appendix B.1). Second, we penalize only high-probability tokens associated with unsatisfactory outputs. The confidence-weighted margin, μ, operationalizes this by imposing a threshold on token probabilities, such that only high-confidence issues are discouraged, while low-confidence predictions are preserved to avoid over-penalization. With this, we define active token index set on a completion $\mathbf{y}_{k,j}^{-}$ as

$$\mathcal{T}_\tau(\mathbf{y}_{k,j}^{-}) \;=\; \begin{cases} \left\{ \, |\mathbf{y}_{k,j}^{-}| - \min(\tau, |\mathbf{y}_{k,j}^{-}|),\ \ldots,\ |\mathbf{y}_{k,j}^{-}| - 1 \, \right\} & \tau > 0 \\ \left\{ 0,\ \ldots,\ |\mathbf{y}_{k,j}^{-}| - 1 \right\} & \tau = 0, \end{cases} \tag{5}$$

with the full completion targeted when $\tau = 0$. $|\mathbf{y}_{k,j}^{-}|$ denotes the cardinality of the sequence. Then, with $\mathcal{T}_\tau(\mathbf{y}_{k,j}^{-})$, the confidence-gated, tail-targeting unlikelihood objective can be described as

$$\begin{aligned} \ell_{\text{lose}}\left(\boldsymbol{\theta}_k^{(t)}\right) = -\mathbb{E}_{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{-}, \alpha_{k,j}) \sim \mathcal{L}_k^{(t)}} \Bigg[\alpha_{k,j} \sum_{i \in \mathcal{T}_\tau(\mathbf{y}_{k,j}^{-})} \mathbb{1}\Big[p_{\boldsymbol{\theta}}\Big(y_{k,j,i}^{-} \mid \mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j,<i}^{-} \Big) > \mu \Big] \\ \cdot \log\Big(1 - p_{\boldsymbol{\theta}}\Big(y_{k,j,i}^{-} \mid \mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j,<i}^{-} \Big) \Big) \Bigg]. \end{aligned} \tag{6}$$

Therefore, the loss function of SPEAR finalizes to

$$\ell_{\text{SPEAR}}\left(\boldsymbol{\theta}_k^{(t)}\right) = \lambda_w \ell_{\text{win}}\left(\boldsymbol{\theta}_k^{(t)}\right) + \lambda_l \ell_{\text{lose}}\left(\boldsymbol{\theta}_k^{(t)}\right), \tag{7}$$

where λ_w and λ_l are weighting factors for the SFT and unlikelihood objectives respectively. After training for E local iterations, the parameters are transmitted to the server for aggregation. When aggregating at the server, the aggregation weight given to each client is relative to the number of *win traces* on client k for that iteration t, as we would like to prioritize clients that give more satisfactory answers. In an online setting, the number of samples continuously evolves, making it inappropriate to consider the number of samples for weighting to be fixed, as is common with traditional FedAvg.

4 ANALYSIS

We now provide a theoretical characterization of the SPEAR objective, showing that it implicitly enforces a *log-probability margin* between win and lose completions. This property is specific to the joint structure of SPEAR's NLL and confidence-gated unlikelihood objectives, and yields justification for its hyperparameters. We demonstrate this behavior by analyzing the objective on a single paired sample, which suffices to capture the underlying mechanism. Concretely, we consider a win trace $(\mathbf{c}^{(0)}, \mathbf{y}^+) \in \mathcal{W}$ and a lose trace $(\mathbf{c}^{(0)}, \mathbf{y}^-, \alpha) \in \mathcal{L}$ sharing a common initial context $\mathbf{c}^{(0)}$.

Assumption 1 (Active Token Confidence)**.** *All tokens in the active tail set exceed the confidence threshold, i.e.,* $p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-) > \mu, \quad \forall i \in \mathcal{T}_\tau(\mathbf{y}^-)$.

Assumption 2 (Single Sample)**.** *The sets $\mathcal{W}$ and $\mathcal{L}$ each contain exactly one trace, namely* $(\mathbf{c}^{(0)}, \mathbf{y}^+)$ *and* $(\mathbf{c}^{(0)}, \mathbf{y}^-, \alpha)$*, respectively.*

Margin definition. We define the *SPEAR log-probability margin* for a paired win-lose sample as:

$$M_\tau(\boldsymbol{\theta};\, \mathbf{y}^+, \mathbf{y}^-) \;:=\; \underbrace{\frac{1}{|\mathbf{y}^+|} \log p_{\boldsymbol{\theta}}(\mathbf{y}^+ \mid \mathbf{c}^{(0)})}_{\text{win per-token log-prob}} - \underbrace{\frac{1}{|\mathcal{T}_\tau(\mathbf{y}^-)|} \sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \log p_{\boldsymbol{\theta}}\Big(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-\Big)}_{\text{active-lose per-token log-prob}}, \tag{8}$$

where, since $p_{\boldsymbol{\theta}} \in (0, 1]$, both log-probability terms lie in $(-\infty, 0]$. A larger M_τ indicates the model assigns higher per-token probability to $\mathbf{y}^+$ and lower probability to the active tail tokens of $\mathbf{y}^-$, i.e., a stronger preference for the win over the lose completion.

Theorem 1 (SPEAR Log-Probability Margin)**.** *Under Assumptions 1 and 2, suppose* $\ell_{\mathrm{SPEAR}}(\boldsymbol{\theta}) \le \epsilon$ *for the paired win-lose sample above. Then*

$$M_\tau(\boldsymbol{\theta};\, \mathbf{y}^+, \mathbf{y}^-) \;\ge\; h(\mu) \;-\; \epsilon\left(\frac{1}{\lambda_w\, |\mathbf{y}^+|} \;+\; \frac{1}{\lambda_l\, \alpha\, |\mathcal{T}_\tau(\mathbf{y}^-)|}\right), \tag{9}$$

where the confidence-amplified margin constant is

$$h(\mu) \;=\; \log \frac{1}{\max\!\left(\frac{1}{4},\; \mu(1-\mu)\right)} \;=\; \begin{cases} \log 4 & \mu \le \frac{1}{2}, \\ \log \dfrac{1}{\mu(1-\mu)} & \mu > \frac{1}{2}. \end{cases} \tag{10}$$

The derivation is provided in full in Appendix E. While the theorem utilizes a single win-lose pair to make the mechanism transparent, the objective-level result extends directly to arbitrary finite minibatches (Appendix E.3). For the derived result, we highlight two implications.

Universal minimum margin. For all $\mu \in (0, 1)$, $h(\mu) \ge \log 4$, with equality at $\mu = 1/2$. Consequently, as $\epsilon \to 0$, any minimizer of ℓ_{SPEAR} achieves at least $\log 4$ separation up to the error term between win and active-lose distributions. Crucially, $\log 4$ is the *tight* constant arising from the algebraic coupling between ℓ_{win} and ℓ_{lose}: the inequality $4p(1-p) \le 1$ holds for all $p \in [0, 1]$ with equality at $p = \frac{1}{2}$, reflecting the maximum entropy configuration of a binary token decision.

Coverage-strength tradeoff. For $\mu > \frac{1}{2}$, $h(\mu) > \log 4$ and grows monotonically to $+\infty$ as $\mu \to 1$, so the guaranteed per-token margin strictly strengthens with μ. However, Theorem 1 characterizes only tokens that *exceed* the threshold; tokens below μ are excluded from penalization entirely. This creates a coverage-strength tradeoff that motivates μ as a hyperparameter to tune: a larger μ yields a stronger per-token margin guarantee but a smaller active set $\mathcal{T}_\tau(\mathbf{y}^-)$, reducing the gradient signal available during training. Empirically, values of μ well below 0.5 can outperform larger thresholds. The confidence weight α plays an analogous role: a larger α (assigned to lose traces where self-correction succeeded, indicating higher certainty of dissatisfaction) causes the margin in equation 9 to increase, further justifying SPEAR's distinction between the two lose-trace types. We also note that for a larger tail token selection parameter τ, the RHS of equation 9 increases, meaning a larger margin. While this would seem to indicate targeting the entire unsatisfactory trace would be beneficial, as outlined in Sec. 3.2.2, unsatisfactory reasoning will concentrate towards the end of the completion, meaning targeting the entire completion will result in tokens we want close to be pushed away, resulting in suboptimal performance. We verify this empirically in Appendix B.1.

5 EXPERIMENTAL RESULTS

Experimental Setup. We apply SPEAR to two widely used open-source LLM families: Qwen2.5 (1.5B) (Bai et al., 2023; Yang et al., 2024; Qwen Team, 2024) and Llama3.2 (3B) (Grattafiori et al., 2024; Touvron et al., 2023) (Additional experiments on Qwen2.5-7B and Qwen3.5-2B can be found in Appendix B.9 and B.11). These models were selected to capture performance across varying model scales, while remaining representative of the smaller footprints typical in client-side FL settings. For the FL network, 50 total clients are considered, with each client attempting to correct itself with feedback a maximum of two times ($N = 2$) for each sample (ablation against $N = 1$ found in Appendix B.4). We consider the usage of LoRA (Hu et al., 2022) as our PEFT method, targeting the attention and projection modules at each layer. For all experiments, the AdamW optimizer is utilized (Loshchilov & Hutter, 2017), with linear warmup and cosine decay on the learning rate (Alimisis et al., 2025). All experiments are conducted on a server with an NVIDIA A100-40GB GPU, utilizing the HuggingFace (Jain, 2022) and PyTorch (Paszke et al., 2019) libraries for implementation. More detailed specifications can be found in Appendix C.

Datasets. We consider four benchmark datasets encompassing a diverse range of domains: ARC-Challenge (Clark et al., 2018) for science-based question answering, HellaSwag (Zellers et al., 2019) for common-sense reasoning sentence completion, MathMCQA for competition-level mathematics (Biderman, 2025), and StrategyQA (Geva et al., 2021) for multi-hop reasoning (an open-ended setting on MBPP can be found in Appendix B.13). We measure the accuracy of the final produced answer after reasoning from the testing dataset to evaluate the performance of SPEAR and the baselines. In terms of the incomplete but informative feedback given to the model, for HellaSwag and ARC-Challenge, we include the first 2-3 words of the correct completion. MathMCQA's feedback includes the first three lines or 300 characters of the step-by-step mathematical reasoning field (whichever is shorter). StrategyQA's feedback encompasses half of the "facts" column for each sample. Importantly, SPEAR does not require the ground-truth answer to be exposed to the model. For benchmark reproducibility, we instantiate the binary satisfaction signal using task correctness as a proxy for user satisfaction. The reference answer itself is never provided as a training target. Additional results demonstrating SPEAR's performance on comparatively weaker feedback can be found in Appendix B.12. Specific text templates utilized can be found in Appendix D.

Baselines. We evaluate SPEAR against four recent relevant approaches: GRPO (Shao et al., 2024), OPSD (Zhao et al., 2026), RLTF-SD (Song et al., 2026), and a feedback-based supervised fine-tuning variant without unlikelihood training (Feedback SFT). The RLTF-SD implementation uses a batch rollout strategy (collecting all rollouts upfront before multiple gradient updates) for time-efficiency. These baselines were selected to span the dominant paradigms for improving reasoning in foundation models: RL with explicit reward design (GRPO), self-distillation via structured feedback (OPSD), a mixture of both RL and self-distillation (RLTF-SD), and feedback-driven supervised fine-tuning without unlikelihood loss considerations (Feedback SFT). Importantly, GRPO, RLTF-SD, and OPSD do not operate in the setting we consider: GRPO and RLTF-SD rely on explicitly specified reward functions, while OPSD assumes access to privileged or high-quality ground-truth feedback during distillation. In contrast, SPEAR targets *online learning from imperfect feedback*, where neither explicit reward functions nor ground-truth answers are provided. To the best of our knowledge, prior work has not studied this combination of constraints, and therefore there are no existing baselines for direct comparison. As a result, we compare against the closest established methods (additional results comparing against a DPO-based baseline can be found in Appendix B.10 and B.11).

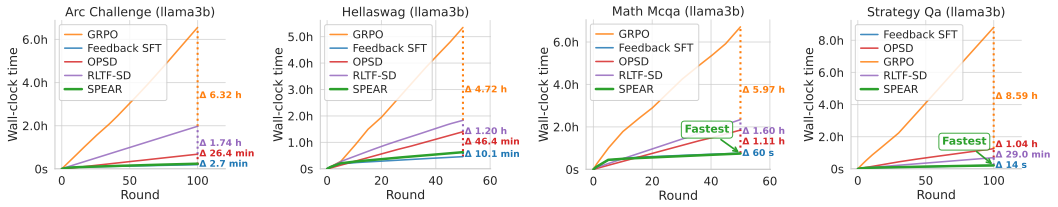
Additional Experimental Results. Additional experimental results not included in the main text (e.g., additional ablation studies, additional clients, etc.) can be found in Appendix B.

5.1 RESULTS

Performance in Comparison to Baselines. Considering the results in Table 1, we note the superiority of SPEAR in comparison to all baselines across all evaluated datasets. In particular, we note the consistent high performance of SPEAR across all scenarios, while some baselines struggle (e.g., GRPO and RLTF-SD achieve only 40% and 45% on Llama for MathMCQA). We can also see engaging in pure feedback SFT on only the satisfactory traces (Feedback SFT) results in performance

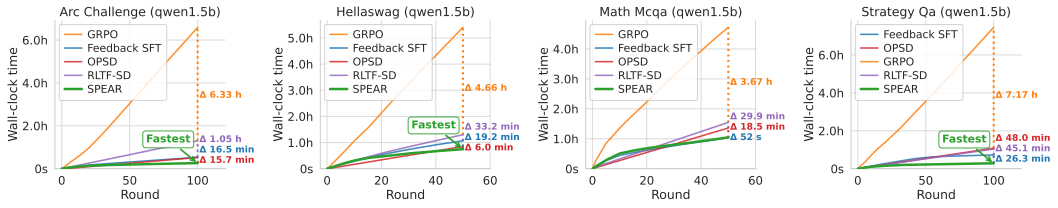
Table 1: Performance of SPEAR vs. baselines on Llama3.2-3B and Qwen2.5-1.5B across 3 random seeds, showing that SPEAR outperforms the baselines across multiple models and datasets.

Model	Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Llama3.2-3B	Feedback SFT	67.95±1.45	84.60±0.98	48.60±9.89	49.49±1.76	62.66
	GRPO	65.70±3.48	75.99±1.53	40.20±14.89	49.34±1.51	57.81
	OPSD	24.69±1.55	25.03±0.64	0.00±0.00	0.00±0.00	12.43
	RLTF-SD	63.28±3.58	60.05±12.30	44.70±7.92	51.58±0.08	54.90
	SPEAR (Ours)	69.17±0.85	86.72±0.67	56.70±0.78	65.84±0.78	**69.61**
Qwen2.5-1.5B	Feedback SFT	73.89±3.38	76.14±3.61	54.93±2.04	50.51±1.76	63.87
	GRPO	76.30±0.17	79.00±1.65	50.87±3.86	51.38±6.44	64.39
	OPSD	25.03±1.23	24.92±0.69	0.00±0.00	0.00±0.00	12.49
	RLTF-SD	75.03±0.40	60.65±20.78	44.70±7.41	54.29±10.08	58.67
	SPEAR (Ours)	76.93±1.41	80.65±0.79	59.07±1.16	66.08±3.20	**70.68**



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

(a) Llama3.2-3B



(e) ARC-Challenge (f) HellaSwag (g) MathMCQA (h) StrategyQA

(b) Qwen2.5-1.5B

Figure 2: Cumulative training time of each algorithm on (a) Llama-3B and (b) Qwen-1.5B models. Δ delineates the final wall-clock time difference between the corresponding algorithm and SPEAR. When SPEAR achieves the fastest final performance, it's marked with a "Fastest" box.

that lags behind SPEAR, giving credence to the use of our unlikelihood-based loss. Another important observation is the collapse of self-distillation based algorithm OPSD on all datasets in comparison to other baselines. We attribute this phenomenon to the reliance of self-distillation techniques on high-quality feedback-generated outputs. Especially in the case of smaller models typically used in edge settings, the initial feedback-generated responses may be suboptimal and/or outputting an invalid format, leading the KL objective to guide the model toward an undesired distribution. We especially believe this to be the case due to the fact that the performance of OPSD on MathMCQA and StrategyQA achieves specifically 0.0, which is below random chance, meaning that OPSD is learning to match the KL divergence, but is aligning it with suboptimal completions generated from the feedback due to the lack of checking of the feedback produced answer. Lastly, we also observe that the margin of improvement with SPEAR in comparison with the baselines is greater on tasks that require longer chain-of-thought (CoT) reasoning, as observed with MathMCQA and StrategyQA.

Wall-clock Time Comparison. In Fig. 2, we compare the wall-clock training time of SPEAR and the baselines over the course of federated training. For GRPO and RLTF-SD, due to memory

constraints, the number of gradient accumulation steps is increased ($\times$2(Llama)/$\times$4(Qwen) for ARC and HellaSwag, $\times$2 for MathMCQA, and $\times$4 for StrategyQA) and group size is kept low (specifics found in Appendix C). We can see across both Llama3.2-3B and Qwen2.5-1.5B, SPEAR remains the fastest algorithm across a majority of cases, with the difference between Feedback SFT and SPEAR being small when the former is quicker than SPEAR. We find that SPEAR outperforms Feedback SFT in most scenarios, despite the added cost of computing the confidence-gated tailed unlikelihood loss. This advantage arises because SPEAR achieves higher accuracy, reducing the number of self-play feedback loops required and thereby improving overall time efficiency. This is also the reason why SPEAR's cumulative time is not always linear; with fewer corrections needed as the model improves, the total number of expensive generations decreases. GRPO and RLTF-SD suffer from substantial memory overhead due to their group-based computation, which necessitates additional gradient accumulation steps and smaller group sizes, which is particularly problematic in resource-constrained federated settings. Combined with extra generation steps, this leads to considerably slower training than SPEAR. While RLTF-SD does not suffer as much as GRPO due to early termination for already satisfactory completions, its reliance on GRPO-based group generation still adds significant overhead in comparison to SPEAR.

Table 2: Ablation study on μ for SPEAR across 3 random seeds for Llama3.2-3B and Qwen2.5-1.5B. SPEAR's accuracy remains stable to differing μ hyperparameter values, indicating robustness.

Model	μ	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA
Llama3.2-3B	0.1	67.92±1.79	**87.09**±0.85	**56.27**±2.36	64.68±3.61
	0.5	**69.25**±1.25	86.10±0.58	55.70±2.56	67.20±0.69
	0.7	**69.25**±1.37	87.00±0.28	54.60±1.30	**68.51**±2.59
	0.9	69.00±1.00	85.95±0.93	52.50±1.93	66.23±2.70
Qwen2.5-1.5B	0.1	**77.25**±1.14	**79.69**±3.34	**58.80**±1.21	**66.86**±1.03
	0.5	63.23±21.81	78.81±2.26	56.77±1.53	66.62±1.25
	0.7	77.08±0.49	78.27±1.80	57.80±0.72	64.05±3.28
	0.9	75.09±0.98	79.41±1.53	57.27±1.34	66.62±1.69

Ablation over μ. We study how the μ hyperparameter influences the training process of SPEAR. Based on the results in Table 2, we can note that in general, setting μ to a lower threshold more greatly benefits the training. This could be due to the fact that with a lower threshold, more tokens are targeted in the unlikelihood loss, leading to a greater gradient signal (as outlined in Sec. 4). However, we can see that depending on the setting, the optimal setting differs (e.g., 0.5/0.7 works best for ARC on Llama or 0.7 for StrategyQA on Llama). This indicates that μ is a hyperparameter to be tuned rather than merely setting to > 0.5 or the lowest value possible. An important point, however, *is that in general, the performance of SPEAR still remains high across differing settings* (with the exception of 0.5 for ARC on Qwen2.5-1.5B), indicating that the methodology is *not overly sensitive to hyperparameter tuning with respect to μ.* For example, the difference between $\mu = 0.1$ and $\mu = 0.9$ on StrategyQA for Qwen2.5-1.5B is less than 0.5%.

6 Conclusion

In this paper, we introduced SPEAR, a novel *online, self-play federated fine-tuning* algorithm that works with *imperfect, but informative feedback sources*. In SPEAR, by deliberately separating generated completions into win and lose traces, we allow the model to maximize information available to it and discourage undesirable token generation through a confidence-gated, tail-targeted unlikelihood objective. Through extensive experiments against relevant baselines across a diverse set of datasets, we demonstrated the effectiveness and superiority of SPEAR in both training efficiency and performance. Despite the benefits demonstrated by SPEAR, a potential limitation and future avenue of our work is the need for the feedback to be related to the task: for example, if the user gives feedback that is contrary or irrelevant to the task at hand, it can lead to a scenario where the majority of completions are unsatisfactory, leading to no win traces to sample from.

AI USE STATEMENT

In this work, we used generative AI tools to improve the clarity of the manuscript (e.g., wording, paraphrasing, and readability) and for double-checking the manuscript for inconsistencies (e.g., typos, notation inconsistencies). Moreover, it was used as an aid in the creation of tables throughout the manuscript and code implementation. Research ideation of the proposed methodology was derived from the authors, and the rest of the required disclosure tasks (e.g., generation of new synthetic datasets) are not applicable to this work. We have reviewed all AI-assisted work. As some examples, we have verified the accuracy of the hyperparameter settings in the tables and ensured the accuracy of the generated text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For reproducibility, we provide the implementation code of the SPEAR method in the supplementary material of the submission. Moreover, the hyperparameter settings and prompting templates used across the experiments can be found in Appendix C and D respectively. The pseudocode of SPEAR can additionally be found in Appendix A. Lastly, the derivation of the Theorem in Sec. 4 and its associated batch-level extension can be found in Appendix E.

REFERENCES

Foivos Alimisis, Rustem Islamov, and Aurelien Lucchi. Why do we need warm-up? a theoretical perspective. *arXiv preprint arXiv:2510.03164*, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Gustav A Baumgart, Jaemin Shin, Ali Payani, Myungjin Lee, and Ramana Rao Kompella. Not all federated learning algorithms are created equal: A performance evaluation study. *arXiv preprint arXiv:2403.17287*, 2024.

Stella Biderman. Math-mcqa: A multiple choice adaptation of the math dataset, 2025. URL `https://huggingface.co/datasets/stellaathena/math_mcqa`.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PmLR, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Wenzhi Fang, Dong-Jun Han, Liangqi Yuan, Seyyedali Hosseinalipour, and Christopher G Brinton. Federated sketching lora: On-device collaborative fine-tuning of large language models. *arXiv preprint arXiv:2501.19389*, 2025.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Peter J Green and Sylvia Richardson. Modelling heterogeneity with and without the dirichlet process. *Scandinavian journal of statistics*, 28(2):355–375, 2001.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M Hadi Amini. A survey on federated learning for resource-constrained iot devices. *IEEE Internet of Things Journal*, 9(1):1–24, 2021.

Shashank Mohan Jain. Hugging face. In *Introduction to transformers for NLP: With the hugging face library and models to solve problems*, pp. 51–67. Springer, 2022.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Kevin Kuo, Arian Raje, Kousik Rajesh, and Virginia Smith. Federated lora with sparse communication. *arXiv preprint arXiv:2406.05233*, 2024.

Evgeny Lagutin, Daniil Gavrilov, and Pavel Kalaidin. Implicit unlikelihood training: Improving neural text generation with reinforcement learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 1432–1441, 2021.

Seohyun Lee, Wenzhi Fang, Anindya Bijoy Das, Seyyedali Hosseinalipour, David J Love, and Christopher G Brinton. Cooperative decentralized backdoor attacks on vertical federated learning. *IEEE Transactions on Networking*, 34:2004–2019, 2025a.

Seohyun Lee, Wenzhi Fang, Dong-Jun Han, Seyyedali Hosseinalipour, and Christopher G Brinton. Tap: Two-stage adaptive personalization of multi-task and multi-modal foundation models in federated learning. *arXiv preprint arXiv:2509.26524*, 2025b.

Margaret Li, Stephen Roller, Ilia Kulikov, Sean Welleck, Y-Lan Boureau, Kyunghyun Cho, and Jason Weston. Don't say that! making inconsistent dialogue unlikely with unlikelihood training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4715–4728, 2020a.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. Pmlr, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Hamza Reguieg, Mohammed El Hanjri, Mohamed El Kamili, and Abdellatif Kobbane. A comparative evaluation of fedavg and per-fedavg algorithms for dirichlet distributed heterogeneous data. In *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pp. 1–6. IEEE, 2023.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9):3400–3413, 2019.

Osama Shahid, Seyedamin Pouriyeh, Reza M Parizi, Quan Z Sheng, Gautam Srivastava, and Liang Zhao. Communication efficiency in federated learning: Achievements and challenges. *arXiv preprint arXiv:2107.10996*, 2021.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ming Shen. Rethinking data selection for supervised fine-tuning. *arXiv preprint arXiv:2402.06094*, 2024.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback. *arXiv preprint arXiv:2602.02482*, 2026.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE journal on selected areas in communications*, 37(6):1205–1221, 2019.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. *arXiv preprint arXiv:1908.04319*, 2019.

Sophie Xhonneux, Alessandro Sordoni, Stephan Günnemann, Gauthier Gidel, and Leo Schwinn. Efficient adversarial training in llms with continuous attacks. *Advances in Neural Information Processing Systems*, 37:1502–1530, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. Openfedllm: Training large language models on decentralized private data via federated learning. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 6137–6147, 2024.

Liangqi Yuan, Dong-Jun Han, Shiqiang Wang, and Christopher Brinton. Local-cloud inference offloading for llms in multi-modal, multi-task, multi-dialogue settings. In *Proceedings of the Twenty-sixth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pp. 201–210, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Appendix

A Pseudocode of SPEAR **15**

A.1 Client-Side SPEAR 15

A.2 Server-side Aggregation 16

B Additional Experimental Results **16**

B.1 Analysis on Assumption of Issues Near Completion End 16

B.1.1 Ablation over τ 16

B.1.2 Initial Error Localization 16

B.1.3 Qualitative Example on MathMCQA 17

B.2 High Number of Clients with High Heterogeneity 18

B.3 Fine-tuning on Only Last H Layers 19

B.4 Attempting Revision Only Once 19

B.5 Additional Comparisons of $N = 1$ vs. $N = 2$ 20

B.6 Ablation on λ_l 22

B.7 SPEAR on Differing FL Methods 23

B.8 Centralized Setting 23

B.9 Larger Model Size: Qwen2.5-7B 24

B.10 Comparison of SPEAR against DPO-based method 24

B.11 Experiment Over Newer Model Family 24

B.12 SPEAR under Weaker Feedback 25

B.12.1 Correction Percentage before Training under Weaker Feedback 25

B.12.2 Accuracy on Weaker Feedback 25

B.13 Open-Ended Scenario with Weak Feedback 26

C Detailed Hyperparameter Setup **27**

C.1 Hyperparameter Settings (SPEAR) 27

C.2 Hyperparameter Settings of baselines (GRPO / OPSD / RLTF-SD) 28

D Prompt Format **31**

D.1 Main Format 31

D.2 Format for Process Critique Feedback 32

D.3 MBPP Prompt Format 33

E Derivation of Theorem **33**

E.1 Proof of Lemma 33

E.2 Proof of Theorem 34

E.3 Theorem Extension to Batch-Level 35

A PSEUDOCODE OF SPEAR

In this section, we present detailed pseudocode of the SPEAR algorithm. Specifically, Algorithm 1 outlines the client-side training process of SPEAR for a client k, with Algorithm 2 outlining the federated aggregation process at the server with win-trace based FedAvg.

A.1 CLIENT-SIDE SPEAR

Algorithm 1 SPEAR on client k at federated round t

Require: Global weights $\boldsymbol{\theta}^{(t)}$, feedback generator $\mathsf{FB}(\cdot)$, local steps E, loss weights λ_w, λ_l, margin μ, tail tokens τ, max revision attempts N

Initialize $\boldsymbol{\theta}_k^{(t)} \leftarrow \boldsymbol{\theta}^{(t)}$
$\mathcal{W}_k^{(t)} \leftarrow \emptyset$, $\mathcal{L}_k^{(t)} \leftarrow \emptyset$

PHASE 1: INTERACTION

for each $\mathbf{x}_{k,j} \sim \mathcal{D}_k$ sampled for client k **do**
  $\mathbf{c}_{k,j}^{(0)} \leftarrow \mathsf{encode}(\mathbf{x}_{k,j})$
  Generate $\mathbf{y}_{k,j}^{(0)} \sim p_{\boldsymbol{\theta}_k}(\cdot \mid \mathbf{c}_{k,j}^{(0)})$
  if $\mathsf{IsSatisfied}(\mathbf{y}_{k,j}^{(0)}, \mathbf{x}_{k,j})$ **then**
    $\mathcal{W}_k^{(t)} \leftarrow \mathcal{W}_k^{(t)} \cup \{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)})\}$ ▷ Win: satisfactory on first try
  else
    $\mathsf{fixed} \leftarrow \mathsf{false}$; $\mathbf{y}^{\mathrm{prev}} \leftarrow \mathbf{y}_{k,j}^{(0)}$
    for $n = 1$ **to** N **do**
      $\mathbf{f}_{k,j}^{(n)} \leftarrow \mathsf{FB}(\mathbf{x}_{k,j}, \mathbf{y}^{\mathrm{prev}})$ ▷ Get feedback on previous attempt
      $\mathbf{c}_{k,j}^{(n)} \leftarrow \mathbf{c}_{k,j}^{(0)} \,\|\, \mathbf{y}^{\mathrm{prev}} \,\|\, \mathbf{f}_{k,j}^{(n)}$
      Generate $\mathbf{y}_{k,j}^{(n)} \sim p_{\boldsymbol{\theta}_k}(\cdot \mid \mathbf{c}_{k,j}^{(n)})$
      if $\mathsf{IsSatisfied}(\mathbf{y}_{k,j}^{(n)}, \mathbf{x}_{k,j})$ **then**
        $\mathcal{W}_k^{(t)} \leftarrow \mathcal{W}_k^{(t)} \cup \{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(n)})\}$ ▷ Win: self-corrected at attempt n
        $\mathcal{L}_k^{(t)} \leftarrow \mathcal{L}_k^{(t)} \cup \{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)}, \alpha_{k,j}{=}1.0)\}$ ▷ Lose: original was unsatisfactory
        $\mathsf{fixed} \leftarrow \mathsf{true}$; **break**
      end if
      $\mathbf{y}^{\mathrm{prev}} \leftarrow \mathbf{y}_{k,j}^{(n)}$ ▷ Feed latest attempt into next revision
    end for
    if not fixed **then**
      $\mathcal{L}_k^{(t)} \leftarrow \mathcal{L}_k^{(t)} \cup \{(\mathbf{c}_{k,j}^{(0)}, \mathbf{y}_{k,j}^{(0)}, \alpha_{k,j}{=}0.5)\}$ ▷ Lose: all N attempts failed
    end if
  end if
end for

PHASE 2: TRAINING

for $e = 1$ to E **do**
  Sample batches from $\mathcal{W}_k^{(t)}$ and $\mathcal{L}_k^{(t)}$
  Compute ℓ_{win}: negative log-likelihood on win batch
  Compute ℓ_{lose}: confidence-weighted unlikelihood on lose batch, restricted to $\mathcal{T}_\tau(\mathbf{y}^-)$
  $\boldsymbol{\theta}_k^{(t)} \leftarrow \boldsymbol{\theta}_k^{(t)} - \eta \nabla(\lambda_w \ell_{\mathrm{win}} + \lambda_l \ell_{\mathrm{lose}})$
end for
return $\boldsymbol{\theta}_k^{(t)}$

A.2 Server-side Aggregation

Algorithm 2 Federated training with SPEAR

Require: Initial global weights $\boldsymbol{\theta}^{(0)}$
1: **for** $t = 0, 1, 2, \ldots$ **do**
2: Select client subset $\mathcal{K}_t$
3: Broadcast $\boldsymbol{\theta}^{(t)}$ to clients in $\mathcal{K}_t$
4: Each client $k \in \mathcal{K}_t$ returns $\boldsymbol{\theta}_k^{(t)}$ after running SPEAR
5: Calculate number of win traces per client: $w_k^{(t)} = |\mathcal{W}_k^{(t)}|$
6: Aggregate (FedAvg): $\boldsymbol{\theta}^{(t+1)} \leftarrow \sum_{k \in \mathcal{K}_t} \frac{w_k^{(t)} \boldsymbol{\theta}_k^{(t)}}{\sum_{k' \in \mathcal{K}_t} w_{k'}^{(t)}}$
7: **end for**

B Additional Experimental Results

In this section, we consider experimental results in addition to those presented in Sec. 5. Unless stated otherwise, the same settings considered in Sec. 5 are applied.

B.1 Analysis on Assumption of Issues Near Completion End

B.1.1 Ablation over τ

Table 3: Ablation study on the hyperparameter τ for SPEAR on MathMCQA and StrategyQA.

Model	τ	MathMCQA	StrategyQA
Llama3.2-3B	8	**55.89**±0.65	66.47±2.93
	64	55.13±0.77	66.42±1.57
	128	54.43±1.08	**67.39**±2.38
Qwen2.5-1.5B	8	**57.27**±0.41	**64.39**±2.48
	64	54.61±4.17	64.00±1.49
	128	56.85±1.18	62.20±4.63

We consider if the assumption that unsatisfactory tokens concentrate near the end of the completion is valid by conducting an ablation study on τ. We consider only MathMCQA and StrategyQA as HellaSwag and ARC-Challenge do not require the lengthy chain-of-thought (CoT) reasoning chains that MathMCQA and StrategyQA do. From the results in Table 3, we can see that the motivation of targeting the tail of the sequence holds. With the exception of StrategyQA on Llama3.2-3B, we see that targeting 8 tokens leads to better performance in comparison to 64 and 128 tokens. In terms of why StrategyQA with Llama3.2-3B performs the best, it could be that the model is producing extremely lengthy outputs, thereby requiring more tokens to be targeted.

B.1.2 Initial Error Localization

Table 4: Location of the first error across incorrect responses on StrategyQA and MathMCQA.

Error Quartile Location	StrategyQA	MathMCQA
0–25%	23.6%	19.6%
25–50%	19.9%	23.1%
50–75%	26.1%	24.1%
75–100%	**30.4%**	**33.2%**

Outside of the ablation study over τ, we additionally consider where unsatisfactory steps initially appear within lose trace completions. For this, we take Qwen2.5-7B and instruct an LLM Judge (Qwen2.5-14B-Instruct) to find the first occurrence of an error within 200 incorrect answers across MathMCQA and StrategyQA. Based on the results from Table 4, we can note that the majority of issues initially occur in the bottom half of the completion, with the error percentage most prevalent in the bottom 25%. This indicates a consistent tail skew, further validating our choice to target the last τ tokens.

B.1.3 Qualitative Example on MathMCQA

Lastly, we consider a qualitative example of an incorrect output produced from Qwen2.5-7B on MathMCQA. In the example, we see for the initial model response that the error occurs near the end, where it treats the two isolated values as the entire solution set, ignoring the $+2\pi t$ periodic family entirely. Moreover, we note that the model has corrected not only its answer, but also the reasoning steps required to get to the correct solution, indicating that the model did not merely flip the label and get lucky in its selection.

Problem

Solve

$$\cos(4x) = -\frac{1}{2},$$

given that all solutions can be expressed as

$$x = \frac{(kn \pm 1)\pi}{6}, \qquad n \in \mathbb{Z},$$

and determine the positive value of k.

Answer choices: A. 2, B. 3, C. 6, D. 4

Reference answer: B ($k = 3$)

Initial model response — Incorrect

We know that

$$\cos(4x) = -\frac{1}{2}$$

when

$$4x = \frac{2\pi}{3} \quad \text{or} \quad 4x = \frac{4\pi}{3}.$$

Solving for x, we obtain

$$x = \frac{\pi}{6} \quad \text{or} \quad x = \frac{\pi}{3}.$$

The solutions have a common factor of $\pi/6$, so they can be written in the form

$$\frac{(kn \pm 1)\pi}{6}.$$

Comparing with the required form gives $k = 2$.
Answer: A

Error in the initial reasoning. The initial response correctly finds the two principal solutions, but it incorrectly treats them as the complete solution set. In particular, it omits the $2\pi t$ periodicity of cosine.

Model response after feedback — Corrected

If

$$\cos(4x) = -\frac{1}{2},$$

then, for some integer t,

$$4x = \frac{2\pi}{3} + 2\pi t = \frac{2(3t+1)\pi}{3},$$

or

$$4x = \frac{4\pi}{3} + 2\pi t = \frac{2(3t+2)\pi}{3}.$$

Dividing by 4 gives

$$x = \frac{(3t+1)\pi}{6} \qquad \text{or} \qquad x = \frac{(3t+2)\pi}{6}.$$

Thus, $k = 3$.
Answer: B

Why the corrected reasoning is valid. The revised response explicitly restores the missing periodic terms and propagates them through the derivation. The second solution family can be reindexed as

$$3t + 2 = 3(t+1) - 1.$$

Therefore, the two families can be written together as

$$x = \frac{(3n \pm 1)\pi}{6}, \qquad n \in \mathbb{Z},$$

which gives $k = 3$.

B.2 High Number of Clients with High Heterogeneity

Table 5: Performance Comparison of SPEAR vs. baselines across benchmark datasets on Llama3.2-3B and Qwen2.5-1.5B with 100 total clients and heterogeneity of `Dirichlet(0.3)`.

Model	Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Llama3.2-3B	Feedback SFT	65.67±0.98	83.07±1.63	41.58±0.36	51.53±0.01	60.46
	GRPO	66.68±0.64	66.84±2.03	37.01±8.73	49.34±0.87	54.97
	OPSD	25.10±0.63	24.57±1.88	0.00±0.00	0.00±0.00	12.42
	RLTF-SD	61.86±2.56	42.19±16.46	40.29±0.81	25.76±13.26	42.53
	SPEAR (Ours)	68.86±0.03	85.45±1.20	48.87±1.81	54.01±1.74	**64.30**
Qwen2.5-1.5B	Feedback SFT	73.76±1.15	76.15±1.14	44.90±0.16	48.47±0.12	60.82
	GRPO	76.45±0.60	78.32±1.06	48.95±1.93	50.22±1.75	63.48
	OPSD	24.75±0.27	26.49±0.04	0.00±0.00	0.00±0.00	12.81
	RLTF-SD	66.34±8.75	71.41±7.88	46.59±5.89	50.00±1.53	58.59
	SPEAR (Ours)	70.77±3.53	79.34±1.79	47.24±5.86	62.53±2.84	**64.97**

We consider how SPEAR performs in comparison to the evaluated baselines with two additional dimensions common in FL settings: a large number of clients within the network, and high data heterogeneity between the clients. To model this, we consider a setting with 100 clients and data splitting across clients via a Dirichlet parameter of 0.3 (Green & Richardson, 2001; Reguieg et al., 2023). For this experiment, 3 clients are aggregated every FL round.

From the results in Table 5, we note the performance advantages exhibited by SPEAR in Table 1 persist even when considering the additional dimensions of higher client cardinality and high heterogeneity in data between local clients. We can also see this consistency in performance persists across models, with it being the best performing method across both Llama3.2-3B and Qwen2.5-1.5B.

SPEAR's consistency is maintained on all datasets, but the same can't be said for some baselines, such as Feedback SFT, RLTF-SD, or GRPO (e.g., 25.76 on StrategyQA for RLTF-SD). Overall, this indicates that SPEAR is robust to changes in the number of clients and data heterogeneity considerations in comparison to the baselines.

B.3 Fine-tuning on Only Last H Layers

Table 6: Performance Comparison of SPEAR vs. baselines across benchmark datasets on Qwen2.5-1.5B, fine-tuned on the last $H = 12$ layers.

Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Feedback SFT ($H = 12$)	63.82±3.11	71.40±1.34	47.20±2.64	50.00±1.53	58.11
GRPO ($H = 12$)	71.50±0.09	70.81±1.20	41.65±2.73	47.74±0.73	57.93
OPSD ($H = 12$)	24.32±0.01	25.41±0.33	0.00±0.00	0.00±0.00	12.43
RLTF-SD ($H = 12$)	71.20±0.04	69.62±2.88	38.81±3.70	48.33±0.15	56.99
SPEAR ($H = 12$)	69.75±0.38	75.73±1.05	46.08±1.22	57.06±7.57	**62.16**

We now consider a scenario more constrained than those presented in Sec. 5, where only the last H layers of the model are fine-tuned with LoRA. We consider $H = 12$ on Qwen2.5-1.5B. The results are presented in Table 6.

Firstly, we can note that SPEAR remains the highest-performing method, with the highest average score exhibited in comparison to all baselines. We note that unlike the baselines, SPEAR is always consistently well performing across all datasets, in comparison to GRPO (e.g., only 41.65% on MathMCQA) or Feedback SFT (e.g., only 63.82% on ARC-Challenge). Overall, this means even in a scenario where we only fine-tune on a limited number of layers, SPEAR is still the superior algorithm in comparison to the baselines.

B.4 Attempting Revision Only Once

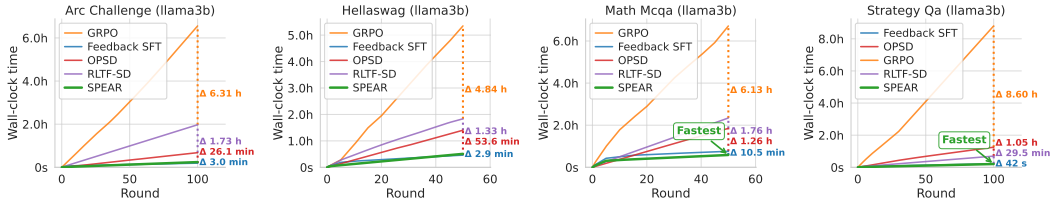
Table 7: Performance Comparison of SPEAR vs. baselines with $N = 1$ revision attempt.

Model	Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Llama3.2-3B	Feedback SFT	67.95±1.45	84.60±0.98	48.60±9.89	49.49±1.76	62.66
	GRPO	65.70±3.48	75.99±1.53	40.20±14.89	49.34±1.51	57.81
	OPSD	24.69±1.55	25.03±0.64	0.00±0.00	0.00±0.00	12.43
	RLTF-SD	63.28±3.58	60.05±12.30	44.70±7.92	51.58±0.08	54.90
	SPEAR ($N = 1$)	69.91±0.20	86.86±0.18	53.33±1.87	66.28±1.28	**69.10**
Qwen2.5-1.5B	Feedback SFT	73.89±3.38	76.14±3.61	54.93±2.04	50.51±1.76	63.87
	GRPO	76.30±0.17	79.00±1.65	50.87±3.86	51.38±6.44	64.39
	OPSD	25.03±1.23	24.92±0.69	0.00±0.00	0.00±0.00	12.49
	RLTF-SD	75.03±0.40	60.65±20.78	44.70±7.41	54.29±10.08	58.67
	SPEAR ($N = 1$)	76.68±0.56	82.44±0.75	56.51±1.57	56.72±6.44	**68.09**

In Sec. 5, since we consider $N = 2$ attempts at revision, we also seek to see if $N = 1$ still maintains superior performance in comparison to the baselines. We consider both the performance of SPEAR with $N = 1$ and the run time in comparison to the baselines. Accuracy results are presented in Table 7.

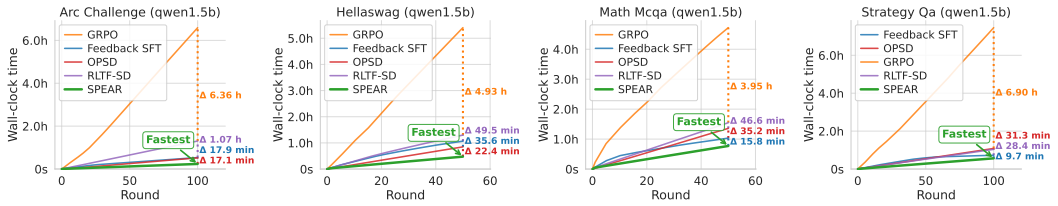
Firstly, from Table 7, we note that even with $N = 1$, the performance of SPEAR remains superior to the evaluated baselines on all settings. Likewise, the performance across all datasets and models remains high, meaning that SPEAR is robust to settings where the number of revision attempts is set low.

We also note the training time of doing SPEAR for $N = 1$ revisions in comparison to the baselines in Fig. 3. We can see similarly to Fig. 2 in Sec. 5, SPEAR is in a majority of scenarios the fastest



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

(a) Llama3.2-3B



(e) ARC-Challenge (f) HellaSwag (g) MathMCQA (h) StrategyQA

(b) Qwen2.5-1.5B

Figure 3: Cumulative training time of each algorithm on (a) Llama3.2-3B and (b) Qwen2.5-1.5B models for $N = 1$ maximum revision on SPEAR.

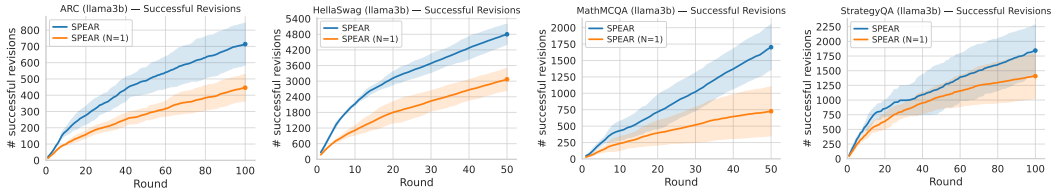
algorithm. When it isn't (ARC-Challenge and HellaSwag on Llama3.2-3B, with Feedback SFT being the best), it is very close to the fastest, with a difference of only a couple of minutes. Moreover, as seen in Table 7, the slightly slower training time is made up for with performance gains in accuracy. When further comparing with Fig. 2 in Sec. 5, we can see the margin of cumulative time difference between the slower baselines increases, which intuitively makes sense as when the number of generations is lower, less overall tokens are generated. Overall, this means that in settings where training must be conducted in a quick manner, SPEAR is able to accommodate only one attempt at revision without significant change to the overall performance after multiple FL rounds.

B.5 Additional Comparisons of $N = 1$ vs. $N = 2$

Based on the results in Appendix B.4, we additionally seek to observe the following characteristics of N: the cumulative number of successful revisions (Fig. 4), the number of cumulative samples needing revision (Fig. 5), and the percentage of samples needing revision each round over the course of FL training (Fig. 6).

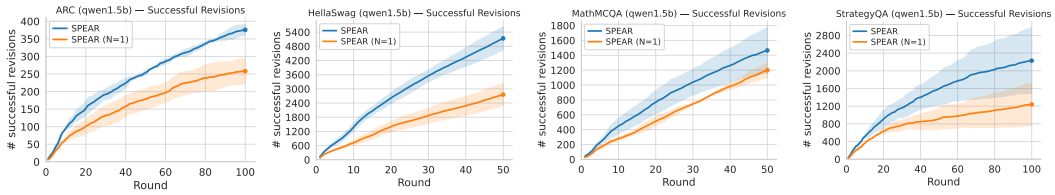
From Fig. 4, we can first observe that SPEAR with $N = 2$ achieves a higher level of total success in terms of number of samples it is able to correct in comparison to only attempting revision one time. This is to be expected as allowing the model to make multiple attempts at generation allows it to explore differing solutions to come to a satisfactory answer. However, the total number of samples overall needing revision over the course of training does not significantly change, as indicated in Fig. 5. This observation further matches with Appendix B.4, where the performance of SPEAR $(N = 1)$ remains higher than the baselines and is comparable to the results of SPEAR in Table 1. This means that while attempting more revisions does increase the number of satisfactory traces, it does not necessarily mean the total number of samples that will need to be revised changes significantly in the context of the entire data.

Secondly, we can note that in Fig. 6 that over time, the percentage of samples per-round needing revision decreases, indicating that the SPEAR algorithm is effective at learning from the win-lose trace dual loss function. Additionally, in most cases, the difference between $N = 1$ and $N = 2$ revision attempts does not significantly differ from one another. Overall, this further reinforces the results from Appendix B.4, where conducting revision only once with SPEAR is still an effective means of fine-tuning in an online manner with incomplete feedback.



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

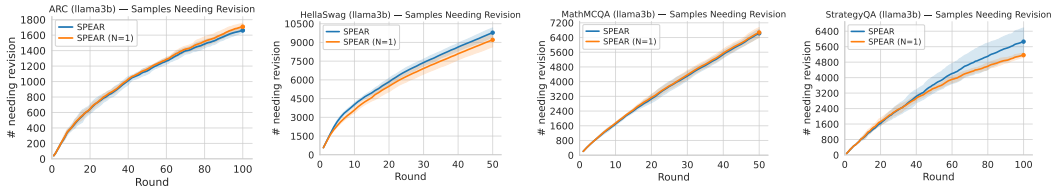
(a) Llama3.2-3B



(e) ARC-Challenge (f) HellaSwag (g) MathMCQA (h) StrategyQA

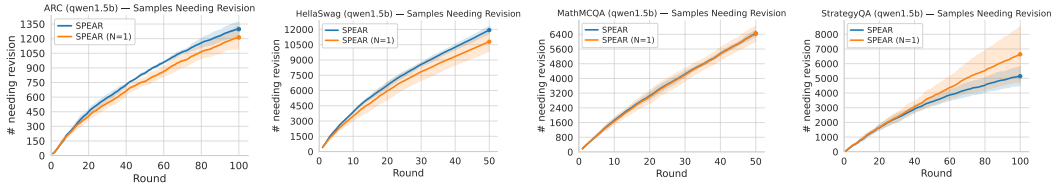
(b) Qwen2.5-1.5B

Figure 4: Cumulative successful revisions of SPEAR with $N = 1$ and $N = 2$ (default) on (a) Llama3.2-3B and (b) Qwen2.5-1.5B models.



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

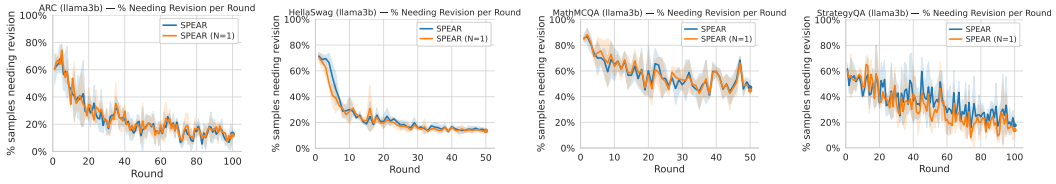
(a) Llama3.2-3B



(e) ARC-Challenge (f) HellaSwag (g) MathMCQA (h) StrategyQA

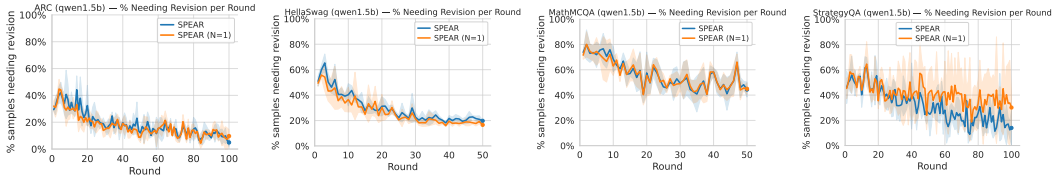
(b) Qwen2.5-1.5B

Figure 5: Cumulative number of samples needing revision of SPEAR with $N = 1$ and $N = 2$ (default) on (a) Llama3.2-3B and (b) Qwen2.5-1.5B models.



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

(a) Llama3.2-3B



(e) ARC-Challenge (f) HellaSwag (g) MathMCQA (h) StrategyQA

(b) Qwen2.5-1.5B

Figure 6: Percentage of samples per-round needing revision of SPEAR with $N = 1$ and $N = 2$ (default) on (a) Llama3.2-3B and (b) Qwen2.5-1.5B models.

B.6 Ablation on λ_l

Table 8: Ablation study on λ_l for SPEAR across benchmark datasets on Llama3.2-3B and Qwen2.5-1.5B.

Model	λ_l	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA
Llama3.2-3B	0.1	69.17±0.85	**86.72**±0.67	**56.70**±0.78	65.84±0.78
	0.3	68.83±0.92	86.46±1.10	53.83±0.83	**67.69**±1.66
	0.5	69.48±1.29	86.81±0.09	52.93±0.21	66.08±2.61
	0.7	**69.57**±1.11	86.17±0.51	52.35±1.99	66.72±1.14
Qwen2.5-1.5B	0.1	76.93±1.41	80.65±0.79	**59.07**±1.16	**66.08**±3.20
	0.3	75.57±3.42	**82.03**±1.22	56.31±0.54	62.93±1.55
	0.5	**78.01**±0.46	79.11±2.50	55.87±0.84	59.58±2.13
	0.7	76.51±1.01	82.02±0.42	51.10±4.04	64.29±2.08

As outlined in Sec. 3.2.2, the confidence-gated and tail-targeted unlikelihood loss plays an important role in the overall formulation of SPEAR. For this reason, we present an ablation study on how the weighting given to this loss (λ_l) affects the overall performance of the proposed algorithm.

From the results presented in Table 8, we first note that while there is no set value that is best across all datasets, setting a lower value for λ_l often results in better performance. For example, with MathMCQA, the difference between 0.1 and 0.5 weighting is around 3-4%, with around a 6% difference from 0.1 to 0.5 on StrategyQA for Qwen2.5-1.5B. Overall, across all datasets, the optimal value seems to range from 0.1 to 0.3 in weight. We believe this to occur due to the fact that weighting the lose traces higher leads to less emphasis to be put on the win traces. Because accuracy is primarily driven by MLE on the win traces, the SFT loss should remain the dominant training signal. Our auxiliary confidence-gated, tail-targeted unlikelihood loss improves results by steering the model away from some completions, but if weighted too heavily, it can pull optimization away from the main objective and hurt accuracy. However, we can see that for ARC-Challenge, the converse is true–higher λ_l results in better performance. For this factor, we believe this is because the outputs we prompt ARC-Challenge to produce are inherently shorter than datasets such as MathMCQA or

StrategyQA. Overall, while the results indicate that setting λ_l lower than λ_w is more appropriate, it is also important to consider the length of the generated outputs.

B.7 SPEAR ON DIFFERING FL METHODS

Table 9: Usage of differing common FL algorithms on SPEAR across benchmark datasets on Llama3.2-3B and Qwen2.5-1.5B.

Model	FL Method	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA
Llama3.2-3B	FedAvg	69.58±0.47	86.99±0.48	56.25±0.05	65.36±0.58
	FedProx	69.58±0.47	87.13±0.26	54.06±0.06	65.36±0.58
	FedAdam	68.09±0.17	85.20±0.27	53.74±1.40	56.40±6.77
	FedYogi	70.22±0.43	84.88±0.38	53.72±1.52	67.98±0.58
Qwen2.5-1.5B	FedAvg	76.92±1.41	80.55±0.77	59.60±0.70	64.48±1.60
	FedProx	76.92±1.41	80.67±0.50	57.07±0.77	65.87±2.98
	FedAdam	76.15±1.49	76.34±1.33	55.86±0.98	65.57±0.51
	FedYogi	67.58±6.40	72.12±2.40	54.63±0.75	64.56±0.36

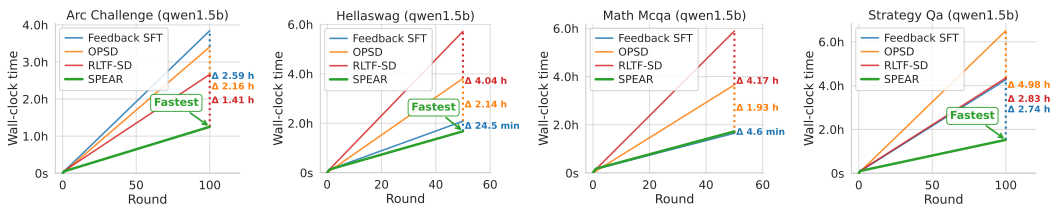
Despite FedAvg (McMahan et al., 2017) being the dominant default adopted in most FL works, this section seeks to see if SPEAR still maintains good performance with differing, but common FL protocols. For this, we consider three well-established alternatives to FedAvg: FedProx (Li et al., 2020b), FedAdam (Reddi et al., 2020; Baumgart et al., 2024), and FedYogi (Reddi et al., 2020; Baumgart et al., 2024). For FedProx, the proximal term is set to 10^{-3}, with β_1 and β_2 set to 0.9 and 0.99 for FedAdam and FedYogi. In addition, the adaptivity hyperparameter for FedAdam and FedYogi is set to 0.005. Results are presented in Table 9.

From Table 9, we note that SPEAR maintains relatively good and consistent performance across all evaluated FL methods (FedAvg, FedProx, FedAdam, and FedYogi). There are, however, rare cases where there is some performance difference, e.g., HellaSwag on Qwen2.5-1.5B, where the difference between FedAvg and FedYogi is around 8%. However, we can still see accuracy is always above 72%, meaning no matter the method chosen, high accuracy is still maintained. These results indicate that SPEAR is robust to differing FL protocols across differing model types, sizes, and datasets.

B.8 CENTRALIZED SETTING

Table 10: Performance Comparison of SPEAR vs. baselines in a centralized setting on Qwen2.5-1.5B.

Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Feedback SFT	75.00±1.11	85.06±0.10	58.68±0.32	66.81±0.29	71.39
OPSD	22.69±0.16	24.71±0.03	0.00±0.00	0.00±0.00	11.85
RLTF-SD	75.85±2.05	75.48±4.16	43.36±10.08	63.90±1.46	64.65
SPEAR (Ours)	78.67±0.68	86.68±0.09	60.73±0.07	67.47±0.51	**73.39**



(a) ARC-Challenge (b) HellaSwag (c) MathMCQA (d) StrategyQA

Figure 7: Cumulative training time of each algorithm on Qwen-1.5B model in a centralized setting.

We consider whether SPEAR can also be extended to the centralized setting outside of FL considerations (i.e., equivalent to having one client), and whether it maintains superior performance in both accuracy and wall-clock time in comparison to the baselines. Results are found in Table 10 and Fig. 7.

From the results in Table 10, we note the performance advantages presented by SPEAR in Sec. 5 persist even in a centralized setting. Specifically, we can note that SPEAR still maintains superior performance on all datasets in comparison to all baselines on Qwen2.5-1.5B. We see a similar result in Fig. 7, where SPEAR exhibits faster training times in comparison to all baselines in 3 out of 4 datasets. In the case of MathMCQA, the difference with Feedback SFT is only a couple of minutes, and this tradeoff is marked by 2% improvement in accuracy. These results indicate that SPEAR is an effective method of online fine-tuning of LLMs in both an FL and centralized environment.

B.9 Larger Model Size: Qwen2.5-7B

Table 11: Performance Comparison of SPEAR vs. baselines across benchmark datasets on Qwen2.5-7B.

Algorithm	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA	Avg.
Feedback SFT	89.38±0.47	91.93±0.07	54.90±2.74	73.87±0.51	77.52
RLTF-SD	89.55±0.30	91.96±0.11	–	27.44±21.76	69.65
SPEAR (Ours)	89.72±0.21	93.12±0.09	59.75±0.17	69.65±2.55	**78.06**

Here, we consider whether SPEAR's advantages persist on models of larger size by comparing against the baselines on Qwen2.5-7B, with the results shown in Table 11. For ARC-Challenge, HellaSwag, and MathMCQA, we use a maximum and minimum learning rates of 5×10^{-5} / 1×10^{-5}. For StrategyQA, it is 1×10^{-4} / 5×10^{-5}.

Based on the results in Table 11, we note that the performance gains from SPEAR persist on larger model sizes. RLTF-SD on MathMCQA is omitted as well as GRPO due to memory constraints. This further highlights SPEAR's memory advantages in comparison to RL group generation based methods.

B.10 Comparison of SPEAR against DPO-based method

Table 12: Performance Comparison of SPEAR against Online Feedback DPO across benchmark datasets.

Algorithm	ARC-Challenge	StrategyQA	MathMCQA	HellaSwag	Avg.
Online Feedback DPO	69.62	53.81	31.25	60.35	53.76
SPEAR (Ours)	76.93	66.08	59.07	80.65	**70.68**

We consider whether SPEAR's MLE and unlikelihood objective can outperform preference-based RLHF learning. For this, we have conducted experiments to compare against an online variant of DPO (Rafailov et al., 2023) that trains on satisfactory-unsatisfactory pairs found during training on Qwen2.5-1.5B. Results are presented in Table 12.

We can see from the results that SPEAR maintains superior performance in comparison to Online Feedback DPO on average and across each individual dataset. This indicates that the proposed method with its dual MLE and unlikelihood objective outperforms a directly matched online preference-optimization baseline.

B.11 Experiment Over Newer Model Family

In contrast to the Qwen2.5 and Llama3.2 model families, here we consider Qwen3.5 (Qwen Team, 2026), which is a more recent family of models. Specifically, we consider SPEAR against Feedback-

Table 13: Performance Comparison of SPEAR vs. baselines across benchmark datasets on Qwen3.5-2B.

Algorithm	ARC-Challenge	StrategyQA	MathMCQA	HellaSwag	Avg.
Online Feedback DPO	80.56	39.80	31.34	60.44	53.04
Feedback SFT	82.59	48.47	52.60	85.21	67.22
SPEAR (Ours)	82.79	59.83	54.38	85.92	**70.73**

SFT and Online Feedback DPO on Qwen3.5-2B (results in Table 13). For ARC-Challenge, HellaSwag, MathMCQA, and StrategyQA, the maximum and minimum learning rates are 1×10^{-5} / 5×10^{-6}, 5×10^{-5} / 1×10^{-5}, 5×10^{-5} / 1×10^{-5}, and 1×10^{-5} / 5×10^{-6} respectively.

From Table 13, we note that SPEAR's performance gains observed in Sec. 5 persist on Qwen3.5-2B, with SPEAR outperforming the baselines on all four evaluated datasets. Overall, the results indicate SPEAR's advantages persist across more recent architectures in addition to those evaluated in Sec. 5.

B.12 SPEAR under Weaker Feedback

In this section, we consider whether SPEAR can still meaningfully learn under comparatively weaker feedback than those utilized in Sec. 5. Specifically, we consider two types of weaker feedback compared to the original partial feedback: noisy partial feedback and process critique. For **Noisy Partial Feedback**, we replace 20% of the original partial feedback content with random and irrelevant content. With **Process Critique**, the same generic advice (e.g., "The answer is unsatisfactory. Find the first error and continue from there.") not specific to any one problem is given. The specific feedback given for Process Critique for each dataset is found in Appendix D.2.

B.12.1 Correction Percentage before Training under Weaker Feedback

Table 14: Correction percentage of Qwen2.5-1.5B and Llama3.2-3B before training under different feedback types.

Model	Feedback Type	ARC-Challenge	HellaSwag	MathMCQA	StrategyQA
Qwen2.5-1.5B	Process Critique	16.1%	16.6%	15.3%	35.6%
	Noisy Partial Feedback	18.3%	21.5%	12.2%	45.7%
	Partial Feedback (Original)	28.2%	19.5%	12.6%	46.3%
Llama3.2-3B	Process Critique	22.9%	22.4%	10.4%	35.9%
	Noisy Partial Feedback	28.2%	31.7%	11.3%	38.3%
	Partial Feedback (Original)	34.6%	31.7%	12.8%	45.6%

In Table 14, we consider whether the foundation model can meaningfully correct itself under the weaker feedback types before training. While we note that the model under Process Critique and Noisy Partial Feedback in general corrects itself less often than the original partial feedback, it is still able to meaningfully correct a non-trivial number of samples. Moreover, for some settings (e.g., HellaSwag for Llama3.2-3B / MathMCQA for Qwen2.5-1.5B), the weaker feedback settings outperform or equal the original partial feedback in correction percentage. We demonstrate in Appendix B.12.2 that SPEAR can learn from all three feedback types.

B.12.2 Accuracy on Weaker Feedback

Next, we consider whether performance on comparatively weaker feedback still enables comparable performance to the original partial feedback. In addition to Qwen2.5-1.5B and Llama3.2-3B (Table 15), we also consider the Qwen3.5-2B model (Table 16).

From the results in Tables 15 and 16, SPEAR remains effective under both Process Critique and Noisy Partial Feedback across all model families and datasets evaluated. Therefore, the model's ability to

Table 15: SPEAR under noisy and generic process critique feedback on Qwen2.5-1.5B and Llama3.2-3B.

Model	Feedback Type	MathMCQA	HellaSwag	StrategyQA	ARC-Ch.	Avg.
Qwen2.5-1.5B	Process Critique	49.44	80.28	64.92	76.79	67.86
	Noisy Partial Feedback	56.56	82.46	65.07	78.16	70.56
	Partial Feedback (Orig.)	59.07	80.65	66.08	76.93	70.68
Llama3.2-3B	Process Critique	56.38	87.57	62.30	76.19	70.61
	Noisy Partial Feedback	55.02	87.77	62.30	70.14	68.81
	Partial Feedback (Orig.)	56.70	86.72	65.84	69.17	69.61

Table 16: SPEAR under noisy and generic process critique feedback on Qwen3.5-2B.

Feedback Type	MathMCQA	HellaSwag	StrategyQA	ARC-Challenge	Avg.
Process Critique	40.62	86.19	50.95	82.59	65.09
Noisy Partial Feedback	54.72	85.56	51.52	83.20	68.75
Partial Feedback (Original)	54.38	85.92	59.83	82.79	70.73

learn **under weaker generic and noisy feedback utilizing SPEAR's training methodology is not isolated to a specific model family or dataset**. Moreover, by observing Table 1 in Sec. 5, both Process Critique and Noisy Partial Feedback-based SPEAR **outperform the baselines.** Overall, these results indicate SPEAR remains an effective methodology when considering more practical scenarios where high-quality feedback is not always guaranteed.

B.13 Open-Ended Scenario with Weak Feedback

Table 17: MBPP results on Qwen2.5-1.5B.

	SPEAR	Feedback SFT	Online Feedback DPO	RLTF-SD	GRPO
Pass Rate (%)	40.6	37.2	24.5	44.0	40.8

We consider whether SPEAR can remain effective under two difficult conditions simultaneously: (i) an open-ended scenario where multiple differing completions can be satisfactory and (ii) under weak feedback. For this, we consider the Mostly Basic Python Problems (MBPP) (Austin et al., 2021) dataset, which accepts multiple differing solutions, given that they pass the test cases for each sample. In addition, we utilize the Process Critique-based feedback from Appendix B.12. The exact formatting prompts used can be found in Appendix D.3.

Given the results from Table 17, SPEAR achieves a 40.6% pass rate with Qwen2.5-1.5B, **trained using only general process critique-based feedback.** It **outperforms both online feedback baselines** (Feedback SFT and Online Feedback DPO) while remaining nearly identical in performance to GRPO. Importantly, although RLTF-SD exhibits a moderately higher pass rate of 44%, **GRPO and RLTF-SD require privileged ground-truth information during training, incur substantially greater training cost, and do not operate in the same online feedback setting**, in contrast with SPEAR. These results therefore demonstrate that SPEAR can effectively extend to open-ended tasks under simple feedback while retaining its online-learning advantages. Importantly, this experiment also illustrates why SPEAR's binary signal is better interpreted as interaction-level satisfaction rather than supervision from a unique ground-truth completion. MBPP admits multiple distinct programs as successful whenever they satisfy the task's test cases; the binary signal therefore indicates whether the produced response is acceptable without specifying which response the model should generate.

C Detailed Hyperparameter Setup

In this section, we outline the hyperparameter settings utilized in the running of experiments within the manuscript (in addition to those presented in Sec. 5) for each algorithm.

C.1 Hyperparameter Settings (SPEAR)

Table 18: Training and federated learning hyperparameters for the Qwen2.5-1.5B SPEAR runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1024	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Generation temperature	0.8	0.8	0.8	0.8
Lambda win weight λ_w	1.0	1.0	1.0	1.0
Lambda loss weight λ_l	0.1	0.1	0.1	0.1
Unlikelihood threshold μ	0.3	0.3	0.3	0.3
τ	32	0	24	0
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	4	16	16	16
Grad accumulation steps	4	1	1	1
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}

Table 19: Training and federated learning hyperparameters for the Llama-3.2-3B SPEAR runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1024	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Generation temperature	0.8	0.8	0.8	0.8
Lambda win weight λ_w	1.0	1.0	1.0	1.0
Lambda loss weight λ_l	0.1	0.1	0.1	0.1
Unlikelihood threshold μ	0.3	0.3	0.3	0.3
τ	32	0	24	0
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	4	8	16	8
Grad accumulation steps	4	2	1	2
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}

C.2 HYPERPARAMETER SETTINGS OF BASELINES (GRPO / OPSD / RLTF-SD)

Here, the hyperparameter settings of all baselines evaluated (GRPO / OPSD / RLTF-SD) against SPEAR are presented. For StrategyQA, note the max sequence length is increased for the baselines as the entirety of the “facts" column is inputted as feedback for OPSD; for GRPO, it’s included as the regular input, thereby requiring a larger sequence of tokens. For SPEAR, only partial portions of this column are included when attempting revision, allowing a smaller sequence length.

Table 20: Training, federated learning, and GRPO hyperparameters for the Qwen2.5-1.5B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	2	4	4	4
Grad accumulation steps	8	4	4	4
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}
GRPO β_{KL}	0.02	0.02	0.1	0.02
GRPO clip range	0.2	0.2	0.2	0.2
GRPO num. generations	2	4	2	4

Table 21: Training, federated learning, and GRPO hyperparameters for the Llama-3.2-3B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	2	4	4	4
Grad accumulation steps	8	4	4	4
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}
GRPO β_{KL}	0.02	0.02	0.1	0.02
GRPO clip range	0.2	0.2	0.2	0.2
GRPO num. generations	2	4	2	4

Table 22: Training and federated learning hyperparameters for OPSD on the Qwen2.5-1.5B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	4	16	4	8
Grad accumulation steps	4	1	4	2
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}

Table 23: Training and federated learning hyperparameters for OPSD on the Llama-3.2-3B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	2	8	4	8
Grad accumulation steps	8	2	4	2
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}

Table 24: Training, federated learning, and RLTF-SD hyperparameters for the Qwen2.5-1.5B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	2	4	4	4
Grad accumulation steps	8	4	4	4
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}
RLTF-SD γ	1.0	1.0	1.0	1.0
RLTF-SD SD Coefficient	0.1	0.1	0.1	0.1
RLTF-SD RL Coefficient	1.0	1.0	1.0	1.0
RLTF-SD Early Termination	True	True	True	True

Table 25: Training, federated learning, and RLTF-SD hyperparameters for the Llama-3.2-3B runs.

Hyperparameter	**Math MCQA**	**ARC-Challenge**	**StrategyQA**	**HellaSwag**
Max sequence length	2048	2048	1400	2048
LoRA rank	16	16	16	16
LoRA alpha	32	32	32	32
LoRA dropout	0.1	0.1	0.1	0.1
Number of clients	50	50	50	50
Dirichlet alpha	1.0	1.0	1.0	1.0
Number of FL rounds	50	100	100	50
Clients per round aggregated	3	3	5	5
Local training steps per round	5	5	5	10
Ratio of rounds to warmup	0.1	0.1	0.1	0.1
Learning rate	5.05×10^{-5}	5.0×10^{-5}	1.0×10^{-4}	5.0×10^{-5}
Batch size	2	4	4	4
Grad accumulation steps	8	4	4	4
Linear warmup ratio	0.1	0.1	0.1	0.1
Minimum LR for cosine decay	1.0×10^{-5}	1.0×10^{-5}	5.0×10^{-5}	1.0×10^{-5}
RLTF-SD γ	1.0	1.0	1.0	1.0
RLTF-SD SD Coefficient	0.1	0.1	0.1	0.1
RLTF-SD RL Coefficient	1.0	1.0	1.0	1.0
RLTF-SD Early Termination	True	True	True	True

D PROMPT FORMAT

D.1 MAIN FORMAT

Here, the text prompt templates (both initial input and feedback loop) for each dataset are provided below. Parts outlined with {} in blue indicate data fields to be inputted into the text prompts.

Prompt Template — MathMCQA

```
Solve the math problem step by step, then choose the correct answer.
Show your reasoning, then end with exactly: "Answer: X"
where X is the letter A, B, C, or D.

[PROBLEM]
{problem}

[OPTIONS]
{options}
```

+ Feedback (appended on unsatisfactory response):

```
Incorrect. Your answer '{predicted_letter}' is wrong.

Hint -- consider these opening steps of the solution:
{hint_lines}

Reason step by step, then end your response with exactly: "Answer:
X" where X is A, B, C, or D. DO NOT repeat the letter you used
previously.
```

Prompt Template — HellaSwag

```
Choose the best continuation of the context.
Respond with ONLY the letter (A, B, C, or D).

[CONTEXT]
{context}

[OPTIONS]
{options}

Answer:
```

+ Feedback (appended on unsatisfactory response):

```
Incorrect. '{predicted_letter}' is wrong.

Hint: Consider an answer that starts with "{hint_words}..."
Choose the most natural continuation.

Answer with exactly one letter: A, B, C, or D. DO NOT choose the
answer you used previously.
```

Prompt Template — ARC-Challenge

```
Answer the multiple-choice question by choosing the best option.
Respond with ONLY the letter (A, B, C, or D).

[QUESTION]
{question}

[OPTIONS]
{options}

Answer:
```

+ Feedback (appended on unsatisfactory response):

```
Incorrect. '{predicted_letter}' is wrong.

Hint: Consider an answer that starts with "{hint_words}..."
Consider the underlying scientific principle.
```

```
Answer with exactly one letter:  A, B, C, or D. DO NOT choose the
answer you used previously.
```

Prompt Template — StrategyQA

```
Answer the following yes/no question by reasoning step by step.
Think through the question carefully, then end your response with
exactly:  "Answer:  yes" or "Answer:  no" on the final line.

[QUESTION]
{question}
```

+ Feedback (appended on unsatisfactory response):

```
Incorrect.  '{predicted_answer}' is not the right answer.

Hint -- consider these relevant facts:
  - {facts}
  ⋮

Reason through the question step by step, then end your response with
exactly:  "Answer:  yes" or "Answer:  no".  DO NOT repeat the answer
you gave previously.
```

D.2 Format for Process Critique Feedback

Below, we outline the specifics of the process critique-based feedback outlined in Appendix B.12 for each dataset. Importantly, outside of the feedback field, the other components are the same as those presented in Appendix D.1.

Process Critique Feedback — ARC-Challenge

```
The prior answer is unsatisfactory.  Check what quantity or scientific
relationship the question asks about, eliminate options that violate
that relationship, and then revise.

I am not providing the correct option.  Do not use the previous
answer.
```

Process Critique Feedback — HellaSwag

```
The prior continuation is unsatisfactory.  Check chronology, actors,
physical plausibility, and whether the continuation is stylistically
natural.

I am not providing the correct option.  Do not use the previous
answer.
```

Process Critique Feedback — MathMCQA

```
The prior solution is unsatisfactory.  Locate the earliest step whose
arithmetic or algebra is not independently justified, recompute from
there, and compare the result with every option.

I am not providing the correct option.  Do not use the previous
answer.
```

Process Critique Feedback — StrategyQA

```
The prior answer is unsatisfactory.  Separate the question into
factual claims, verify whether each claim supports or contradicts
the conclusion, and then reconsider.

I am not providing the correct answer.
```

D.3 MBPP Prompt Format

Feedback Template — MBPP

General feedback

```
The implementation executed but failed at least one hidden functional
test.  Re-read the task, check boundary cases and return-value
semantics, and revise the algorithm.

[REQUIRED INTERFACE] Your revised program must define this callable
interface exactly.  The function name and parameter list are part
of the task specification; the function body remains for you to
implement.

def example_function(n):
    ...

The hidden tests, test inputs, expected outputs, assertions, and
dataset reference implementation are not being provided.  Return only
a complete Python implementation in one "'python code block.
```

The first instruction above is replaced by one of the following when a specific execution error occurs:

Syntax error

```
The implementation has a Python syntax or indentation error near line
{line_number}.  Repair the program structure and check that every
block is complete.
```

Runtime error

```
The implementation raised {exception_type} while a hidden functional
test was running.  Check initialization, input types, boundary cases,
and all return paths.
```

Timeout

```
The implementation did not finish within the execution limit.  Check
termination, recursion, loop bounds, and algorithmic complexity.
```

No extractable Python code

```
The response is unsatisfactory because no executable Python
implementation could be extracted.
```

E Derivation of Theorem

In this section, we provide the full proof of Theorem 1 and an associated key algebraic lemma utilized for the final derivation of said theorem.

E.1 Proof of Lemma

Lemma 2 (Confidence-Amplified Log Bound)**.** *For any $\mu \in (0, 1)$ and $p \in (\mu, 1)$:*

$$-\log(1-p) \;\geq\; \log p \;+\; h(\mu), \tag{11}$$

where $h(\mu)$ is as defined in equation 10. The bound is tight: equality holds at $p = 1/2$ when $\mu < 1/2$, in the limit $p \to 1/2^+$ when $\mu = 1/2$, and in the limit $p \to \mu^+$ when $\mu > 1/2$.

Proof. Define $f(p) := -\log(1-p) - \log p = \log \frac{1}{p(1-p)}$. Inequality equation 11 is equivalent to $f(p) \geq h(\mu)$ for all $p > \mu$.

Since $p(1-p) = \frac{1}{4} - (p - \frac{1}{2})^2 \leq \frac{1}{4}$ for all $p \in (0,1)$, with equality if and only if $p = \frac{1}{2}$, we have $f(p) \geq \log 4$ universally, with $f(p)$ uniquely minimized at $p^* = \frac{1}{2}$. Moreover, $f'(p) = \frac{2p-1}{p(1-p)}$, so f is strictly decreasing on $(0, \frac{1}{2})$ and strictly increasing on $(\frac{1}{2}, 1)$.

Case 1 ($\mu \leq \frac{1}{2}$): The domain $\{p > \mu\}$ contains $p^* = \frac{1}{2}$, so $\inf_{p>\mu} f(p) = f(\frac{1}{2}) = \log 4 = h(\mu)$.

Case 2 ($\mu > \frac{1}{2}$): f is strictly increasing on $(\mu, 1)$, so $\inf_{p>\mu} f(p) = \lim_{p\to\mu^+} f(p) = f(\mu) = \log \frac{1}{\mu(1-\mu)} = h(\mu)$.

In both cases $f(p) \geq h(\mu)$ for all $p > \mu$, establishing equation 11. □

E.2 Proof of Theorem

Proof of Theorem 1. Since $\lambda_w \ell_{\text{win}}(\boldsymbol{\theta}) \geq 0$ and $\lambda_l \ell_{\text{lose}}(\boldsymbol{\theta}) \geq 0$, the bound $\ell_{\text{SPEAR}}(\boldsymbol{\theta}) \leq \epsilon$ implies individually:

$$\ell_{\text{win}}(\boldsymbol{\theta}) \leq \frac{\epsilon}{\lambda_w}, \tag{12}$$

$$\ell_{\text{lose}}(\boldsymbol{\theta}) \leq \frac{\epsilon}{\lambda_l}. \tag{13}$$

Step 1: Win lower bound. Expanding ℓ_{win} from equation 4 for the single sample and applying equation 12:

$$-\sum_{i=0}^{|\mathbf{y}^+|-1} \log p_{\boldsymbol{\theta}}(y_i^+ \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^+) \;\leq\; \frac{\epsilon}{\lambda_w}.$$

Dividing by $|\mathbf{y}^+|$:

$$\frac{1}{|\mathbf{y}^+|} \log p_{\boldsymbol{\theta}}(\mathbf{y}^+ \mid \mathbf{c}^{(0)}) \;\geq\; -\frac{\epsilon}{\lambda_w |\mathbf{y}^+|}. \tag{14}$$

Step 2: Active-lose upper bound. Expanding ℓ_{lose} from equation 6 under Assumption 1 (so all tokens in $\mathcal{T}_\tau(\mathbf{y}^-)$ are active) and applying equation 13:

$$\alpha \sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \Big(-\log\big(1 - p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-)\big)\Big) \;\leq\; \frac{\epsilon}{\lambda_l}.$$

Dividing by $\alpha > 0$:

$$\sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \Big(-\log\big(1 - p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-)\big)\Big) \;\leq\; \frac{\epsilon}{\lambda_l \alpha}. \tag{15}$$

Step 3: Applying Lemma 2. By Assumption 1 and Lemma 2, for each $i \in \mathcal{T}_\tau(\mathbf{y}^-)$:

$$\log p_{\boldsymbol{\theta}}(y_i^- \mid c^{(0)}, \mathbf{y}_{<i}^-) \;\leq\; -\log\big(1 - p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-)\big) - h(\mu).$$

Summing over $\mathcal{T}_\tau(\mathbf{y}^-)$ and applying equation 15:

$$\sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \log p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-) \;\leq\; \frac{\epsilon}{\lambda_l \alpha} - |\mathcal{T}_\tau(\mathbf{y}^-)|\, h(\mu). \tag{16}$$

Dividing by $|\mathcal{T}_\tau(\mathbf{y}^-)|$:

$$\frac{1}{|\mathcal{T}_\tau(\mathbf{y}^-)|} \sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \log p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-) \;\leq\; \frac{\epsilon}{\lambda_l \alpha\, |\mathcal{T}_\tau(\mathbf{y}^-)|} - h(\mu). \tag{17}$$

Step 4: Assembling the margin bound. Substituting equation 14 and equation 17 into the definition of M_τ equation 8:

$$M_\tau(\boldsymbol{\theta}; \mathbf{y}^+, \mathbf{y}^-) = \frac{1}{|\mathbf{y}^+|} \log p_{\boldsymbol{\theta}}(\mathbf{y}^+ \mid \mathbf{c}^{(0)}) - \frac{1}{|\mathcal{T}_\tau(\mathbf{y}^-)|} \sum_{i \in \mathcal{T}_\tau(\mathbf{y}^-)} \log p_{\boldsymbol{\theta}}(y_i^- \mid \mathbf{c}^{(0)}, \mathbf{y}_{<i}^-)$$

$$\geq -\frac{\epsilon}{\lambda_w |\mathbf{y}^+|} - \left(\frac{\epsilon}{\lambda_l \alpha |\mathcal{T}_\tau(\mathbf{y}^-)|} - h(\mu) \right)$$

$$= h(\mu) - \epsilon \left(\frac{1}{\lambda_w |\mathbf{y}^+|} + \frac{1}{\lambda_l \alpha |\mathcal{T}_\tau(\mathbf{y}^-)|} \right),$$

which establishes equation 9 and completes the proof. The characterization of $h(\mu)$ follows from Lemma 2.

□

E.3 Theorem Extension to Batch-Level

Theorem 1 is stated for a single win-lose pair to make the mechanism transparent. However, the objective-level result extends directly to arbitrary finite minibatches containing multiple win and lose traces, as used in practice. Firstly, let the win and lose subsets induced by minibatch $\mathcal{B}_k$ be

$$\mathcal{W}_\mathcal{B} = \left\{ \left(\mathbf{c}_j^{(0)}, \mathbf{y}_j^+ \right) \right\}_{j=0}^{b_w - 1}, \qquad \mathcal{L}_\mathcal{B} = \left\{ \left(\mathbf{c}_r^{(0)}, \mathbf{y}_r^-, \alpha_r \right) \right\}_{r=0}^{b_l - 1},$$

where b_w and b_l denote the number of win and lose traces, respectively. Next, we define the set of lose tail tokens activated by the confidence gate as

$$A_r(\boldsymbol{\theta}) = \left\{ i \in \mathcal{T}_\tau(\mathbf{y}_r^-) : p_{\boldsymbol{\theta}} \left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^- \right) > \mu \right\}. \tag{18}$$

Now, we define the total number of win tokens and the weighted number of active lose tokens as

$$S_w = \sum_{j=0}^{b_w - 1} |\mathbf{y}_j^+|, \qquad S_l = \sum_{r=0}^{b_l - 1} \alpha_r |A_r(\boldsymbol{\theta})| \tag{19}$$

and define the token-weighted batch margin as

$$\overline{M}_\tau(\boldsymbol{\theta}) = \frac{\sum_{j=0}^{b_w - 1} \log p_{\boldsymbol{\theta}} \left(\mathbf{y}_j^+ \mid \mathbf{c}_j^{(0)} \right)}{S_w} - \frac{\sum_{r=0}^{b_l - 1} \alpha_r \sum_{i \in A_r(\boldsymbol{\theta})} \log p_{\boldsymbol{\theta}} \left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^- \right)}{S_l}. \tag{20}$$

Batch-Level Margin Extension. Suppose $S_w > 0$ and $S_l > 0$. If

$$\ell_{\text{SPEAR}}(\boldsymbol{\theta}) \leq \epsilon,$$

then

$$\overline{M}_\tau(\boldsymbol{\theta}) \geq h(\mu) - \epsilon \left(\frac{b_w}{\lambda_w S_w} + \frac{b_l}{\lambda_l S_l} \right). \tag{21}$$

Proof. As in the proof of Theorem 1, nonnegativity of the two components of the SPEAR objective implies

$$\ell_{\text{win}}(\boldsymbol{\theta}) \leq \frac{\epsilon}{\lambda_w}, \qquad \ell_{\text{lose}}(\boldsymbol{\theta}) \leq \frac{\epsilon}{\lambda_l}. \tag{22}$$

For a minibatch containing b_w win traces, the win loss is the average negative log-likelihood across win examples, resulting in

$$-\frac{1}{b_w} \sum_{j=0}^{b_w - 1} \log p_{\boldsymbol{\theta}} \left(\mathbf{y}_j^+ \mid \mathbf{c}_j^{(0)} \right) \leq \frac{\epsilon}{\lambda_w},$$

which gives

$$\frac{\sum_{j=0}^{b_w-1} \log p_{\boldsymbol{\theta}}\left(\mathbf{y}_j^+ \mid \mathbf{c}_j^{(0)}\right)}{S_w} \geq -\frac{b_w \epsilon}{\lambda_w S_w}. \tag{23}$$

Similarly, for the lose subset,

$$\frac{1}{b_l}\sum_{r=0}^{b_l-1}\alpha_r \sum_{i\in A_r(\boldsymbol{\theta})}\left[-\log\left(1-p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right)\right)\right] \leq \frac{\epsilon}{\lambda_l},$$

and therefore

$$\sum_{r=0}^{b_l-1}\alpha_r \sum_{i\in A_r(\boldsymbol{\theta})}\left[-\log\left(1-p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right)\right)\right] \leq \frac{b_l \epsilon}{\lambda_l}. \tag{24}$$

By construction, every $i \in A_r(\boldsymbol{\theta})$ satisfies

$$p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right) > \mu.$$

Hence applying Lemma 2 tokenwise results in

$$\log p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right) \leq -\log\left(1-p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right)\right) - h(\mu),$$

and then multiplying by α_r, summing over all active lose tokens and all lose examples, and using equation 24, we obtain

$$\begin{aligned}
&\sum_{r=0}^{b_l-1}\alpha_r \sum_{i\in A_r(\boldsymbol{\theta})} \log p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right) \\
&\qquad \leq \frac{b_l \epsilon}{\lambda_l} - h(\mu)\sum_{r=0}^{b_l-1}\alpha_r |A_r(\boldsymbol{\theta})| \\
&\qquad = \frac{b_l \epsilon}{\lambda_l} - S_l h(\mu).
\end{aligned} \tag{25}$$

Dividing by S_l gives

$$\frac{\sum_{r=0}^{b_l-1}\alpha_r \sum_{i\in A_r(\boldsymbol{\theta})} \log p_{\boldsymbol{\theta}}\left(y_{r,i}^- \mid \mathbf{c}_r^{(0)}, \mathbf{y}_{r,<i}^-\right)}{S_l} \leq \frac{b_l \epsilon}{\lambda_l S_l} - h(\mu). \tag{26}$$

Finally, substituting equation 23 and equation 26 into equation 20 yields

$$\begin{aligned}
\overline{M}_\tau(\boldsymbol{\theta}) &\geq -\frac{b_w \epsilon}{\lambda_w S_w} - \left(\frac{b_l \epsilon}{\lambda_l S_l} - h(\mu)\right) \\
&= h(\mu) - \epsilon\left(\frac{b_w}{\lambda_w S_w} + \frac{b_l}{\lambda_l S_l}\right),
\end{aligned}$$

which establishes equation 21. □

Theorem 1 is recovered as the single-pair special case $b_w = b_l = 1$. Under Assumption 1, $A_1(\boldsymbol{\theta}) = \mathcal{T}_\tau(\mathbf{y}^-)$, so that

$$S_w = |\mathbf{y}^+|, \qquad S_l = \alpha|\mathcal{T}_\tau(\mathbf{y}^-)|,$$

and equation 21 reduces exactly to equation 9.