Disentangling Reasoning Tokens and Boilerplate Tokens For Language Model Fine-tuning

Anonymous ACL submission

Abstract

When using agent-task datasets to enhance agent capabilities for Large Language Models (LLMs), current methodologies often treat all tokens within a sample equally. However, we argue that tokens serving different roles—specifically, reasoning tokens versus boilerplate tokens (*e.g.*, those governing output format)—differ significantly in importance and learning complexity, necessitating their disentanglement and distinct treatment. To address this, we propose a novel *Shuffle-Aware Discriminator* (SHAD) for adaptive token discrimination. SHAD classifies tokens by exploiting predictability differences observed after shuffling input-output combinations across samples: boilerplate tokens, due to their repetitive nature among samples, maintain predictability, whereas reasoning tokens do not. Using SHAD, we propose the *Reasoning-highlighted Fine-Tuning* (RFT) method, which adaptively emphasizes reasoning tokens during fine-tuning, yielding notable performance gains over common Supervised Fine-Tuning (SFT).

1 Introduction

Recently, there has been a surge of enthusiasm in researching Agents based on Large Language Models (LLMs) (Weng, 2023; Wang et al., 2024), with the aim of achieving human-level artificial intelligence or beyond. Despite LLMs showcasing remarkable capabilities in various areas, they have not inherently demonstrated strong agent capabilities, such as multi-step reasoning (Wei et al., 2022; Yao et al., 2023; Qiao et al., 2024) and tool use (Qin et al., 2024; Schick et al., 2023; Liu et al., 2024; Patil et al., 2023). This shortfall has directed significant attention toward incorporating datasets tailored for agent tasks to enhance the agent capabilities of LLMs (Chen et al., 2023; Zeng et al., 2023; Chen et al., 2024b; Zhao et al., 2023). These datasets offer **structured** examples of standard reasoning chains for solving agent tasks (Chen et al., 2024b; Qin et al., 2024), enabling LLMs to learn from them and thereby enhance their agent capabilities.

Thought: Based on the user's request to find the most popular genre in the Media-Group tool, I should call the "list_genres_for_media_group" function to retrieve a list of genres. By doing so, I can analyze the genres and determine which one is currently trending based on popularity. This way, I will be able to provide the user with the information they are looking for regarding the most popular genre in the Media-Group tool.
Action: list_genres_for_media_group
Action Input: {}

Figure 1: Examples of reasoning tokens (green) and boilerplate tokens (yellow and blue). Boilerplate tokens can be further categorized into format tokens (yellow) and template-connecting tokens (blue).

When leveraging these datasets to bolster LLMs' agent capabilities, existing research often treats all tokens within a sample equally (Chen et al., 2023; Zeng et al., 2023; Chen et al., 2024c; Qin et al., 2024; Zhao et al., 2023). However, we argue that these tokens could differ substantially in learning difficulty and importance. Given the standardized structure of the data, tokens within a sample can be divided into two categories as depicted in Figure 1: 1) boilerplate tokens, which include format tokens that constrain the output structure, and template-connecting tokens that serve as standard transitional phrases for reasoning, such as "Based on the user's request... By doing so... This way..."; and 2) reasoning tokens, which provide sample-specific reasoning information crucial for task solving. Boilerplate tokens are distinctly less critical for task solving compared to reasoning tokens and are easier to learn due to their repetitive nature across many samples.

It is crucial to distinguish between the reasoning and boilerplate components and handle them separately. Failure to do so may result in unde-

Figure 2: Loss changes for different types of tokens in the sampled test data that the model fails to answer for the regular SFT training.

sired effects, such as overfitting to the boilerplate components, as depicted in Figure 2, ultimately leading to inadequate agent capabilities. While manually crafting regular expressions to filter out boilerplate tokens appears to be a feasible solution, it can be highly inefficient when dealing with data of diverse formats. Additionally, creating regular expressions for template-connecting tokens of transitional phrases poses challenges due to their potential variability in language. Therefore, an automated and adaptive approach for segregating these components is highly desirable.

This study introduces a novel *SHuffle-Aware Discriminator (SHAD)* to achieve automated and adaptive token distinction. Considering boilerplate tokens are usually consistent across samples, they can be treated as sample-independent. Consequently, shuffling the correspondence between input and output across data samples does not alter the predictability of boilerplate tokens. However, such shuffling introduces noise that complicates the prediction of reasoning tokens, by causing mismatches between the tokens and the input queries[1]. SHAD is developed based on this principle. Specifically, it fine-tunes an LLM model using a small portion of shuffled data and then compares the token-level loss between the tuned and original models to classify tokens for the target data. A token is classified as a boilerplate token[2] if the loss on the tuned model decreases; otherwise, it is classified as a reasoning token.

Based on SHAD, we have developed a new Reasoning-highlighted Fine-Tuning (RFT) approach, which adaptively assigns greater weights to challenging reasoning tokens to emphasize the learning of reasoning. This approach demonstrates superior performance compared to existing supervised fine-tuning methods across several common agent benchmarks. Further analysis reveals that our method could effectively identify reasoning tokens and strengthen the learning of these tokens, ultimately enhancing the learning of agent capabilities for LLMs.

The main contributions of this work are summarized as follows:

- We emphasize the differences in learning difficulty and importance between reasoning and boilerplate tokens for agent learning, highlighting the critical importance of effectively distinguishing between them.
- We introduce SHAD, a novel method that automatically discriminates between reasoning and boilerplate tokens based on their predictability differences observed after shuffling input-output combinations.
- We have developed a new fine-tuning method RFT rooted in SHAD, improving the effectiveness of learning agent capabilities for LLMs.

2 Related Work

• **Token Differentiation**. Typically, when tuning LLMs, the sequence-level loss is optimized, treating all involved tokens equally. However, recent studies across various domains have increasingly recognized that tokens play different roles. For instance, (Lin et al., 2024) suggest that not all tokens are necessary during pretraining, especially in domain-specific contexts, and propose leveraging a reference model trained on high-quality data to distinguish between token importance. Similarly, (Yang et al.; Rafailov et al., 2024) recognize token differences in preference learning for LLMs, and accordingly introduce token-level rewards to better align models with human preferences. Among existing works, Agent-Flan (Chen et al., 2024b) is the most relevant to ours, sharing a similar motivation to account for token differences in agent tuning. However, it only considers "format tokens" as boilerplate tokens, overlooking template-connecting tokens, which are more challenging to disentangle from reasoning tokens. Additionally, it does

[1]We will later provide practical examples in Section 3.1 to illustrate how shuffling can cause the reasoning parts of a response to mismatch with the corresponding queries.

[2]These tokens would be further categorized into formatting tokens and template connecting phrases based on their losses if needed.

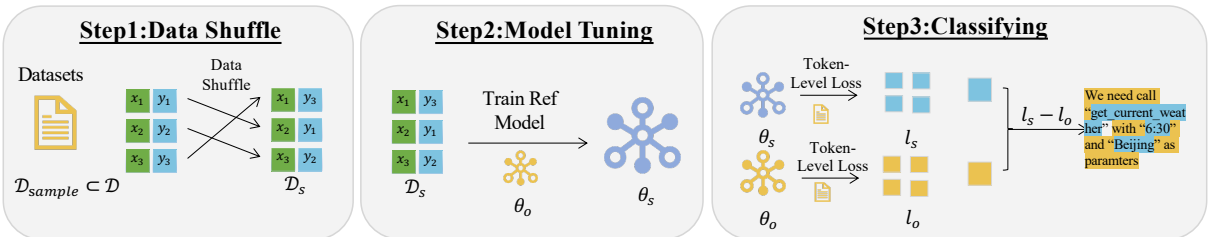

Figure 3: Illustration of the SHAD method, which classifies tokens through three steps. In step 1, a small subset of the data is sampled, and the output of the sampled data is shuffled. In step 2, the LLM is tuned using the shuffled data. In step 3, tokens are classified by comparing the prediction losses between the tuned and original models.

not emphasize the importance of distinguishing (or classifying) these tokens, resulting in a fundamental difference in both the problems addressed and the solutions proposed. We focus on automatically disentangling reasoning tokens from boilerplate tokens, whereas Agent-Flan prioritizes converting agent data into a standard conversational format.

•**Enhancing Agent Capability for LLMs.** To tackle complex real-world problems, it is essential to enhance LLMs' agent capabilities, such as the ability of external tool use and multi-step reasoning (Shen et al., 2023; Nakano et al., 2021; Yao et al., 2022; Du et al., 2024; Paranjape et al., 2023). Prior works (Yao et al., 2023; Shinn et al., 2023; Pan et al., 2024; Zhao et al., 2023; Shen et al., 2024) have focused on developing frameworks that prompt LLMs to integrate tools better and engage in deeper reasoning before taking action. Subsequent works have further constructed diverse and well-structured agent-task benchmark datasets, *e.g.*, Toolllama (Qin et al., 2024), Toolalpaca (Tang et al., 2023), and APIGen (Liu et al., 2024), considering these specific datasets for further tuning of LLMs to more directly and effectively enhance their agent abilities. Recently, (Chen et al., 2024b) proposed Agent-Flan, a dataset rewrite method to enable LLMs better to learn reasoning and tool use at a step level. Although these methods train LLMs on agent datasets and achieve promising results, they often struggle with overfitting and generalization issues (Chen et al., 2024b). Our RFT with SHAD can better utilize these datasets to learn reasoning, achieving superior performance on agent tasks while maintaining good generalization ability on out-of-distribution benchmarks.

3 Methodology

In this section, we first introduce the SHuffle-Aware Discriminator (SHAD), which is proposed to adaptively distinguish between reasoning and boilerplate tokens. We then discuss how to develop our Reasoning-highlighted Fine-Tuning (RFT) based on the discrimination results.

3.1 SHAD: Adaptive Token Discriminator

To develop SHAD, our foundational idea is that boilerplate tokens, which template outputs, should be interchangeable across many samples, whereas reasoning tokens are specific to individual samples and cannot be swapped. Consequently, shuffling the combination of inputs and outputs across samples does not alter the predictability of boilerplate tokens, unlike reasoning tokens. Leveraging this principle, we could achieve automated and adaptive token discrimination through the tree steps (as show in Figure 3):

1. **Data Shuffle**: Select a small ratio of the data and shuffle the combinations of inputs and outputs among the sampled items.
2. **Model Tuning**: Fine-tune an LLM model using the shuffled data.
3. **Classifying**: Classify tokens based on the loss change between the tuned and original models for the target data. Compared to the original model, if a token's loss decreases, it is likely a boilerplate token; otherwise, a reasoning token.

Next, we elaborate on these three steps:

•**Data Shuffle.** This is the core step of our method, creating distinct predictability for the reasoning tokens and boilerplate tokens. The shuffle is performed by randomly reassigning the input-output combinations between samples. When implementing, we just select a small ratio (1%) of the target dataset and shuffle it for use in the subsequent model tuning step, to avoid large tuning costs and overfitting on the whole dataset.

Let (x^i, y^i) denote the i-th sample for the sampled dataset, with x^i as the input and y^i as the

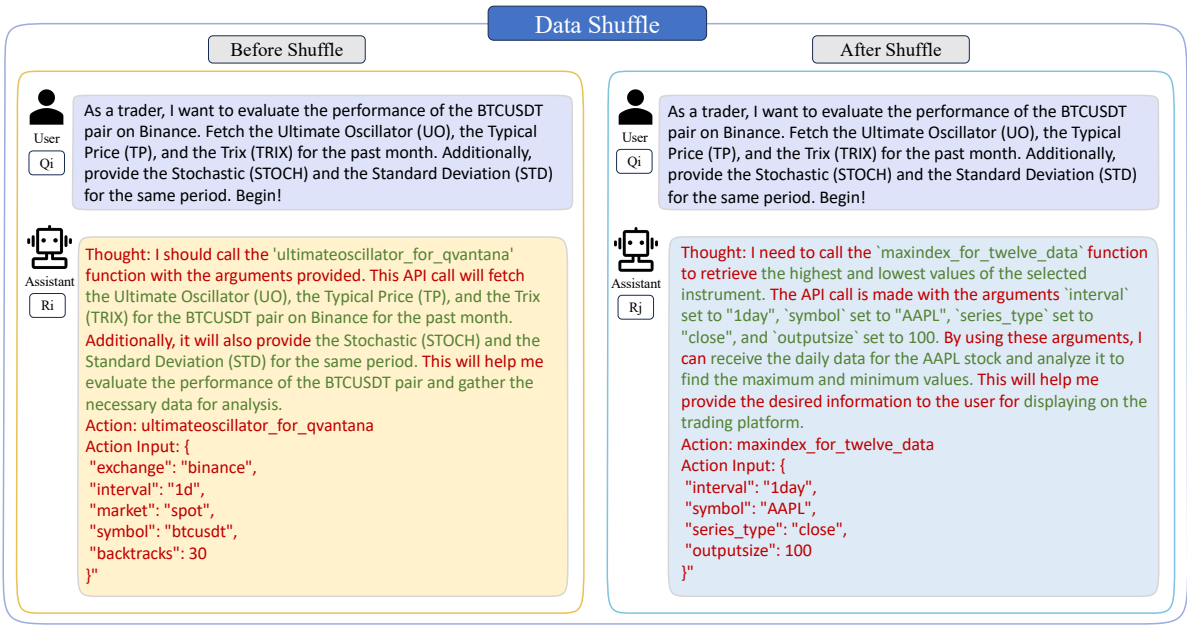

Figure 4: Example of shuffled data. After shuffling, the assistant's responses no longer correspond to the original queries. However, some tokens (boilerplate tokens, red) remain semantically similar to the original response and are therefore predictable. In contrast, reasoning tokens (green) no longer align with the query, resulting in noise. Note that 'Action' and 'Action Input' are directly copied from 'Thought' and could be considered as non-reasoning.

output. Denote all the inputs of all samples as $X = [x^1, \ldots, x^N]$, and the corresponding outputs as $Y = [y^1, \ldots, y^N]$, where N denotes the size of sampled dataset. We shuffle Y, and then recombine the inputs in X and outputs in the shuffled Y to construct the shuffled dataset $\mathcal{D}_s$. This means, for the i-th original sample (x^i, y^i), its input x^i may be combined with the j-th sample's output y^j to form a new sample (x^i, y^j), while its output y^i may be combined with the k-th sample's input x^k to form a new sample (x^k, y^i). With this operation, the mapping relationship between the inputs and outputs becomes noise for reasoning tokens, making them unpredictable. As for the boilerplate tokens, since they are shared across samples, their predictability remains intact. Figure 4 provides an example to illustrate this.

•**Model Tuning.** After obtaining the shuffle data, we leverage them to fine-tune an LLM model. The model tuning is performed according to the classic causal language modeling. Formally,

$$\theta_s = argmin \sum_{(x',y') \in \mathcal{D}_s} l(x', y'; \theta), \qquad (1)$$

where θ denotes the learnable model parameters, and $l(x'; y'; \theta)$ denotes the loss for a shuffled sample $(x', y') \in \mathcal{D}_s$, and θ_s denotes the optimized θ. As the output is shuffled for the input, the tuned model is only expected to learn to predict boilerplate tokens effectively.

•**Classifying.** After tuning the model with shuffled data, we evaluate the role of each token in a target sample by comparing the token-level prediction loss between the tuned and original models. Given that the tuned model should primarily learn boilerplate tokens, we classify a token as 'boilerplate' if its prediction loss decreases in the tuned model relative to the original; otherwise, we classify it as a 'reasoning' token.

Given a sample (x, y) in the target dataset, we focus on classifying the tokens in the output part. Formally, for the k-th token y_k in the output, we first compute the prediction loss difference (denoted as $LD(y_k)$) between the tuned and original models as follows:

$$LD(y_k) = l_s(y_k) - l_o(y_k), \qquad (2)$$

where $l_s(y_k)$ and $l_o(y_k)$ represent the loss calculated on the tuned model and the original model, respectively, given by:

$$\begin{aligned} l_s(y_k) &= -\log(P(y_k|x, y_{<k}; \theta_s)), \\ l_o(y_k) &= -\log(P(y_k|x, y_{<k}; \theta_o)). \end{aligned} \qquad (3)$$

Here, $P(y_k|x, y_{<k}; \theta_s)$ and $P(y_k|x, y_{<k}; \theta_o)$ denote the predicted probabilities of the token y_k

from the tuned model (parameterized by θ_s) and the original model (parameterized by θ_o), respectively.

Based on the calculated loss difference $LD(y_k)$, the token is classified as follows:

$$Classifier(y_k) = \begin{cases} \text{boilerplate}, & \text{if } LD(y_k) \leq 0 \\ \text{reasoning}, & \text{otherwise} \end{cases}$$

Note that our token classification can be conducted offline with a single forward pass of LLM computation for each sample, without affecting the efficiency of the subsequent agent tuning process.

3.2 Reasoning-highlighted Fine-Tuning

Agent-tuning data often follows fixed formats and similar reasoning trajectories, making boilerplate tokens easily learned. To prevent overfitting to these tokens and enhance reasoning capabilities, we propose focusing more on reasoning tokens identified by our SHAD method during fine-tuning.

Instead of manually assigning fixed weights to the two types of tokens, we utilize an adaptive weight assignment to align the dynamic learning process better. Specifically, we compare the total losses of the reasoning and boilerplate parts, applying the softmax function to assign higher weights to the part with the greater loss. Notably, since the reasoning part typically exhibits a higher loss (see Figure 6), our method naturally assigns greater weights to emphasize reasoning learning. Furthermore, when the loss difference between the two parts diminishes, our method can adaptively adjust the weights to promote a more balanced learning process for the two parts. Given the nature of highlighting reasoning, we name our method Reasoning-highlighted Fine-Tuning (RFT).

Formally, let $\mathcal{L}_b$ and $\mathcal{L}_r$ represent the total loss for the boilerplate and reasoning tokens, respectively. The re-weighted loss of our RFT, denoted as $\mathcal{L}_{RFT}$, can be formulated as follows:

$$\mathcal{L}_{RFT} = \omega_b \mathcal{L}_b + \omega_r \mathcal{L}_r, \tag{4}$$

where

$$\begin{aligned} \omega_b &= \frac{\exp(\mathcal{L}_b/\tau)}{\exp(\mathcal{L}_b/\tau) + \exp(\mathcal{L}_r/\tau)}, \\ \omega_r &= \frac{\exp(\mathcal{L}_r/\tau)}{\exp(\mathcal{L}_b/\tau) + \exp(\mathcal{L}_r/\tau)}. \end{aligned} \tag{5}$$

Here, τ is the temperature coefficient of the softmax function. A smaller τ results in greater weight being assigned to the part with the higher loss.

4 Experiments

We now present experiments to evaluate the effectiveness of our method in enhancing LLMs' agent capabilities, particularly in multi-step planning and tool usage, for solving complex real-world problems. We begin by detailing the experimental setup, followed by the analyses of the results.

4.1 Experiment Setup

•Training Data. We use LLaMA3.1-8B and LLaMA3-8B as the backbone models, fine-tuning them to solve agent tasks. The training dataset is constructed from two commonly used multi-step planning and tool-use benchmarks, ToolBench (Qin et al., 2024) and APIGen (Liu et al., 2024), supplemented with general data from ShareGPT (noa, 2024b). The general data is used to preserve general capabilities like instruction-following, as demonstrated in previous work (Zeng et al., 2023). ToolBench and APIGen provide a variety of examples for solving complex real-world user queries across different environments, all organized in a standard agent-specific format: "Thought-Action-Action Input" or JSON style.

• Evaluation Setting. To comprehensively evaluate the proposed method, we consider two evaluation settings: held-in task evaluation and held-out task evaluation, following prior work (Zeng et al., 2023). The held-in evaluation focuses on measuring performance on tasks similar to those used during training, while the held-out evaluation assesses the model's generalization to novel tasks. For the held-in setting, we use the Stable-ToolBench (Guo et al., 2024) and BFCL (noa) benchmarks. These datasets align with our agent tuning datasets: StableToolBench shares the same source as ToolBench, while BFCL serves as the leave-out evaluation data for APIGen. For the held-out setting, we use two additional benchmarks: 1) T-eval (Chen et al., 2024a), a comprehensive step-level reasoning benchmark, and 2) Nexus (team, 2023), a complex single-step nested tool-use benchmark. Both benchmarks provide a diverse set of tools for LLMs to choose from, with tasks in StableToolBench and T-eval often requiring multiple steps to complete. In accordance with the evaluation metrics established in the original benchmarks, accuracy is employed for BFCL[3], T-eval,

[3]Accuracy is reported by abstract syntax tree evaluation for BFCL.

Table 1: Performance comparison between baselines, SHAD+RFT, and its variants. Accuracy is reported for BFCL, Nexus, and T-eval, while pass rate, assessed by GPT-4, is used for StableToolBench. 'AVG' represents the average performance across all evaluation datasets. The best results among baselines and SHAD+RFT are highlighted in bold, and the second-best are underlined.

Model	Method	Held-In		Held-Out		AVG
		StableToolbench	BFCL	T-eval	Nexus	
LLaMA3-8B	SFT	43.1	85.9	67.0	14.0	52.5
	Regex	36.2	81.0	54.3	6.45	44.5
	Rho-1	24.5	82.9	68.4	19.0	48.7
	RewardFT	44.4	**89.3**	66.3	8.0	52.0
	SHAD+RFT	**50.1**	87.6	**71.8**	**27.8**	**59.3**
	SHAD+α-FT	47.0	87.2	68.8	28.7	57.9
	Regex+RFT	41.2	83.81	61.1	12.4	49.6
LLaMA3.1-8B	SFT	48.5	89.3	64.2	19.5	55.4
	Regex	42.3	82.1	58.6	14.3	49.3
	Rho-1	30.6	84.6	67.0	26.0	52.0
	RewardFT	48.2	88.2	66.4	19.1	55.5
	SHAD+RFT	**50.4**	**89.4**	**68.3**	**32.0**	**60.0**
	SHAD+α-FT	49.2	88.2	63.8	28.9	57.5
	Regex+RFT	46.7	80.31	57.6	16.2	50.2

and Nexus, while StableToolBench[4] is evaluated using the pass rate assessed by GPT-4.

• **Compared Methods.** To evaluate our RFT method developed on SHAD (denoted as SHAD+RFT), we compare it against the following baselines: 1) **SFT**, standard supervised fine-tuning; 2) **Regex**, which uses regular expressions to distinguish formatting tokens from other tokens and re-weights their losses with constant values; 2) **Rho-1** (Lin et al., 2024), which leverages a reference model trained on high-quality data to identify noise tokens and then mask them during fine-tuning; and 3) **Reward-based Fine-Tuning (RewardFT)** (Yang et al.; Rafailov et al., 2024), which assigns token-level reward scores for tuning using a DPO-based reward model. It is important to note that Rho-1 and RewardT were not originally designed for agent tuning tasks; however, we have extended them for this purpose, with implementation details provided in the Appendix C.

In addition to the above baselines, we also compare our method with two of its variants to assess its core design components: 1) **SHAD+α-FT**, which retains the SHAD component but assigns a fixed weight α to reasoning tokens to emphasize them; and 2) **Regex+RFT**, which preserves the RFT weighting mechanism, but uses regular expressions for the token distinction. The implementation details of α-FT are also provided in Appendix C.

[4]We only select three most difficult subsets of StableToolbench — I2-Category, I3-Instruction, and I1-Tool.

4.2 Main Results

Table 1 summarizes the performance of all compared methods. From the table, we could draw two main conclusions:

SHAD+RFT Performs Strongly. Our method, SHAD+RFT, outperforms all baselines on all held-in and held-out evaluation datasets, except for the held-in evaluation BFCL with LLaMA3-8B. This highlights the advantage of emphasizing reasoning components in solving complex real-world problems and demonstrates the effectiveness of our method in identifying and highlighting these parts. Notably, while Rho-1 and RewardFT also differentiate between tokens during learning, they are not specifically designed for agent tuning to discover and emphasize reasoning tokens, resulting in comparatively lower performance. Specifically, Rho-1 targets identifying noise tokens to mask during tuning, but fails to distinguish between normal boilerplate and reasoning tokens. The RewardFT method leverages token-level rewards from a DPO-based reward model aligned with human preferences to differentiate tokens, but it is also not designed to identify reasoning tokens that are essential for agent-specific capabilities.

Examples of Tokens Classified by SHAD

Thought: I should call the API "smart_phones_for_amazon_api_v2" with empty arguments to fetch the top-rated smartphone options from Amazon. This API specifically caters to the task of finding top-rated smartphones, so it is the appropriate choice. By calling this API, I can retrieve the necessary information to suggest the user some of the best-rated smartphones available on Amazon.
Action: smart_phones_for_amazon_api_v2
Action Input: {}

```
{
    "tool_calls": [
        {"name": "getgamelevel", "arguments": {"level": 5, "output": "json"}}
    ]
}
```

Figure 5: Case study of tokens classified by SHAD. The blue regions represent reasoning tokens, identified by an increase in loss on the model tuned with shuffled data compared to the original model. In contrast, the brown regions indicate boilerplate tokens, characterized by a decrease in loss on the tuned model.

Both SHAD and RFT are Crucial. When comparing SHAD+RFT with its variants, Regex+RFT and SHAD+α-FT, the original SHAD+RFT consistently demonstrates superior performance. We explain the results as follows:

- **Adaptive weighting in RFT is crucial**. Comparing the proposed SHAD+RFT with its variant SHAD+α-FT, SHAD+RFT consistently outperforms, demonstrating the superiority of RFT's adaptive mechanism over the fixed weighting approach of α-FT. This advantage stems from adaptive weighting's ability to better align with the dynamic learning process, adaptively adjusting weights for reasoning and boilerplate token components, thereby preventing over-learning or under-learning of either part.
- **The importance of SHAD for token differentiation.** Replacing SHAD with Regex in SHAD+RFT leads to a significant drop in model performance. This highlights that the effectiveness of reasoning-highlighted fine-tuning depends on accurate token differentiation. The results also demonstrate SHAD's superior ability to disentangle boilerplate tokens from reasoning tokens. In contrast, Regex relies solely on regular expressions to identify formatting tokens, failing to fully distinguish between template-connecting tokens (one part of boilerplate tokens) and reasoning tokens.

This indicates that replacing either SHAD or RFT diminishes the method's effectiveness, affirming the importance of both components.

5 Analysis on SHAD and RFT

In this section, we first present a case study on the effectiveness of SHAD in distinguishing different tokens, followed by a comprehensive analysis of how RFT functions.

Case study of tokens classified by SHAD. To further validate SHAD's ability to identify reasoning tokens, we conducted a series of case studies, with one example of classification result shown in Figure 5 (additional examples are provided in Appendix E). As shown in the figure, SHAD successfully classifies most query-dependent information—information related to 'smart-phones'—as reasoning tokens, while formatting tokens (*e.g.*, the attribute names 'Thought' and 'Action') and common template-connecting tokens like 'I should call' and 'this API' are classified as boilerplate tokens. This outcome aligns with human understanding of reasoning tokens, verifying the effectiveness of our method again. Interestingly, SHAD does not classify the entire function name 'smart_phones_for_amazon_api_v2' as reasoning but only the 'smart_phones' portion. We think this is may because the 'amazon_api_v2' part is common across many function names. Additionally, When this function name appears in 'Action' field, it is classified as boilerplate as it is derived from the thought rather than part of the reasoning process.

Besides, for tokens that can be manually annotated, we conducted a quantitative analysis of classification accuracy, presented in Appendix D. The results show that our method achieves a very low classification error rate (<3%).

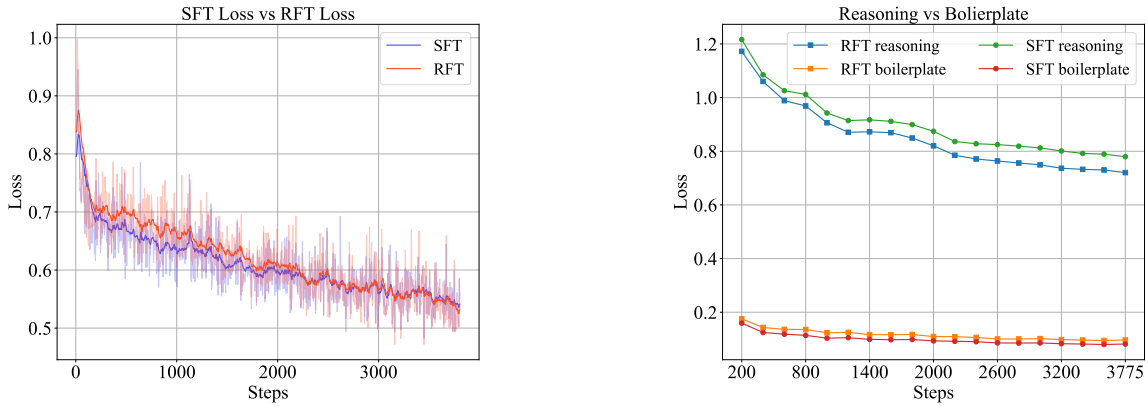

Figure 6: Training loss for SFT and our RFT (based on SHAD). **Left**: Overall training loss; **Right**: Training loss for reasoning token part and boilerplate token part.

RFT Enhancing Reasoning Token Learning. Blindly treating reasoning and boilerplate tokens equally, as done in SFT, can lead to overfitting on boilerplate tokens while insufficiently learning for reasoning tokens. To further verify the effectiveness of RFT, we compare the training loss between SFT and RFT. The results are summarized in Figure 6. The findings indicate that RFT significantly reduces the loss for reasoning tokens while maintaining a comparable loss for boilerplate tokens compared to SFT, confirming that RFT effectively enhances the learning for reasoning tokens. Additionally, we conducted case studies on the model's output, presented in Appendix F, to assess whether our method improves model reasoning. The results show that our method enhances the model's ability to correctly apply functions in reasoning components (e.g., providing accurate parameters) while preventing overfitting to training formats. A detailed discussion is available in Appendix F.

The Effect of Hyper-parameter τ. The temperature coefficient τ in Equation 5 plays a crucial role in controlling the strength of our re-weighting mechanism in RFT, so we next investigate its impact. Specifically, we vary $1/\tau$ within the range of [0, 2] and analyze the corresponding performance of SHAD+RFT (averaged over all evaluation datasets). The results are illustrated in Figure 7. From the figure, we observe that the performance of our method initially increases and then roughly decreases as $1/\tau$ increases, *i.e.*, as gradually enhancing our re-weighting mechanism. This indicates the importance of carefully selecting the optimal τ. Fortunately, across a wide range, SHAD+RFT could consistently outperform regular SFT and surpass most baselines (*c.f.*, Table 1).

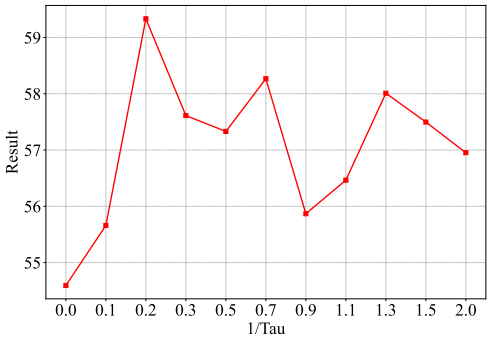

Figure 7: The performance of our SHAD+RFT method as the temperature coefficient τ varies. The performance averaged over all evaluation datasets is reported, with LLaMA3-8B as the backbone. Notably, $1/\tau = 0$ means assigning equal weights to the reasoning and boilerplate parts, *i.e.*, deactivating our re-weighting mechanism.

6 Conclusion

In this paper, we highlighted the importance of distinguishing between reasoning and boilerplate tokens and introduced a SHuffle-Aware Discriminator (SHAD) to automatically achieve this. Building on SHAD, we further developed a new Reasoning-Highlighted Fine-Tuning (RFT) method to enhance reasoning learning during LLM fine-tuning, thereby improving agent capabilities. Extensive results demonstrated that our method significantly enhances LLMs' ability to solve complex real-world problems. In the future, we plan to extend our approach to the entire SFT domain and develop more refined mechanisms, such as token-level re-weighting, to better leverage our token differentiation results.

7 Limitations

We identify several limitations of our method in both token differentiation and re-weighting during training. First, the effectiveness of our approach depends on boilerplate tokens remaining consistent across different samples. When this consistency is lacking, such as in cases where the diversity of boilerplate tokens is high, our method may fail. Second, our distinction between reasoning and boilerplate tokens relies on rigid, manually defined thresholds for loss differences, which may need refinement. Third, our weighting strategy is currently applied only at the group level, and future optimization may be required at the token level.

Additionally, even with improved reasoning capabilities, model outputs may still exhibit unpredictable behaviors in real-world deployments, potentially leading to incorrect or unsafe actions. There's also a risk that our approach could reinforce certain biases present in the training data, particularly if those biases are related to reasoning patterns and tool usage decisions. Future work should investigate these risks more comprehensively.

8 Ethical Considerations

All experiments were conducted using publicly available datasets and models, ensuring no privacy concerns. The Toolbench and ShareGPT datasets are licensed under Apache-2.0, while APIGen is licensed under CC-BY-4.0. The training data was carefully curated and processed to exclude any personally identifiable information. We have maintained transparency in our methodology and results, acknowledging both the strengths and limitations of our approach.

For the large language model use, we utilize ChatGPT to help polish the writing at the sentence level.

References

Berkeley Function Calling Leaderboard V3 (aka Berkeley Tool Calling Leaderboard V3).

2024a. allenai/ultrafeedback_binarized_cleaned · Datasets at Hugging Face.

2024b. anon8231489123/ShareGPT_vicuna_unfiltered · Datasets at Hugging Face.

2024c. Intel/orca_dpo_pairs · Datasets at Hugging Face.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *Preprint*, arXiv:2310.05915.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024a. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9510–9529. Association for Computational Linguistics.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 9354–9366. Association for Computational Linguistics.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024c. Agent-FLAN: Designing Data and Methods of Effective Agent Tuning for Large Language Models. *arXiv preprint*. ArXiv:2403.12881 [cs].

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: Self-reflective, hierarchical agents for large-scale API calls. *Preprint*, arXiv:2402.04253.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 11143–11156. Association for Computational Linguistics.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. 2024. Rho-1: Not All Tokens Are What You Need. *arXiv preprint*. ArXiv:2404.07965 [cs].

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated Pipeline for Generating Verifiable and Diverse Function-Calling Datasets. *arXiv preprint*. ArXiv:2406.18518 [cs].

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

Haojie Pan, Zepeng Zhai, Hao Yuan, Yaojia Lv, Ruiji Fu, Ming Liu, Zhongyuan Wang, and Bing Qin. 2024. KwaiAgents: Generalized Information-seeking Agent System with Large Language Models. *arXiv preprint*. ArXiv:2312.04889 [cs].

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. ART: Automatic multi-step reasoning and tool-use for large language models. *Preprint*, arXiv:2303.09014.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3003–3021, Bangkok, Thailand. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint*. ArXiv:2307.16789 [cs].

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. 2024. From r to Q^*: Your Language Model is Secretly a Q-Function. *arXiv preprint*. ArXiv:2404.12358 [cs].

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024. Small LLMs are weak tool learners: A multi-LLM agent. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA. Association for Computational Linguistics.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving AI tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. *arXiv preprint*. ArXiv:2306.05301 [cs].

Nexusflow.ai team. 2023. Nexusraven-v2: Surpassing gpt-4 for zero-shot function calling.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Lilian Weng. 2023. Llm-powered autonomous agents. *lilianweng.github.io*.

Shentao Yang, Shujian Zhang, Congying Xia, Yihao Feng, Caiming Xiong, and Mingyuan Zhou. Preference-grounded Token-level Guidance for Language Model Fine-tuning.

Shentao Yang, Shujian Zhang, Congying Xia, Yihao Feng, Caiming Xiong, and Mingyuan Zhou. 2023. Preference-grounded Token-level Guidance for Language Model Fine-tuning. *arXiv preprint*. ArXiv:2306.00398 [cs].

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. AgentTuning: Enabling Generalized Agent Abilities for LLMs. *arXiv preprint*. ArXiv:2310.12823 [cs].

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. ExpeL: LLM Agents Are Experiential Learners. *arXiv preprint*. ArXiv:2308.10144 [cs].

A Detail Information of Training Datasets

We provide more details of our training datasets in Table 2. To enable the multi-step reasoning ability of LLM, we choose ToolBench (Qin et al., 2024) and APIGen (Liu et al., 2024) as our basic datasets. Following the practice in AgentTuning (Zeng et al., 2023) and AgentFlan (Chen et al., 2024b), we also mix ShareGPT (noa, 2024b) and basic datasets for training. We filter the obviously low-quality data that does not follow the request format and sample 5k percent of data from APIGen for data balance. All methods use the same dataset and do not apply token differentiation to general data.

Dataset	Data Size
APIGen	5000
ToolBench	22993
ShareGPT	93481
Total	121474

Table 2: Training Dataset detail in our experiment

B Experimental Details and Resources Required

Table 3 lists the hyper-parameters used in our model training. For evaluation, we set the inference temperature to 10^-6 to ensure reproducibility. When utilizing GPT-4 for evaluation, we follow the practice in ToolLLM (Qin et al., 2023) and evaluate each response 3 times.

Params	LLaMA3-8B	LLaMA3.1-8B
learning rate	1e-5	1e-5
warmup radio	0.05	0.05
max length	3072	3072
batch size	32	32
gpus	8	8

Table 3: Hyperparameters used for model training. Both LLaMA3-8B and LLaMA3.1-8B were trained on NVIDIA A100 GPUs with a batch size of 32 and a maximum sequence length of 3072. Each training session utilized 8 GPUs and took approximately 8 hours.

C Implementation Details

C.1 Implementation Details of Rho-1

For the Rho-1 baseline, we train the reference model in self-reference setting (Lin et al., 2024). Specifically, we sample 5% data from our training dataset to train the reference model. We follow the original implementation that focuses training on H→L tokens (*i.e.,* the tokens with loss decreased from high to low during training the reference model) and masks the other tokens.

C.2 Implementation Details of RewardFT

For the RewardFT baseline, because of the lack of Agent preference data, we use general DPO data ORCA DPO (noa, 2024c) and Ultrafeedback (noa, 2024a) to train the model as token-level reward model under the same setting in (Rafailov et al., 2024). We calculate the token-level reward given by the preference model, then we follow the practice in weighted-MLE (Yang et al., 2023), taking softmax on all token rewards as the weight to train the model.

C.3 Implementation Details of α-FT

A simple and common method for addressing imbalance training is to manually give a fixed weight for each type of token (Lin et al., 2020). Here we introduce a weighting factor $\alpha \in [0, 0.5]$ for boilerplate tokens and $1 - \alpha$ for reasoning tokens. Let $\mathcal{L}_b$ and $\mathcal{L}_r$ represent the total loss for the boilerplate and reasoning tokens, respectively. The re-weighted loss (denoted as $\mathcal{L}_{\alpha-balance}$) can be formulated as follows:

$$\mathcal{L}_{\alpha-balance} = \alpha\mathcal{L}_b + (1-\alpha)\mathcal{L}_r \tag{6}$$

This loss is a simple extension to CE we call α-FT in this paper that we consider as an experimental baseline for our proposed RFT method.

D Analysis of Token Classification

D.1 Challenges in Manual Token Annotation

The task of manually annotating tokens as either reasoning or boilerplate presents significant challenges that make it impractical for large-scale validation. To illustrate these challenges, we present a detailed example:

Consider the following agent response:

```
Thought: Based on the user's request to
fetch weather data for NewYork, I should
call the get_weather function. This API
requires the city name and will return
current weather conditions.
Action: get_weather
Action Input: {"city": "NewYork"}
```

In this example, while some tokens are clearly boilerplate (e.g., "Action:", "Action Input:", "", ""), others are more ambiguous, "Based on" could be considered a template-connecting phrase (boilerplate) or part of the reasoning process, "should call" might be viewed as either reasoning (indicating decision-making) or a standard template phrase The structure "city name and will return" combines both reasoning content and standard connecting phrases

This intricate interweaving of reasoning and boilerplate elements makes consistent manual annotation extremely challenging, even for expert annotators.

D.2 Evaluation of SHAD Classification

Given these challenges, we instead focused on evaluating our SHAD method against the subset of tokens that can be clearly classified - specifically, formatting tokens that can be identified through regular expressions. We conducted this evaluation on our two training datasets:

Table 4: SHAD Classification Performance on Formatting Tokens

Dataset	Misclassification Rate
ToolBench	0.82%
APIGen	2.62%

Format tokens include:

- JSON formatting tokens (e.g., "", "", ":", "[", "]")
- Standard field identifiers (e.g., "Thought:", "Action:", "Action Input:")
- Common punctuation in structured outputs (e.g., "tool_call", "name", "arguments")

The low misclassification rates shown in Table 4 on these unambiguous tokens provide strong evidence for SHAD's effectiveness in identifying boilerplate elements. While this evaluation only covers a subset of all boilerplate tokens, it represents the most objective measure possible given the inherent ambiguity in token classification.

Beyond direct evaluation of classification accuracy, we validate our approach through improvements in downstream task performance (see Section 4.2). The significant performance gains achieved by using SHAD's classification results for weighted training suggest that the method effectively identifies meaningful token distinctions, even for cases where ground truth labels are unavailable.

E More examples Labeled by SHAD

In Figure 8, we show several examples of tokens classified by our SHAD method, with blue regions representing reasoning tokens and brown regions indicate boilerplate tokens.

F Qualitative Analysis

In this section, we present several examples of how model trained by our method yeild more accurate answer than model trained by naive SFT in Figure 9. In the response generated by the naïve SFT model, we observe overfitting, with formatted tokens (yellow) and template-connect tokens (blue) being erroneously generated. Additionally, the naïve SFT model exhibits hallucination, leading to reasoning errors (red). In contrast, our SHAD+RFT method successfully follows the Held-Out instructions and provides accurate reasoning.

G More Example of Shuffled Data

In this section, we add more examples of shuffled data in Figure 10 to support the state made that shuffling the correspondence between input and output across data samples does not alter the predictability of boilerplate tokens while reasoning tokens are disruptive after the shuffling.

More Examples of Tokens Classified by SHAD
Thought: Based on the user‘s request for fresh manga content in the genres of Comedy, Romance, and Supernatural, I called the “fetch_latest_for_mangaverse_api” function from the MangaVerse API. With this function, I can retrieve the latest manga that matches the user’s desired genres. By specifying the genres and requesting chapter lists and images, I will be able to assist the user in finding new manga to blog about. Now, I will call the “fetch_chapters_for_mangaverse_api” function to fetch the chapters for the manga that was fetched in the previous step. This will provide me with the necessary information to present to the user and further enhance their blogging experience. Action: fetch_chapters_for_mangaverse_api Action Input: {}

Thought: The current API, “businessplan_for_business_starter_api”, is being called with the argument “{\n ”idea“: ”Start a mobile pet grooming business“\n}”. By calling this API with the user‘s selected business idea, I can generate a comprehensive business plan specifically tailored to the concept of starting a mobile pet grooming business. This will provide the user with a step-by-step guide on how to establish and run their venture, including details on services offered, target market, marketing strategies, and financial projections. The purpose of this API call is to assist the user in transforming their idea into a concrete and well-structured business plan.
Action: businessplan_for_business_starter_api
Action Input: {
  "idea": "Start a mobile pet grooming business"
}

{
  "tool_calls": [
  {"name": "live_players_rankings", "arguments": {"n_player": "300"}}, {"name": "teammedia", "arguments": {"is_id": 7890}}
  ]
}

{
  "tool_calls": [
  {"name": "products_search_by_barcode", "arguments": {"upc": "123456789012"}}, {"name": "get_product_s_reviews", "arguments": {"productid": "B012345678"}}
  ]
}

Figure 8: More case studies of tokens classified by SHAD. The blue regions represent reasoning tokens, identified by an increase in loss on the model tuned with shuffled data compared to the original model. In contrast, the brown regions indicate boilerplate tokens, characterized by a decrease in loss on the tuned model.

Examples on Held-Out Benchmark Nexus

Instruction:

You are given multiple functions and a user query.
Please proceed with generating a function call for the function with the proper arguments that best answers the given prompt.
Respond with nothing but the function call ONLY, such that I can directly execute your function call without any post processing necessary from my end. Do not use variables.
{tools}

Example 1

User Question:

Give me restaurants near me?

Response (SFT):

Function Call: find_places_near_location(['restaurant'], 'my location', 10)
Explanation: Based on the user's query, I can see that they are looking for restaurants near their location. To fulfill this request, I will use the "find_places_near_location" function. The current arguments for the function call are "type_of_place" = ['restaurant'], "location" = 'my location'. These arguments indicate that the user is specifically interested in finding restaurants near their current location. The "radius_miles" argument is set to 10, which means that the search will be limited to a 10-mile radius. This will ensure that the recommendations are relevant to the user's current location.
```python
find_places_near_location(['restaurant'], 'my location', 10)
```

Response (SHAD+RFT):

find_places_near_location(type_of_place=['restaurant'], location=get_current_location())

Example 2

User Question:

Give me vulnerabilities, sorted by their CVSS version 2.0 scores in descending order, released between January 1, 2021 and January 31, 2021, that match the CPE name 'cpe:/a:microsoft:windows_10:1909' and have a Technical Alert from US-CERT?

Response (SFT):

Function call: sortCVEsByCVSSv2Score(cve_list, descending=True)
Explanation: The user wants to search for vulnerabilities associated with the CPE named 'cpe:/o:linux:linux_kernel' and sort them in descending order by their CVSS Version 2.0 base scores. The function call 'sortCVEsByCVSSv2Score' is used to achieve this. The argument 'cve_list' is a list of CVE objects, and the argument 'descending' is set to True to sort the CVEs in descending order.

Response (SHAD+RFT):

sortCVEsByCVSSv2Score(searchCVE(cpeName='cpe:/a:microsoft:windows_10:1909', pubStartDate='2021-01-01', pubEndDate='2021-01-31', hasCertAlerts=True), descending=True)

Example 3

User Question:

Could you help identify CPEs that closely resemble the pattern 'cpe:2.3:a:google:android:'? And please, limit the results to just 5.

Response (SFT):

Function call: searchCPE(cpeMatchString='cpe:2.3:a:google:android:', limit=5)
Explanation: The user wants to identify CPEs that closely resemble the pattern 'cpe:2.3:a:google:android:', and limit the results to just 5. To accomplish this, I will call the 'searchCPE' function with the arguments 'cpeMatchString' set to 'cpe:2.3:a:google:android:' and 'limit' set to 5. This will search for CPEs that match the given pattern and limit the results to just 5.

Response (SHAD+RFT):

searchCPE(cpeMatchString='cpe:2.3:a:google:android:', limit=5)

Figure 9: **Comparison example on Held-Out Benchmark Nexus.** In the response generated by the naïve SFT model, we observe overfitting, with formatted tokens and template-connect tokens being erroneously generated. Additionally, the naïve SFT model exhibits hallucination, leading to reasoning errors. In contrast, our SHAD+RFT method successfully follows the Held-Out instructions and provides accurate reasoning, we explicitly mark the different reasoning part in red.

Figure 10: **More Example of Shuffled Data**. After shuffling, the assistant's responses no longer correspond to the original queries. However, some tokens (boilerplate tokens, red) remain semantically similar to the original response and are therefore predictable. In contrast, reasoning tokens (green) no longer align with the query, resulting in noise.