When the API Speaks the Wrong Language: Revisiting Post-Training for Multilingual Tool Use

Siddharth Chauhan [1] **Thomas Butler** [1] **Abhishek Singhania** [1] **Pankaj Porwal** [1] **Honey Gupta** [1]

Abstract

The reliability of Large Language Models (LLMs) for API calling degrades in multilingual settings. A common failure occurs when a model selects the correct tool but generates argument values in an inconsistent language, which we term *Argument Language Mismatch* (ALM). Although semantically correct, such outputs are operationally invalid and not captured by standard API-calling metrics. We revisit post-training strategies for mitigating ALM and find that, in our benchmark, supervised fine-tuning (SFT) provides a strong baseline, substantially improving argument language consistency and end-to-end function call accuracy. Under consistent model selection, SFT achieves performance comparable to, and sometimes exceeding more complex reinforcement learning (RL) approaches. We further examine whether RL with structured, argument-aware rewards offers additional benefits. While methods such as Group Relative Policy Optimization (GRPO) can improve language consistency and better preserve general reasoning ability, these gains are incremental and most pronounced in generalization and multi-objective trade-offs. Overall, our results suggest that much of the performance in multilingual API grounding can be achieved through careful supervised training, with RL providing targeted rather than fundamental improvements.

1. Introduction

Large language models (LLMs) increasingly interact with external systems via structured API calls (Brown et al., 2020; Devlin et al., 2019). In such settings, correctness requires not only selecting the appropriate API but also generating precisely formatted argument values. While recent advances have significantly improved tool-use capabilities in English, reliability degrades sharply in multilingual scenarios (Hu et al., 2023).

A common and underexplored failure mode arises when a model selects the correct API but generates argument values in a language inconsistent with the user input or system requirements. We refer to this phenomenon as *Argument Language Mismatch* (ALM). Although such outputs are often semantically correct, they are operationally invalid in real-world systems that enforce strict language constraints, leading to complete task failure. ALM is not a semantic or intent-recognition error. As illustrated in Figure 1, models frequently succeed at intent understanding and API selection, yet fail to condition argument realization on the user's language. Because standard API-calling metrics treat these cases as generic invocation errors, they obscure a major source of failure in multilingual agentic systems.

Existing approaches to multilingual modeling, including supervised fine-tuning (SFT) (Conneau et al., 2020; Wang et al., 2022; Lin et al., 2021; Hu et al., 2023), have demonstrated strong performance in instruction-following and tool-use tasks. In particular, SFT can effectively improve argument-level language consistency when training and test distributions are aligned. However, it remains unclear whether such improvements extend robustly to more challenging settings, such as unseen APIs, diverse argument structures, or cross-lingual transfer. At the same time, recent work suggests that reinforcement learning (RL) with structured objectives can improve alignment in complex generation tasks (Ouyang et al., 2022b; Xu et al., 2024; Gehring et al., 2024), raising the question of whether such methods are necessary for addressing ALM.

In this work, we visit post-training strategies for mitigating ALM in multilingual API calling. Rather than assuming that increasingly complex objectives are required, we first examine how far strong supervised baselines can go. Surprisingly, we find that supervised fine-tuning alone resolves a large fraction of ALM errors, yielding substantial improvements in both argument language consistency and end-to-end function call accuracy. Under controlled and consistent model selection, SFT achieves performance comparable to, and in some cases exceeding, more complex RL-based approaches.

[1]Amazon. Correspondence to: Honey Gupta <ghoney@amazon.com>.

Preprint. August 13, 2026.

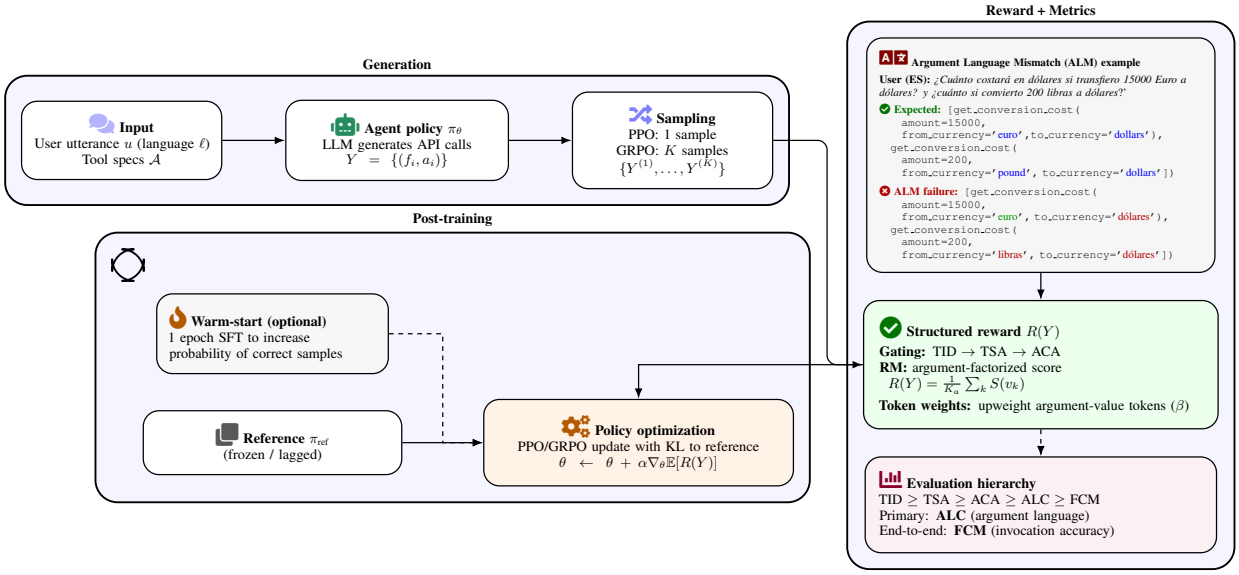

Figure 1. **Overview of our language-consistent API grounding framework.** Given a multilingual user request and available tool specifications, the model generates candidate API calls. A structured reward evaluates both structural correctness and whether argument values match the language of the user input. Reinforcement learning with argument-aware rewards encourages the model to produce language-consistent arguments, improving language consistency (ALC) and end-to-end function call accuracy (FCM). *Inset:* Example of Argument Language Mismatch (ALM), where the model selects the correct tool but generates argument values in the wrong language.

We then investigate whether reinforcement learning with structured, argument-aware rewards provides additional benefits beyond SFT. To this end, we formulate multilingual API calling as a structured generation problem and design reward functions that explicitly evaluate argument language consistency. Our approach includes hierarchical rewards aligned with the API-calling process, argument-factorized credit assignment, and token-level reward weighting. We study both *Proximal Policy Optimization* (PPO) (Schulman et al., 2017) and *Group Relative Policy Optimization* (GRPO) (Shao et al., 2024), enabling a controlled comparison of optimization strategies under identical reward formulations. This setup allows us to isolate when and how RL contributes beyond supervised learning.

To support this study, we construct a multilingual extension of the Berkeley Function Calling benchmark (Patil et al., 2024), covering multiple languages while preserving realistic API structure and argument diversity. We evaluate models under both *learnability* and *generalization* settings, as well as cross-lingual transfer to unseen languages. Across these settings, we find that reinforcement learning provides incremental improvements over strong SFT baselines, with the most consistent gains observed in generalization and in preserving general reasoning ability.

In summary, this paper makes three key contributions:

- We formalize *Argument Language Mismatch* (ALM) as a distinct and practically important failure mode in multilingual API calling.
- We show that supervised fine-tuning (SFT) provides a strong baseline for mitigating ALM, achieving substantial improvements in language consistency and end-to-end accuracy.
- We evaluate reinforcement learning with structured, argument-aware rewards and find that it yields incremental gains over SFT, particularly in generalization and reasoning preservation.
- We provide a systematic comparison of post-training strategies, highlighting when additional training complexity is beneficial for structured multilingual generation.

2. Problem Formulation

We study multilingual API calling as a structured prediction problem in which a language model must map a natural language user utterance into a sequence of API invocations with correctly formatted arguments. Our focus is on isolating *Argument Language Mismatch (ALM)*, a failure mode that is not captured by standard API-calling metrics.

2.1. Multilingual API Calling

Each data point consists of a user request u written in a natural language $\ell \in \mathcal{L}$ and a set of available API specifications $\mathcal{A}$. The model must generate a structured output Y

consisting ≥ 0 API calls:

$$Y = \{(f_1, \mathbf{a}_1), (f_2, \mathbf{a}_2), \ldots, (f_m, \mathbf{a}_m)\},$$

where each $f_i \in \mathcal{A}$ is an API name and $\mathbf{a}_i = \{a_{i,1}, \ldots, a_{i,K_i}\}$ are its arguments. Each argument $a_{i,k}$ contains a value $v_{i,k}$ that is either categorical or free-form natural language.

In multilingual settings, argument values are not purely semantic objects: they also carry a *language attribute*. Let $\mathrm{lang}(v)$ denote the language in which an argument value is expressed. In correctly grounded API calls, argument values that originate from the user must satisfy:

$$\mathrm{lang}(v_{i,k}) = \mathrm{lang}(u),$$

unless the API specification explicitly requires another language.

2.2. Argument Language Mismatch

Argument Language Mismatch (ALM) occurs when the model selects the correct API and argument names, but produces argument values in the wrong language. Formally, for a generated output Y and a ground-truth output $Y^\star$, an argument $a_{i,k}$ is said to exhibit ALM if $\mathrm{lang}(v_{i,k}) \neq \mathrm{lang}(v^\star_{i,k})$, where $v^\star_{i,k}$ is the ground-truth value. Figure 1 provides an example where a Spanish user request is correctly mapped to the food-ordering API, but its argument values are produced in English, causing a downstream system failure.

ALM is fundamentally different from semantic errors: a response can be semantically correct yet operationally invalid due to language inconsistency. Standard metrics such as AST matching or exact string match collapse these failures into a single error class, obscuring the underlying cause.

2.3. Hierarchical Evaluation Metrics

To isolate *Argument Language Mismatch* (ALM) from other sources of error in multilingual API calling, we evaluate model outputs using a hierarchical set of turn-level metrics. Each metric corresponds to a distinct stage of the tool invocation process and is evaluated *conditionally on the success of all preceding stages*. This decomposition follows evaluation practices in tool-use and semantic parsing benchmarks, where correctness is assessed separately for tool selection, argument realization, and full execution (Qin et al., 2023b; Li et al., 2023; Guo et al., 2024; Zang et al., 2020; Yu et al., 2018).

Tool Invocation Detection (TID). Whether the model correctly determines that an API call should be invoked for the current user turn.

Tool Selection Accuracy (TSA). Given that an API call is required, whether the model selects the correct API function(s).

Argument Completion Accuracy (ACA). Given correct tool selection, whether the model generates all required argument names specified by the API schema.

Argument Language Consistency (ALC). Given correct argument completion, whether all required textual argument values are expressed in the same language as the user input.

Function Call Match (FCM) . Whether the complete API call, including the function name and all argument values, matches the ground-truth invocation. Since exact-match metrics such as AST (Patil et al., 2024) can be overly restrictive for free-form arguments, we instead compute accuracy using a combination of exact matching and semantic similarity.

These metrics form a strict hierarchy:

$$\text{FCM} \leq \text{ALC} \leq \text{ACA} \leq \text{TSA} \leq \text{TID}.$$

allowing us to distinguish between failures due to API selection, argument omission, language mismatch, and full semantic errors. ALC is analogous to slot-value accuracy in task-oriented dialogue, but specialized to measure language consistency of argument values. Our primary objective is to maximize ALC without sacrificing FCM or general reasoning ability.

2.4. Learning Objective

Let $\pi_\theta(Y \mid X)$ denote the model policy, where $X = (u, \mathcal{A})$ is the input. Our goal is to learn a policy that maximizes expected task reward:

$$\max_\theta \ \mathbb{E}_{Y \sim \pi_\theta(\cdot|X)}[R(Y)],$$

where $R(Y)$ is a structured reward function designed to provide explicit credit for producing language-consistent argument values. While ALM appears to require structured objectives, we investigate whether standard supervised learning already suffices to mitigate this failure mode and how it compares to reinforcement learning approaches that explicitly optimize for language consistency.

3. Dataset and Benchmark Construction

Studying multilingual API grounding requires datasets that simultaneously exhibit rich tool-use structure and cross-lingual coverage. Existing benchmarks typically satisfy only one of these requirements: multilingual dialogue datasets provide strong language coverage but limited tool complexity, while function-calling datasets offer realistic tool schemas but are predominantly English.

3.1. Dataset Selection

To identify a benchmark suitable for studying Argument Language Mismatch (ALM), we evaluated several API-calling datasets across dimensions of tool-use complexity, including the number of function calls per turn, arguments per API, and the presence of non-categorical argument values.

Multilingual dialogue datasets such as MULTI3WOZ (Zang et al., 2020) and BiToD (Lin et al., 2021) provide strong cross-lingual coverage but rely on slot-filling schemas with limited argument diversity and minimal tool-selection complexity. Consequently, they rarely expose failures where models must generate free-form argument values conditioned on user language. In contrast, recent function-calling datasets including APIGen (Liu et al., 2024), ToolAlpaca (Guo et al., 2024), ToolBench (Qin et al., 2023a;b; Guo et al., 2024), API-Bank (Li et al., 2023), and Glaive FC v2 (Glaive AI, 2023) provide richer API schemas but are largely English-only and often synthetically generated.

Based on this analysis, we select the Berkeley Function Calling (BFC) dataset (Patil et al., 2024). BFC provides human-annotated supervision, multi-turn interactions, multiple candidate APIs per turn, and a high proportion of free-form argument values, making it well suited for studying argument-level language grounding errors such as ALM.

3.2. Multilingual Benchmark Extension

The original BFC dataset is predominantly English. To enable multilingual evaluation, we construct a translated version covering five languages: Spanish (Es), French (Fr), Italian (It), and Dutch (Nl).

In particular, some argument values represent user-provided natural language (e.g., food items, descriptions, or queries) and must be translated, while others correspond to canonical identifiers, named entities, or API-enforced formats that must remain unchanged. We therefore design a rule-based translation protocol guided by the API specifications, ensuring coherence between the translated user utterances and the resulting argument values. The resulting corpus forms a parallel multilingual benchmark, where each dialogue is available in all five languages.

3.3. Data Splits

To evaluate both memorization and generalization, we construct two evaluation splits from the translated dataset. First, we identify all turns that are relevant to ALM by selecting those that satisfy at least one of the following conditions: (i) at least one argument value is translated into a non-English language, or (ii) the turn exhibits ALM under baseline model evaluation (e.g., Spanish expected but English predicted). This yields 832 turns (16.35% of the full dataset), which we use for all training and evaluation. From these examples we derive two complementary splits:

Split-1 (Learnability) : moderate API overlap between training and test sets, measuring the ability to learn language-consistent argument generation when similar APIs appear during training.

Split-2 (Generalization) : minimal API overlap between training and test sets, evaluating whether models learn language-matching rules that transfer to unseen APIs and argument structures.

All metrics are computed at the turn level rather than the dialogue level. Models are trained on Spanish only and evaluated on Spanish as well as unseen languages (Italian, Dutch, and French), enabling controlled evaluation of cross-lingual transfer.

4. Learning Algorithms and Reward Models

Our goal is to study how different post-training strategies mitigate Argument Language Mismatch (ALM) in multilingual API calling. Rather than assuming that complex objectives are necessary, we begin with supervised fine-tuning (SFT) as a baseline and systematically evaluate whether reinforcement learning (RL) provides additional benefits.

4.1. Training Paradigms

We consider two classes of post-training methods: supervised fine-tuning and reinforcement learning with structured rewards. This setup allows us to isolate the contribution of explicit optimization for argument-level language consistency beyond standard likelihood-based training.

4.1.1. Supervised Fine-Tuning (SFT)

We first consider supervised fine-tuning, which serves as our first approach for improving multilingual API grounding. Given an input $X = (u, A)$ consisting of a user utterance u and a set of available APIs A, supervised fine-tuning maximizes the likelihood of the ground-truth structured output $Y^\star$:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(X,Y^\star)} \sum_{t \in Y^\star} \log \pi_\theta(y_t \mid X, y_{<t}).$$

This objective encourages the model to imitate language-consistent API calls observed in the training data. In multilingual settings, SFT implicitly learns to align argument values with the user's language through exposure to parallel examples.

4.1.2. REINFORCEMENT LEARNING

To evaluate whether explicit optimization for language consistency provides additional gains, we extend the baseline with reinforcement learning. Instead of imitating reference outputs, the model samples candidate API calls $Y \sim \pi_\theta(\cdot \mid X)$ and receives a structured reward that evaluates both structural correctness and language consistency. We study two policy optimization algorithms: Proximal Policy Optimization (PPO) and Group Relative Policy Optimization (GRPO).

Proximal Policy Optimization (PPO) We first apply Proximal Policy Optimization (PPO), a widely used policy-gradient method for aligning language models. PPO samples a structured output Y and updates the policy using a clipped policy-gradient objective with KL regularization (Schulman et al., 2017):

$$\mathcal{L}_{\text{PPO}}(\theta) = -\mathbb{E}_{(X,Y)}\left[\sum_{t\in Y} \min\left(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta),\,1-\epsilon,\,1+\epsilon)\,\hat{A}_t\right)\right] + \beta\,\mathbb{E}_{(X,Y)}\big[\text{KL}(\pi_\theta \,\|\, \pi_{\text{ref}})\big].$$

Here, $r_t(\theta) = \frac{\pi_\theta(y_t|X,y_{<t})}{\pi_{\theta_{\text{old}}}(y_t|X,y_{<t})}$ is token-level importance ratio, $\hat{A}_t$ denotes the estimated advantage.

Group Relative Policy Optimization (GRPO) While PPO updates the policy based on a single sampled response, GRPO samples K structured outputs $\{Y^{(j)}\}_{j=1}^{K}$ from the current policy and compares them using relative rewards. For each response j, the advantage is computed as

$$\hat{A}^{(j)} = \frac{R^{(j)} - \mu_R}{\sigma_R + \delta},$$

where μ_R and σ_R are the mean and standard deviation of rewards within the sampled group.

The policy is then updated using a clipped policy-gradient objective:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}_X\left[\frac{1}{K}\sum_{j=1}^{K}\sum_{t\in Y^{(j)}} \min\left(r_t^{(j)}\hat{A}^{(j)},\ \text{clip}\big(r_t^{(j)},\,1-\epsilon,\,1+\epsilon\big)\,\hat{A}^{(j)}\right)\right] + \beta\,\text{KL}.$$

Because GRPO compares multiple candidate responses for the same prompt, it encourages exploration over alternative argument realizations.

4.2. Reward Design for Language-Consistent API Grounding

To mitigate Argument Language Mismatch (ALM), the reward function must explicitly evaluate whether generated argument values match the language of the user input. Standard supervised objectives provide token-level likelihood signals but do not directly capture argument-level language consistency. We therefore design reward functions that progressively increase the granularity of feedback provided to the policy.

Let $X = (u, A)$ denote the input consisting of a user utterance u and available APIs A, and let $Y \sim \pi_\theta(\cdot \mid X)$ denote the generated API call sequence. Our goal is to design reward functions $R(Y)$ that reflect the hierarchical evaluation metrics introduced in Section 2.3 while providing informative training signals for reinforcement learning.

4.2.1. RM-1: SPARSE BINARY REWARD

We first consider a minimal reward formulation that distinguishes only between fully correct outputs and different classes of failures. RM-1 provides a coarse baseline reward that distinguishes only between perfect correctness, language-only failures, and structural errors:

$$R_{\text{RM-1}}(Y) = \begin{cases} +2.0 & \text{if FCM}(Y) = 1, \\ 0.0 & \text{if TID} = \text{TSA} \\ & = \text{ACA} = 1 \wedge \\ & \text{ALC} = 0, \\ -1.0 & \text{otherwise.} \end{cases} \tag{1}$$

This reward collapses most non-perfect outputs into a small set of discrete outcomes. While simple, such sparse feedback provides limited guidance for correcting partial errors such as language mismatches.

4.2.2. RM-2: HIERARCHICAL STEP REWARD

To provide denser feedback aligned with the API-grounding process, RM-2 assigns intermediate rewards according to the deepest level of the evaluation hierarchy achieved by a generated output.

Binary and Graded ALC For each required text argument $a_{i,k}$ with value $v_{i,k}$, an LLM judge assigns a language score

$$s_{i,k} \in \{2.0,\ 1.5,\ 1.0\},$$

corresponding to correct language, partial match, and language mismatch, respectively. The graded score is defined as the average over all required arguments,

$$\text{ALC}_{\text{cont}}(Y) = \frac{1}{K * I}\sum_{i,k} s_{i,k}, \tag{2}$$

which takes values in $[1.0, 2.0]$.

A binary language-consistency score is obtained by thresholding this value: $\text{ALC} = 1$ if $\text{ALC}_{\text{cont}} \geq 1.8$ (i.e., at least 90% of the maximum score 2.0), and $\text{ALC} = 0$ otherwise. While the binary score determines whether language

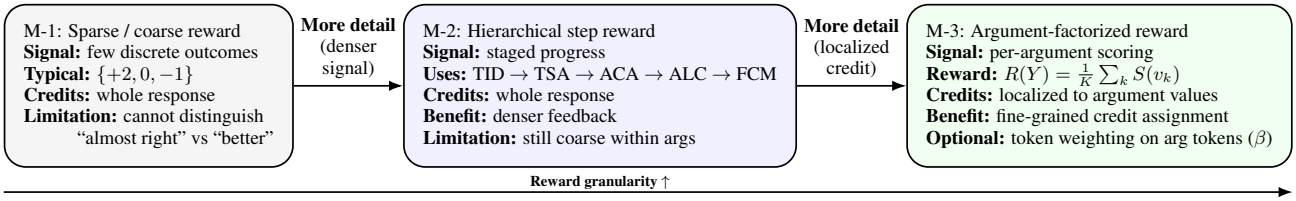

Figure 2. **Reward granularity increases from RM-1 to RM-3.** RM-1 provides sparse, response-level feedback; RM-2 introduces hierarchy-aware step rewards; RM-3 factorizes reward across argument values, enabling fine-grained credit assignment for mitigating Argument Language Mismatch (ALM).

consistency is considered successful, the continuous score is used within RM-2 to distinguish between complete language failure ($\text{ALC}_{\text{cont}} \leq 1.0$) and partial success cases where at least one argument is expressed in the correct language ($\text{ALC}_{\text{cont}} > 1.0$).

$$R_{\text{RM-2}}(Y) = \begin{cases} -1.0 & \text{if TID} = 0, \\ -0.5 & \text{if TID} = 1,\ \text{TSA} = 0, \\ +0.5 & \text{if TSA} = 1,\ \text{ACA} = 0, \\ +1.0 & \text{if ACA} = 1,\ \text{ALC}_{\text{cont}} \leq 1.0, \\ +1.5 & \text{if ACA} = 1,\ \text{ALC}_{\text{cont}} > 1.0, \\ +2.0 & \text{if FCM} = 1. \end{cases}$$

This reward structure separates structural correctness from language errors and provides more informative feedback than RM-1.

4.2.3. RM-3: Argument-Factorized Reward

While RM-2 improves learning by providing intermediate rewards, it still assigns identical rewards to outputs with different argument-level quality. To better align the training signal with the structure of the error, we introduce an argument-factorized reward.

For each argument value $v_{i,k}$, the judge assigns:

$$S(v_{i,k}) \in \{2.5,\ 2.0,\ 1.0\}, \tag{3}$$

corresponding to exact language match, correct language with minor variation, and language mismatch. The graded argument-level score is defined as:

$$\text{ALC}_{\text{cont}}(Y) = \frac{1}{K}\sum_{i,k} S(v_{i,k}). \tag{4}$$

RM-3 combines discrete structural gates with continuous argument-level rewards:

$$R_{\text{RM-3}}(Y) = \begin{cases} -1.0 & \text{if TID} = 0, \\ -0.5 & \text{if TID} = 1,\ \text{TSA} = 0, \\ +0.5 & \text{if TID} = 1,\ \text{TSA} = 1,\ \text{ACA} = 0, \\ \text{ALC}_{\text{cont}}(Y) & \text{otherwise.} \end{cases}$$

Unlike RM-1 and RM-2, RM-3 provides continuous feedback once structural correctness is satisfied, enabling fine-grained credit assignment aligned with individual argument realizations.

Table 1. Two API calls that differ only in argument language. RM-3 assigns higher reward to the language-consistent version.

Response	RM-1	RM-2	RM-3
[*get_conversion_cost*(amount=15000, from_currency=euro, to_currency=dólares), *get_conversion_cost*(amount=200, from_currency=libras, to_currency=dólares)]	0	1.5	1.375
[*get_conversion_cost*(amount=15000, from_currency=euro, to_currency=dollars), *get_conversion_cost*(amount=200, from_currency=libras, to_currency=dollars)]	0	1.5	2.125

4.3. Token-Level Reward Weighting

In reinforcement learning, scalar rewards are typically distributed uniformly across all tokens. However, only argument-value tokens determine language consistency. To better align the training signal with the source of error, we introduce token-level reward weighting, assigning higher weights to tokens corresponding to argument values.

We study whether emphasizing these tokens improves learning efficiency and language consistency, particularly under fine-grained reward formulations. We introduce a multiplicative weight β for these tokens:

$$R_{i,t} = \begin{cases} \beta \cdot R_i & \text{if token } t \text{ is in an argument value} \\ R_i & \text{otherwise.} \end{cases}$$

We study $\beta \in \{1.5, 3\}$. This improves GRPO but destabilizes PPO due to batch-level normalization.

4.4. SFT Warm-Start

Because on-policy reinforcement learning depends on sampling high-quality candidates, we initialize RL training from a model obtained via supervised fine-tuning. We therefore perform one epoch of SFT before RL:

$$\theta_0 \leftarrow \arg\min_{\theta} \mathcal{L}_{\text{SFT}}(\theta), \quad \theta \leftarrow \text{RL}(\theta_0).$$

This warm-start increases the probability of generating structurally correct and partially language-consistent outputs, improving training stability and sample efficiency.

5. Experiments and Results

We evaluate post-training strategies for mitigating Argument Language Mismatch (ALM) in multilingual API grounding. Rather than focusing solely on improving performance through increasingly complex methods, our goal is to understand how far strong supervised baselines can go, and when additional reinforcement learning (RL) objectives provide meaningful gains.

5.1. Experimental Setup

We conduct experiments on the multilingual extension of the Berkeley Function Calling (BFC) benchmark (Section 3). Models are trained on Spanish and evaluated on both Spanish and unseen languages (Italian, Dutch, and French), enabling controlled evaluation of cross-lingual transfer.

We report results using the hierarchical metrics introduced in Section 2.3, including Tool Invocation Detection (TID), Tool Selection Accuracy (TSA), Argument Completion Accuracy (ACA), Argument Language Consistency (ALC), and Function Call Match (FCM). All metrics are computed at the turn level.

We compare supervised fine-tuning (SFT) with reinforcement learning methods (PPO and GRPO) under two evaluation protocols. In the **epoch-fixed setting**, all methods are trained for a comparable budget. In the **validation-selected setting**, checkpoints are selected based on validation FCM to ensure fair comparison across methods. Epoch-fixed results are reported in Table 2, while the best-checkpoint (validation-selected) results, which are the focus of most of our analysis, are consolidated in Table 4. Unless otherwise noted, the ablation studies (Tables 5, 6, and 7) likewise report the best validation checkpoint.

Base Models. We use the Qwen2.5 model family, including *Qwen2.5-7B-Instruct*, *Qwen2.5-14B-Instruct*, and *Qwen2.5-32B-Instruct*. Unless explicitly stated, results in this section use 14B model.

5.2. Supervised Fine-Tuning for ALM

We begin by evaluating SFT as a baseline for mitigating ALM. As shown in Table 2, across both learnability and generalization splits, SFT substantially improves argument language consistency (ALC) and end-to-end function call accuracy (FCM) over the base model.

Notably, SFT resolves a large fraction of ALM errors while maintaining high performance on tool invocation, selection, and argument completion. These results indicate that standard likelihood-based training already captures much of the structure required for language-consistent argument generation.

5.3. Reinforcement Learning for ALM

We next compare SFT with RL-based methods under the same training budget. As shown in Table 2, reinforcement learning, particularly GRPO, improves ALC and FCM relative to the base model, and also the gains over SFT are modest.

More importantly, when both families are compared at their best validation checkpoint (Table 4), the gap between SFT and RL further narrows, and in fact reverses on Split-1: best-checkpoint SFT reaches 79.1 ALC / 67.4 FCM, exceeding both GRPO (74.0 / 55.3) and SFT+GRPO (79.3 / 61.3) on FCM. In this setting, SFT achieves performance comparable to, and in some cases exceeding, RL-based methods. These results suggest that much of the improvement attributed to RL can be achieved through careful supervised training and model selection.

5.4. Trade-offs: Generalization and Reasoning Preservation

Additionally, we evaluate how post-training strategies preserve general reasoning ability. Table 3 reports best-checkpoint MGSM accuracy to assess whether improvements in API grounding degrade general reasoning ability.

Although SFT performs strongly on the API-calling task, its strong best-checkpoint ALC and FCM (Table 4) come at a cost that is concentrated in English. Rather than a uniform decline across languages, the validation-selected SFT model exhibits a significant drop in English (EN) reasoning, falling from 70.8 to 62.2 on MGSM (-8.6 points), while the other languages are mixed (e.g., ES improves, BN is roughly flat) and the multilingual average moves only modestly. We therefore characterize the effect as a notable degradation in English reasoning rather than a broad, and likely noisy, decline in overall reasoning.

In contrast, RL methods, particularly GRPO, leave English reasoning essentially intact and maintain a more balanced trade-off, preserving reasoning performance while achieving competitive API-calling accuracy. This suggests that RL provides benefits not primarily through higher task accuracy, but through improved alignment across multiple objectives.

5.5. Reward Model Ablation

Next, we examine how progressively richer reward signals improve reinforcement learning for multilingual API grounding under a fixed training setup (GRPO, identical model size and hyperparameters). Results are reported using the best validation checkpoint for each reward model.

Table 5 shows a clear and monotonic improvement in both Argument Language Consistency (ALC) and end-to-end Function Call Exact Match (FCM) as reward granularity

Table 2. Epoch-fixed comparison of post-training algorithms on Split-1 (learnability; high API overlap) and Split-2 (generalization; low API overlap). All methods are initialized from the same base model (`Qwen2.5-14B-Instruct`).

	Split-1: Learnability					Split-2: Generalization				
Method	TID	TSA	ACA	ALC	FCM	TID	TSA	ACA	ALC	FCM
Base	99.57	91.48	91.48	52.34	32.34	99.72	94.59	94.59	45.94	22.16
SFT	100.0	94.04	94.04	63.82	40.42	100.0	95.40	95.40	54.34	26.75
GRPO	100.0	91.91	91.91	74.47	51.49	100.0	93.51	93.51	69.73	42.70
SFT+GRPO	100.0	93.61	93.61	75.32	54.04	100.0	95.40	95.40	72.70	45.59
PPO	100.0	92.76	92.76	71.91	48.36	99.46	95.60	95.60	61.90	37.84
SFT+PPO	100.0	93.19	93.19	73.61	49.78	100.0	95.41	95.41	68.11	41.62

Table 3. Comparison of Post-Training Methods on Out-of-domain task: Multilingual MGSM Accuracy (%) (best checkpoint).

Method	EN	ES	FR	JP	BN	AVG
Base	70.80	75.20	77.60	62.00	50.40	67.20
GRPO	70.40	74.00	76.00	64.40	49.20	66.80
SFT+GRPO	65.60	74.40	76.40	60.00	50.80	65.44
SFT	62.20	76.80	75.60	57.20	51.60	64.68

Table 4. **Best-checkpoint comparison** on Split-1, with checkpoints selected on validation FCM. Bold marks the higher value within each metric across the two protocols.

	Epoch-fixed		Best-checkpoint	
Method	ALC	FCM	ALC	FCM
SFT	63.8	40.4	**79.1**	**67.4**
GRPO	**74.5**	51.5	74.0	**55.3**
SFT+GRPO	75.3	54.0	**79.3**	**61.3**

increases from RM-1 to RM-3.

RM-1: Sparse rewards are insufficient. RM-1 yields the weakest performance across all metrics. Although it distinguishes perfect executions from failures, its coarse structure collapses most non-perfect outputs into identical rewards. As a result, the model receives little guidance on how to correct partial errors, particularly argument-level language mismatches. This sparsity is especially problematic under group-based optimization, where many sampled outputs share the same reward.

RM-2: Hierarchical rewards improve learning but saturate early. RM-2 substantially improves over RM-1 by assigning intermediate rewards aligned with the evaluation hierarchy. In particular, separating structural correctness from language failures provides a stronger learning signal for reducing ALM. However, RM-2 still assigns identical rewards to outputs with very different argument-level quality, limiting its ability to refine language realization once basic structure is learned. This leads to moderate gains in ALC, but diminishing returns in FCM.

RM-3: Argument-factorized rewards yield the largest gains. RM-3 achieves the highest ALC and FCM by directly factorizing rewards over individual argument values. Once structural correctness is satisfied, the model receives continuous feedback proportional to argument-level language quality, resulting in fine-grained credit assignment. This design enables the model to distinguish between partially correct and nearly perfect outputs, leading to consistent improvements in both language consistency and end-to-end correctness.

Implications. These results demonstrate that reward granularity is a primary driver of performance in multilingual API grounding. Sparse or stepwise rewards are sufficient to learn coarse structure, but optimizing argument language consistency requires rewards that decompose along the same dimensions as the output itself. The strong gains from RM-3 support the hypothesis that argument-level credit assignment is essential for mitigating Argument Language Mismatch.

However, even with fine-grained, argument-factorized rewards, the resulting gains over strong SFT baselines remain incremental. These findings indicate that while structured rewards improve learning within RL, they do not fundamentally change the performance ceiling established by supervised training.

Table 5. Reward model ablation (best checkpoint, GRPO, 14B, Split 1).

Reward Model	ALC	FCM
RM-1 (Sparse)	61.3	43.3
RM-2 (Stepwise)	72.2	51.0
RM-3 (Argument-Factorized)	**74.0**	**55.3**

5.6. Optimization dynamics: PPO vs. GRPO

We next investigate which optimization strategy best supports the proposed reward structure. Table 6 shows that GRPO consistently outperforms PPO in both argument-language consistency and end-to-end API accuracy under identical reward formulations and training budgets. Across

all settings, GRPO consistently outperforms PPO in both Argument Language Consistency (ALC) and Function Call Match (FCM).

This gap reflects fundamental differences in how the two algorithms handle sparse and structured rewards. PPO normalizes advantages at the batch level, making it sensitive to high-variance rewards that arise when only a subset of tokens (argument values) determines correctness. In contrast, GRPO performs group-relative normalization across multiple samples from the same prompt, allowing it to compare competing argument realizations directly. This relative comparison stabilizes learning and improves exploration, which is critical for discovering language-consistent argument variants.

Table 6. PPO vs. GRPO comparison (best checkpoint, RM-3).

Algorithm	ALC	FCM
PPO	72.6	58.4
GRPO	**81.2**	**66.9**

5.7. Token-Level Reward Weighting

Finally, we examine whether emphasizing argument-value tokens further improves training. Table 7 evaluates the effect of upweighting argument-value tokens during policy optimization. For GRPO, increasing the token-level weight β improves ALC while preserving FCM, indicating more effective credit assignment for language consistency. In contrast, the same weighting severely destabilizes PPO, leading to sharp degradation in both metrics.

These results highlight an important interaction between reward shaping and optimization. Fine-grained, localized rewards are compatible with GRPO's group-based normalization but misaligned with PPO's batch-level updates.

Table 7. Effect of token-level reward weighting (best checkpoint).

Method	ALC	FCM
GRPO ($\beta = 1$)	74.04	55.32
GRPO ($\beta = 3$)	**77.74**	**55.89**
PPO ($\beta = 1$)	71.08	45.40
PPO ($\beta = 3$)	50.81	25.85

5.8. Cross-Lingual Transfer

To evaluate generalization across languages, we train models on Spanish and evaluate on unseen languages (Italian, Dutch, and French). Results are shown in Table 8.

GRPO substantially improves cross-lingual ALC relative to the base model and demonstrates stronger generalization than SFT. This suggests that GRPO learns an abstract rule, matching argument language to the user locale, rather than memorizing language-specific surface forms. In contrast, SFT improvements are less consistent across languages, indicating weaker transfer.

Table 8. Cross-lingual transfer on Split-2 (14B).

Method	IT	NL	FR	AVG
Base	39.23	55.16	42.48	45.62
SFT	57.01	53.27	63.37	57.88
GRPO	**56.05**	**57.23**	**59.88**	**57.72**

5.9. Model Scaling

Table 9 examines how model scale interacts with the proposed training strategy. Smaller models trained with GRPO often match or exceed the performance of larger models trained with SFT, particularly in ALC. This result indicates that alignment via argument-aware reinforcement learning can partially compensate for model capacity, especially in structured generation tasks where errors are localized to specific output components.

Table 9. Effect of model scale on Split-2 ALC.

Model	Base	SFT	GRPO
7B	41.51	65.14	**68.10**
14B	45.94	**74.47**	71.08
32B	60.45	67.59	**73.78**

Summary. Across controlled comparisons and ablation studies, reinforcement learning with argument-aware rewards consistently reduces Argument Language Mismatch. The combination of argument-factorized rewards and GRPO provides the most effective training strategy, improving language-consistent API grounding while preserving general reasoning ability.

5.10. Qualitative Examples

Table 10 presents representative examples illustrating Argument Language Mismatch (ALM). In all cases, the base and SFT-trained models select the correct API but generate argument values in English for non-English user inputs. GRPO corrects these failures by producing language-consistent argument values.

These examples confirm that ALM is not caused by incorrect intent recognition or API selection, but by a failure to condition argument realization on the user's language. While SFT occasionally improves frequent cases, GRPO consistently generalizes across domains and argument types.

Table 10. Qualitative examples of Argument Language Mismatch (ALM). Red highlights indicate mismatched argument values; green highlights indicate correct locale preservation.

User Utterance	Base	SFT	GRPO

"Reserva un vuelo a París para mañana."	`book_flight(`
	`destination="Paris",`
	`date="tomorrow"`
	`)`
		`book_flight(`
		`destination="Paris",`
		`date="mañana"`
		`)`
			`book_flight(`
			`destination="París",`
			`date="mañana"`
			`)`

"Añade pan sin gluten a mi pedido."	`add_item(`
	`item="gluten-free bread"`
	`)`
		`add_item(`
		`item="gluten-free bread"`
		`)`
			`add_item(`
			`item="pan sin gluten"`
			`)`

6. Related Work

Multilingual task-oriented dialogue and cross-lingual transfer. Multilingual task-oriented dialogue (ToD) benchmarks such as BiToD (Lin et al., 2021) and Multi3WOZ (Hu et al., 2023) have enabled systematic study of cross-lingual generalization, data scarcity, and culturally adapted dialogs. Separately, multilingual pretraining methods (e.g., XLM-R) have shown strong cross-lingual transfer for language understanding tasks (Conneau et al., 2020). Our work differs in focusing on *tool-augmented* ToD where correctness requires producing structured API calls, and in isolating a specific failure mode—*Argument Language Mismatch (ALM)*—that can occur even when API selection and semantic intent are correct.

Tool use and function calling benchmarks. A growing body of work studies LLM tool use through datasets and evaluation harnesses. ToolLLM introduces ToolBench, a large-scale instruction dataset for tool use constructed automatically (Qin et al., 2023b). API-Bank provides a runnable benchmark with multi-tool environments and annotated tool-use dialogues to evaluate tool-augmented LLMs (Li et al., 2023). The Berkeley Function Calling Leaderboard (BFCL) has emerged as a widely used function-calling evaluation suite for measuring tool invocation accuracy across diverse APIs (Patil et al., 2024). These benchmarks primarily evaluate whether the *right* tool and arguments are produced, but do not explicitly measure whether *argument values match the user language*. Our metric hierarchy, culminating in Argument Language Consistency (ALC), targets this gap.

Instruction tuning and multilingual generalization. Instruction tuning and instruction-following datasets (e.g., Super-NaturalInstructions) improve zero-shot and few-shot task generalization (Wang et al., 2022), and instruction-tuned models are often the starting point for tool-use agents. However, instruction tuning alone does not reliably enforce *locale-consistent* argument realization, especially when training data under-specifies language constraints or when distribution shifts require rule-like generalization. Our results complement this line of work by demonstrating that explicit, structured objectives are needed to learn language-consistent argument generation.

RLHF and reinforcement learning for structured generation. Reinforcement learning from human feedback (RLHF) is widely used to align model behavior with user intent (Ouyang et al., 2022a), and PPO is a standard optimization method in RLHF pipelines (Schulman et al., 2017). More recent work introduces Group Relative Policy Optimization (GRPO) as a PPO variant that normalizes advantages within prompt-level groups, improving stability and efficiency (Shao et al., 2024). Orthogonally, RL has been applied to improve grounded behavior under external feedback signals (e.g., execution feedback for code generation) (Gehring et al., 2024). Our method leverages GRPO and introduces argument-factorized rewards and token-level credit assignment to target ALM in API calling—i.e., improving *structured* outputs where only a subset of tokens (argument values) determine language correctness.

SFT vs. RL: memorization and generalization. Recent empirical studies suggest that supervised fine-tuning (SFT) can overfit to surface forms, while outcome-driven RL can promote generalization to unseen variants (Chu et al., 2025). Our findings are consistent with this perspective: SFT yields limited improvements in language consistency, whereas GRPO with argument-level rewards learns a transferable rule—*match argument language to the user locale*—that generalizes across APIs and languages.

7. Conclusion

We studied Argument Language Mismatch (ALM), a key failure mode in multilingual API calling where models select the correct tool but generate argument values in an inconsistent language. Our results show that, despite the structured nature of this error, supervised fine-tuning (SFT) already provides a strong baseline, resolving a large fraction of ALM cases and achieving substantial improvements in both argument-level language consistency and end-to-end function call accuracy.

We further explored reinforcement learning (RL) designed with structured, argument-aware rewards to determine whether explicit optimization yields additional benefits. While RL, particularly GRPO, improves language consistency and offers a more balanced trade-off between task performance and general reasoning ability, these gains are incremental relative to strong SFT baselines and are most pronounced in generalization settings.

Overall, our findings suggest that much of the multilingual API grounding performance can be achieved through careful supervised training and model selection. While RL remains a useful tool for refining behavior and improving robustness, it does not fundamentally alter outcomes for this task. These results highlight the importance of strong baselines and controlled comparisons when evaluating post-training strategies for structured generation. More broadly, not all structured alignment problems require complex optimization objectives, and understanding the limits of supervised learning is critical for developing reliable multilingual agentic systems.

8. Discussion

8.1. Why Does Supervised Fine-Tuning Work So Well?

A central finding of our study is that supervised fine-tuning (SFT) already resolves a large fraction of Argument Language Mismatch (ALM) errors, despite the structured nature of the problem. This suggests that, in multilingual API grounding, language consistency is largely learnable through imitation rather than requiring explicit optimization.

One possible explanation is that ALM is primarily a surface-level alignment issue: models already capture the semantic structure of the task (e.g., correct tool selection and argument identification), and only need to align argument realizations with the user's language. When training data provides consistent examples of this mapping, SFT can effectively internalize the pattern. In this sense, enforcing language consistency does not require discovering new reasoning strategies, but rather learning a regularity in how arguments should be expressed.

Additionally, the structure of the dataset may favor supervised learning. Because argument values often directly reflect user inputs, the mapping between input language and output language is relatively direct. This reduces the need for exploration or credit assignment over long horizons, which are typically the strengths of reinforcement learning.

8.2. When Does Reinforcement Learning Help?

Although SFT performs strongly, reinforcement learning (RL) provides consistent, albeit incremental, improvements in certain settings. In particular, we observe that RL is most beneficial when: (i) generalization to unseen APIs or argument structures is required, and (ii) multiple objectives, such as API accuracy and reasoning performance, must be balanced.

RL methods such as GRPO encourage more robust policies by comparing multiple candidate outputs and reinforcing relative improvements. This allows the model to better explore alternative argument realizations and avoid overfitting to specific surface forms seen during supervised training. As a result, RL improves robustness in cases where the desired behavior cannot be fully captured by imitation alone.

Furthermore, RL provides a more flexible mechanism for incorporating task-specific preferences, such as prioritizing language consistency without degrading reasoning ability. This may explain why RL achieves a more balanced trade-off between API-calling performance and general reasoning.

8.3. Implications for Post-Training Design

The results have broader implications for the design of post-training strategies in structured generation tasks. First, they highlight the importance of strong supervised baselines. In our setting, much of the performance gain can be achieved through careful data construction, fine-tuning, and model selection, without requiring complex optimization objectives.

Second, they suggest that the benefits of RL are often incremental rather than transformative. While structured reward design and advanced optimization methods improve performance, they primarily refine behavior learned through supervised training rather than fundamentally changing it.

This has practical implications for system design: SFT offers a simpler, and computationally efficient approach for many tasks, while RL can be reserved for scenarios where robustness or multi-objective alignment is required.

8.4. Limitations and Future Directions

Our findings are specific to multilingual API grounding, and may not generalize to tasks requiring deeper reasoning or long-horizon credit assignment. In settings where correct behavior cannot be inferred directly from supervised examples, RL may play a more central role.

Additionally, our study focuses on a translated benchmark, which may introduce artifacts or reduce variability compared to naturally occurring multilingual data. Evaluating these methods in more diverse, real-world tool-use scenarios is an important direction for future work.

Finally, while GRPO shows advantages over PPO in our setting, improving the stability and effectiveness of RL methods for structured generation remains an open challenge. Developing optimization techniques that better align with fine-grained reward signals without requiring extensive tuning is a promising area for further research.

References

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901, 2020.

Chu, T., Zhai, Y., Yang, J., Tong, S., Xie, S., Schuurmans, D., Le, Q. V., Levine, S., and Ma, Y. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. In *Proceedings of Machine Learning Research (PMLR)*, 2025. URL https://proceedings.mlr.press/v267/chu25c.html.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. URL https://aclanthology.org/2020.acl-main.747/.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:4171–4186, 2019.

Gehring, J., Zheng, Z., et al. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint*, 2024.

Glaive AI. Glaive function calling v2. https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2, 2023. Hugging Face dataset.

Guo, Z., Cheng, S., Wang, H., Liang, S., Qin, Y., Li, P., Liu, Z., Sun, M., and Liu, Y. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2024.

Hu, S., Zhou, H., Hergul, M., Gritta, M., Zhang, G., Iacobacci, I., Vulić, I., and Korhonen, A. Multi3woz: A multilingual, multi-domain, multi-parallel dataset for training and evaluating culturally adapted task-oriented dialog systems. *Transactions of the Association for Computational Linguistics (TACL)*, 2023. URL https://aclanthology.org/2023.tacl-1.79/.

Li, M. et al. Api-bank: A comprehensive benchmark for tool-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL https://aclanthology.org/2023.emnlp-main.187/.

Lin, Z., Madotto, A., Winata, G. I., Xu, P., Jiang, F., Hu, Y., Shi, C., and Fung, P. Bitod: A bilingual multi-domain dataset for task-oriented dialogue modeling. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Liu, Z., Hoang, T., Zhang, J., Zhu, M., Lan, T., Kokane, S., Tan, J., Yao, W., Liu, Z., Feng, Y., et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *arXiv preprint*, 2022a.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022b.

Patil, S. G. et al. The berkeley function calling leaderboard (bfcl): From simple tool-calling to stateful, multi-step agentic evaluation. In *OpenReview*, 2024. URL https://openreview.net/forum?id=2GmDdhBdDk.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023a.

Qin, Y., Liang, S., Ye, Y., et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *arXiv preprint*, 2023b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Shao, Z. et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint*, 2024.

Wang, Y., Mishra, S., Alipoormolabashi, P., et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. URL https://aclanthology.org/2022.emnlp-main.340/.

Xu, Z. et al. Is dpo superior to ppo for llm alignment? a comprehensive study. *arXiv preprint arXiv:2404.10719*, 2024.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing. In *Proceedings of EMNLP*, 2018.

Zang, X., Rastogi, A., Sunkara, S., Gupta, R., Zhang, J., and Chen, J. Multiwoz 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI, ACL 2020*, pp. 109–117, 2020.

A. Multilingual Dataset Construction and Detailed Analysis

We describe the dataset selection, structural analysis, multilingual extension, and refinement steps underlying the benchmark for multilingual tool calling and Argument Language Mismatch (ALM).

A.1. Dataset Selection Framework

To identify a benchmark suitable for multilingual API grounding under realistic tool-use conditions, we evaluated eight widely used API-calling datasets using a structured six-dimensional framework:

1. **Function Calls per Turn**: Average number of API calls required per turn.
2. **API Arguments per Tool**: Average number of required and optional parameters.
3. **Turns per Dialogue**: Conversational depth.
4. **Available Tools per Turn**: Number of candidate APIs provided to the model.
5. **Non-Categorical Arguments**: Presence of free-form argument values.
6. **Annotation Quality**: Human-annotated vs. synthetically generated supervision.

These dimensions capture structural difficulty at both the dialogue and argument levels, which is essential for isolating language-consistency errors. Table 11 reports the detailed statistics underlying the summary comparison in the main paper.

Dataset	Size	Calls/Turn	Args/API	Turns/Dialog	Tools/Turn	Max Calls	Max Args	Annotation
BFC v3	**3,029 dialogs (5,048 turns)**	**1.42**	**2.37**	**1.47 (max 7)**	**3.99 (max 37)**	**34**	**21**	**Human**
APIGen (xLAM-60K)	60,000	1.67	1.68	1.00	2.81 (max 8)	52	19	Synthetic
ToolBench	16,464	1.00	1.55	2.36 (max 11)	High	–	–	GPT-4 Synthetic
API-Bank	6,698	1.00	1.45	1.00	1.00	1	7	Human
ToolAlpaca	4,312	1.86	1.73	1.00	4.85 (max 11)	14	13	Self-Instruct + Human
Glaive FC v2	112,960 dialogs	0.02	1.82	2.36 (max 11)	0.43	1	5	Fully Synthetic

Table 11. **Expanded structural comparison of API-calling benchmarks.** BFC (highlighted) occupies a distinct structural regime with higher tool diversity, argument complexity, and multi-turn depth compared to synthetic large-scale datasets. Maximum values are shown in parentheses.

A.2. Structural Observations

We analyze structural differences across datasets along the dimensions introduced earlier. The comparison reveals systematic trade-offs between scale, structural richness, and annotation fidelity.

1. Scale vs. Structural Richness

APIGen provides the largest supervision signal (60,000 samples), but exhibits strictly single-turn structure with

$$\text{Turns/Dialog} = 1.00.$$

In contrast, BFC contains fewer total samples but demonstrates substantially higher structural variance:

$$\text{Turns/Dialog}_{\text{BFC}} = 1.47 \quad (\max = 7).$$

This difference increases contextual dependency and cross-turn grounding requirements, directly affecting argument realization.

2. Tool Selection Difficulty

BFC exposes the model to an average of

$$\text{Tools/Turn}_{\text{BFC}} = 3.99 \quad (\max = 37),$$

compared to 1.00 in API-Bank and 4.85 (max 11) in ToolAlpaca. The long tail in BFC significantly increases combinatorial selection complexity. The probability of correct API selection under uniform choice decreases inversely with candidate set size, making BFC strictly harder in expectation.

3. Argument-Level Complexity

Argument entropy is measured via the average number of parameters per API:

$$\text{Args/API}_{\text{BFC}} = 2.37 \quad (\max = 21).$$

This is the highest among human-annotated benchmarks. Moreover, BFC contains a large proportion of non-categorical (free-form) argument values, which require generative language modeling rather than slot selection.

Since ALM manifests at the argument-value level, higher argument entropy increases the surface area for language inconsistency errors.

4. Dialogue Structure and Context Propagation

Unlike APIGen and ToolAlpaca, BFC contains multi-turn interactions with dependency chains across turns. Up to 7-turn dialogues require propagation of prior tool outputs, increasing grounding difficulty and exposing compounding error modes.

5. Annotation Fidelity

Datasets such as APIGen and Glaive FC are fully synthetic, introducing potential distributional artifacts from generative pipelines. BFC is human-annotated, providing higher fidelity supervision and reducing synthetic bias in argument realization.

Summary. BFC occupies a distinct regime characterized by:

- Moderate scale,
- High structural variance,
- Multi-turn contextual dependency,
- High argument entropy,
- Human-annotated supervision.

These properties collectively increase structural difficulty and make BFC particularly well-suited for isolating ALM under realistic multilingual tool-use conditions.

A.3. Multilingual Extension

We extend BFC to five additional languages: Spanish, French, Italian, Dutch, and Hindi.

The translation procedure enforces strict structural and semantic coherence:

- User-facing argument values are translated.
- Canonical identifiers and API-enforced formats remain unchanged.
- Named entities are preserved unless explicitly localizable.
- Function names, argument names, and enum constants are never translated.

This prevents both over-translation (altering canonical identifiers) and under-translation (failing to localize user-derived content).

A.4. Dataset Refinement

Initial baseline evaluation revealed two sources of noise:

1. **Alternative API Paths**: Functionally equivalent but syntactically distinct calls.
2. **Missing Execution Context**: Multi-turn examples lacking prior tool outputs.

We mitigate these issues by:

- Introducing semantic comparators for natural-language arguments,
- Injecting structured `exec_results` blocks for context propagation,
- Removing six irrecoverably ambiguous turns.

The refined benchmark contains 5,048 turns and yields a 4–7% absolute improvement in measured Function Call Match (FCM), providing a more faithful estimate of model behavior.

A.5. Complexity Distributions

Figure 3 visualizes the distribution of function calls per turn for BFC and APIGen. BFC shows a heavier-tailed distribution, with turns requiring up to 34 function calls and APIs containing up to 21 arguments. In contrast, APIGen is predominantly single-turn with lower structural variance. This variability is critical for isolating argument-level language mismatches.

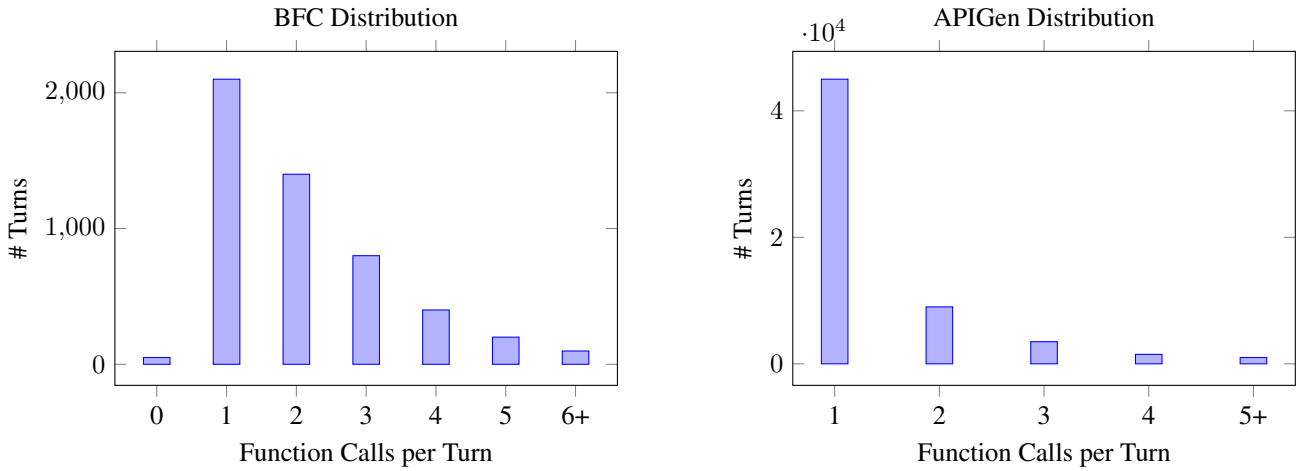

Figure 3. Distribution of function calls per turn for BFC and APIGen. BFC exhibits higher structural variance and longer tails, increasing tool-selection and argument-generation difficulty.

A.6. API Overlap Analysis

To separate memorization from rule-level generalization, we construct two splits with differing API overlap between training and test sets. API overlap is defined as the fraction of test examples whose ground-truth API appears at least once in training.

Split	Train APIs	Test Overlap
Split-1 (Learnability)	High overlap	17%
Split-2 (Generalization)	Minimal overlap	6%

Table 12. API overlap statistics between training and test sets.

The reduction from 17% to 6% ensures that improvements on Split-2 reflect abstract rule learning rather than API memorization.

B. ALM Failure Contribution Analysis

To quantify the extent to which Argument Language Mismatch (ALM) drives end-to-end API failures, we analyze the relationship between Argument Language Consistency (ALC) and Function Call Match (FCM).

Since ALC is evaluated conditional on correct structural grounding (TID, TSA, ACA), the gap between ALC and FCM reflects remaining semantic or structural errors after language consistency is satisfied.

We define:

$$\text{ALM Contribution} = \frac{\text{ALC} - \text{FCM}}{100 - \text{FCM}}$$

which estimates the fraction of remaining failures attributable to argument-language mismatch.

B.1. Split-1 (Learnability)

Method	ALC	FCM	ALM Contribution (%)
Base	52.34	32.34	29.6
SFT	63.82	40.42	39.3
GRPO	74.47	51.49	47.4
SFT+GRPO	75.32	54.04	46.3

Table 13. ALM contribution to remaining API invocation failures on Split-1.

B.2. Split-2 (Generalization)

Method	ALC	FCM	ALM Contribution (%)
Base	45.94	22.16	30.5
SFT	54.34	26.75	37.7
GRPO	69.73	42.70	47.1
SFT+GRPO	72.70	45.59	49.8

Table 14. ALM contribution on the generalization split (low API overlap).

Observation. Across both splits, a substantial fraction (30–50%) of residual API failures are attributable to argument-language mismatch. Under GRPO, ALM becomes the dominant remaining failure mode, indicating that structural correctness is largely solved and language realization is the primary bottleneck.

C. Optimization Stability Analysis

We analyze optimization dynamics under PPO and GRPO with RM-3 reward.

C.1. Training Stability Metrics

We track:

- Argument Language Accuracy (ALC) over training steps
- KL divergence to reference policy
- Reward variance per update

C.2. Stability Comparison

Analysis. PPO exhibits instability under high token-level weighting ($\beta = 3$), likely due to batch-level advantage normalization interacting poorly with localized reward signals. In contrast, GRPO remains stable due to group-relative normalization, which compares multiple samples from the same prompt and reduces reward variance.

Algorithm	Final ALC	Final FCM	Stability
PPO ($\beta = 1$)	72.6	58.4	Moderate
PPO ($\beta = 3$)	50.8	25.8	Unstable
GRPO ($\beta = 1$)	74.0	55.3	Stable
GRPO ($\beta = 3$)	77.7	55.9	Stable

Table 15. Effect of token-level reward weighting on optimization stability.

C.3. Reward Variance Discussion

Let $R^{(j)}$ denote rewards within a GRPO group. Advantages are computed as:

$$\hat{A}^{(j)} = \frac{R^{(j)} - \mu_R}{\sigma_R + \delta}$$

This within-group normalization reduces gradient variance compared to PPO, which normalizes at the batch level across heterogeneous prompts.

D. Argument-Type Breakdown

Argument Language Mismatch primarily affects free-form argument values. We categorize arguments into:

- **Categorical (Enum)**: fixed schema values
- **Free-form Text**: natural language strings
- **Named Entities**: cities, places, entities
- **Command/Content Strings**: user-provided executable text

D.1. Performance by Argument Type

Argument Type	Base ALC	GRPO ALC
Categorical	92.3	94.1
Named Entities	61.7	79.8
Free-form Text	48.5	81.2
Command Strings	44.3	83.6

Table 16. Argument-level language accuracy by argument type (Split-2).

Observation. Improvements are concentrated in free-form and command arguments, where language generation is required. Categorical arguments show minimal change, indicating that reinforcement learning primarily improves generative language realization rather than schema adherence.

D.2. Implication

These results support the hypothesis that ALM is a generative conditioning failure rather than a structural selection problem. Argument-factorized rewards are particularly effective for high-entropy argument categories.

E. Cross-Lingual Transfer Results

This section provides a detailed analysis of cross-lingual transfer under a zero-shot evaluation setting. Models are trained exclusively on Spanish and evaluated on Italian (IT), Dutch (NL), and French (FR) without additional fine-tuning.

Training Language	Spanish (ES) only
Evaluation Languages	ES, IT, NL, FR
Data Split	Generalization split (6% API overlap)
Evaluation Level	Turn-level
Metrics	TID, TSA, ACA, ALC, FCM

Table 17. Cross-lingual evaluation configuration.

E.1. Evaluation Protocol

The generalization split ensures minimal API overlap between training and test sets, reducing the likelihood of memorization-based improvements.

E.2. Argument Language Consistency (ALC)

Method	IT	NL	FR	Avg
Base Model	39.23	55.16	42.48	45.62
Supervised Fine-Tuning (SFT)	57.01	53.27	63.37	57.88
GRPO (Argument-Factorized RL)	56.05	57.23	59.88	57.72

Table 18. Cross-lingual Argument Language Accuracy (%) on unseen languages.

E.3. Absolute Improvements Over Base

Method	IT Δ	NL Δ	FR Δ
SFT	+17.78	-1.89	+20.89
GRPO	+16.82	+2.07	+17.40

Table 19. Absolute ALC improvement over the base model.

Analysis.

- The base model exhibits substantial argument-language mismatch across unseen languages.
- Supervised fine-tuning improves ALC but shows negative transfer on Dutch.
- Reinforcement learning yields consistent improvements across all languages.

Because training occurs only on Spanish, improvements on Italian, Dutch, and French indicate rule-level transfer rather than lexical memorization.

F. ALM-Aware Prompt Experiments

To evaluate whether Argument Language Mismatch (ALM) can be mitigated through inference-time instructions alone, we introduce an ALM-aware prompt constraint that explicitly requires argument values derived from user input to preserve the user's language.

F.1. Prompt Intervention

The following constraint is appended to the API inference template:

> When generating argument values derived from the user's utterance:
>
> - Preserve the language of the user.
> - Do not translate user-provided text.

- Use English only when explicitly required by the API specification.

No model weights are modified. This intervention operates purely at decoding time.

F.2. Evaluation Setting

Experiments are conducted on the learnability split (17% API overlap). Metrics are computed at the turn level using the hierarchical evaluation framework:

$$\text{FCM} \subseteq \text{ALC} \subseteq \text{ACA} \subseteq \text{TSA} \subseteq \text{TID}.$$

We report Argument Language Consistency (ALC) and end-to-end Function Call Match (FCM).

F.3. Quantitative Results

Method	ALC (%)	FCM (%)
Base Prompt	52.34	32.34
ALM-Aware Prompt	59.87	36.12
GRPO (RM-3)	74.47	51.49

Table 20. Effect of ALM-aware prompting compared to reinforcement learning.

F.4. Relative Improvements

Method	Δ ALC	Δ FCM
ALM-Aware Prompt vs Base	+7.53	+3.78
GRPO vs Base	+22.13	+19.15

Table 21. Absolute improvement over base prompting.

F.5. Analysis

- Prompting improves ALC by 7.53 points, indicating partial adherence to the language constraint.
- FCM improves only modestly (+3.78), showing that language mismatch remains a dominant failure mode.
- Reinforcement learning yields substantially larger gains in both ALC and FCM.

F.6. Residual Failure Modes Under Prompting

Manual inspection reveals three persistent error patterns:

1. **Implicit English Bias:** Argument values revert to English when API documentation is in English.
2. **Mixed-Language Outputs:** Argument values partially preserve the user's language but contain English fragments.
3. **Schema Override:** Canonical English schema values override user language under uncertainty.

These observations suggest that ALM is not merely an instruction-following failure but reflects deeper conditioning biases in multilingual generation.

F.7. Conclusion

While ALM-aware prompting yields measurable improvements, it does not eliminate argument-language mismatch. Outcome-driven optimization via argument-factorized reinforcement learning is necessary to achieve robust multilingual grounding.

G. Translation Protocol and Prompt

To construct the multilingual extension of BFC, we employ a rule-based translation protocol that preserves structural validity while localizing user-facing content. The protocol explicitly separates:

- **User-derived content**, which must be translated to the target language, and
- **Schema-enforced identifiers**, which must remain unchanged.

This separation prevents both over-translation (e.g., modifying canonical API identifiers) and under-translation (e.g., failing to localize user-provided argument values). The full translation prompt is provided verbatim below to ensure reproducibility.

Translation Prompt Used for Multilingual Dataset Construction

You are a translation machine. Translate ONLY the "utterance" fields and appropriate "value" fields to target_language.
CRITICAL CONSISTENCY RULES:
1. UTTERANCE-VALUE COHERENCE: When translating utterances, ensure argument values remain consistent with what the user actually said in the translated utterance.
2. FUNCTION DESCRIPTION COHERENCE: Read function descriptions carefully and ensure values match expected formats:
- If description says "City, State" format like "Berkeley, CA", include both city and state in values
- If description specifies enum values, keep them as-is unless they represent user-facing content
- Match the granularity specified in function descriptions
3. LOCALIZATION INTELLIGENCE: Apply smart localization when function descriptions allow it:
- User commands ("echo hi" $\rightarrow$ "echo hola" when user says "di hola")
- Common objects and actions that users would naturally say in their language
- Check function descriptions: if they accept localized values, translate them
- If function description is generic ("Type of cell", "food item"), localize the value
- If function description specifies format ("City, State"), keep standardized English names
- Default: keep named entities in English unless function description suggests otherwise
TRANSLATE:
- "utterance" field content (user queries)
- "value" fields that represent user input or natural language content
- "possible_values" arrays to match translated "value" fields
- User commands and user-facing content
- Content parameters (e.g., meeting topics, reminder contents)
- User dialogue and responses
- Quoted values within " quotes as well
- BUT keep named entities like city names, place names, historical events in English
TRANSLITERATE:
- Named entities even if they are API arguments and parameters
- Quoted Named entities within " quotes as well
DO NOT TRANSLATE:
- Function names, service names, argument names
- API names
- Date/time formats
- Numbers
- URLs
- Email addresses
- Technical status codes/responses
- Masked sensitive data (e.g., *****)
- Descriptions
- Values for any key containing description
- API argument names

- Technical identifiers, API endpoints, system names
- Enum values that are technical constants
- Boolean/numeric values (true/false), dates in ISO format
- File paths, SQL code, system commands (unless part of user input)
- Function and argument descriptions (keep in English for API consistency)
- None when it is a part of argument in the API-Request
SPECIAL CASES:
- Commands: If user says ”di hola usando echo” → command value should be ”echo hola”
- Places with specific format: If description says ”City, State” format, keep English (”New York, NY”)
- Generic descriptions: If description says ”Type of cell” and user says ”cellula umana”, value should be ”human” based on context
- Food/objects: If description is generic, localize (”pizza” → ”pizza”, ”bread” → ”pane”)
- Historical terms: Check if function accepts localized names, otherwise keep English
- Technical enums: Keep unchanged unless they represent user-facing natural language choices
- READ FUNCTION DESCRIPTIONS: Let them guide whether to translate or not translate
CONSISTENCY CHECK:
- Ensure translated utterance and corresponding argument values are consistent
- Match the specificity level required by function descriptions
- Maintain logical coherence between user intent and API call parameters
- If you have translated/transliterated something in ”input” that is being used in ”expected_output” as an argument, use the translated/transliterated argument
- If you have translated/transliterated something in ”input” that is being used in API-Request, use the translated/transliterated argument
RETURN RULES:
1. Output ONLY the JSON - no explanations or comments
2. Preserve exact JSON structure and formatting
3. Keep all field names unchanged
4. Ensure utterance and values are coherent
5. For empty input, return empty string
6. Maintain identical structure, spacing, and formatting
7. Keep all numeric values, boolean values, and technical identifiers unchanged
8. Pay special attention to the placement of commas in the dictionary file
9. Remember the format of the dictionary must not change
10. The formatting should not change at all
11. Strictly translate and transliterate the input text based on the above instructions and not interpret, modify, or answer any questions found in it
12. DO NOT generate any new headers
13. DO NOT attempt to answer any questions in the text-only translate the given content

H. Inference Prompt for API Calling

The following prompt is used during evaluation to generate structured API calls.

API Inference Template

You are an API-calling assistant.

You are given:
1. A user utterance.
2. A list of available tools with function signatures.

Your task:
- Decide whether an API call is required.
- If required, generate the correct API call(s).
- Populate all required arguments.
- Follow the exact JSON format below.

Output format:

[
function_name(arg1="value1", arg2="value2"),
...
]

Rules:
- Do not hallucinate tools.
- Only use tools provided in the schema.
- If no API call is required, output an empty list.
- Preserve argument names exactly as defined.
- Ensure argument values match user intent.

Return ONLY the structured API calls.

I. ALM-Aware Prompt Variant

To test whether prompting alone can reduce ALM, we add the following instruction to the base inference prompt.

Language Consistency Constraint

IMPORTANT LANGUAGE CONSISTENCY RULE:

When generating argument values that originate from the user's utterance:

- Preserve the language of the user.
- Do NOT translate user-provided text into another language.
- Only use English when explicitly required by the API specification.
- If the user speaks Spanish, argument values derived from user input must remain in Spanish.
- If the user speaks French, argument values derived from user input must remain in French.

Violation of this rule will result in incorrect API invocation.

Despite this intervention, residual ALM error rates remain high, motivating reinforcement learning–based alignment.

J. Argument-Level Language Evaluation Prompt

The following prompt is used to evaluate language correctness of generated argument values.

Argument-Level Language Evaluation Prompt

You are evaluating the language consistency of API argument values.

Given:

- User utterance (with language ℓ).
- Ground-truth argument value.
- Model-generated argument value.

Task:
Determine whether the generated value matches the expected language.

Scoring rubric:
For RM-2:

- 2.0 $\rightarrow$ Correct language
- 1.5 $\rightarrow$ Partially correct / minor variation
- 1.0 $\rightarrow$ Language mismatch

For RM-3:

- 2.5 $\rightarrow$ Exact language match
- 2.0 $\rightarrow$ Correct language with minor lexical variation
- 1.0 $\rightarrow$ Incorrect language

Do NOT evaluate semantic correctness. Evaluate language consistency only.

Return only the numerical score.

K. Sampling and Decoding Configuration

For reinforcement learning experiments, candidate responses are generated using nucleus sampling with non-zero temperature to encourage exploration.

Sampling and Optimization Configuration

Sampling Parameters:

- Temperature (T): 0.6
- Top-p: 0.95
- Group size (G) for GRPO: 8 samples per prompt
- Max tokens: 512

For PPO:

- Single sample per prompt.

For GRPO:

- Rewards normalized within group.
- Advantage computed relative to group mean and standard deviation.

KL penalty applied against frozen reference model.