

Tool Calling for Arabic LLMs: Data Strategies and Instruction Tuning

Asim Ersoy, Enes Altinisik, Husrev Taha Sencar, Kareem Darwish

Qatar Computing Research Institute, HBKU, Qatar

{aersoy,ealtinisik,hsencar,kadarwish}@hbku.edu.qa

Abstract

Tool calling is a critical capability that allows Large Language Models (LLMs) to interact with external systems, significantly expanding their utility. However, research and resources for tool calling are predominantly English-centric, leaving a gap in our understanding of how to enable this functionality for other languages, such as Arabic. This paper investigates three key research questions: (1) the necessity of in-language (Arabic) tool-calling data versus relying on cross-lingual transfer, (2) the effect of general-purpose instruction tuning on tool-calling performance, and (3) the value of fine-tuning on specific, high-priority tools. To address these questions, we conduct extensive experiments using base and post-trained variants of an open-weight Arabic LLM. To enable this study, we bridge the resource gap by translating and adapting two open-source tool-calling datasets into Arabic. Our findings provide crucial insights into the optimal strategies for developing robust tool-augmented agents for Arabic.

1 Introduction

Tool calling, frequently referred to as function calling, represents a pivotal feature that significantly extends the operational capabilities of Large Language Models (LLMs) and LLM-based agents. This functionality empowers an LLM to interact with external systems or applications by generating structured requests in response to a user's natural language prompt (Masterman et al., 2024), allowing the LLMs to perform tasks beyond their intrinsic capabilities. Typically, an LLM is provided with a prompt alongside a predefined set of tools (or functions), complete with their descriptions, arguments, and expected output. The LLM then analyzes the prompt to determine if invoking an external tool is necessary to fulfill a user's request. If a tool call is identified, the LLM generates a structured tool call request,

in accordance with what the tools expect. The output generated by the execution of the external tool is subsequently fed back to the LLM to be incorporated into the final response of the LLM, thereby creating a dynamic and iterative problem-solving loop (Masterman et al., 2024). Consequently, an LLM must be explicitly trained to understand tool descriptions, recognize when they are needed, generate structured function calls, and handle their output. Figure 1 shows an example addition function with its invocation.

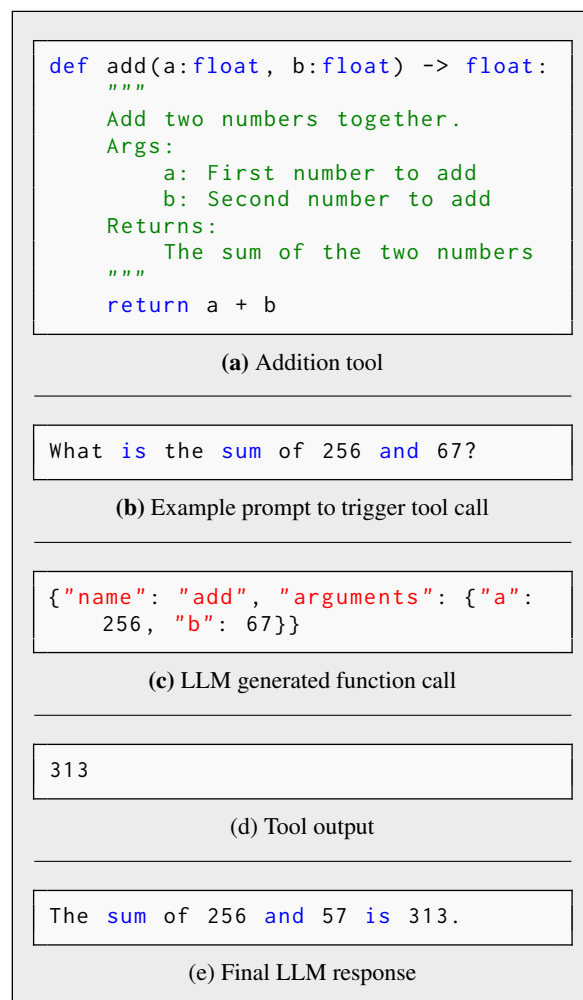


Figure 1: Example tool with invocation

Currently, there are quite a few tool-calling datasets, such as Glaive¹, xLAM (Liu et al., 2024b), ToolAce (Liu et al., 2024a), and Hermes², that provide tens of thousands of different tools with the intended interaction with them. The vast majority of tool-calling datasets are in English, with some that have been translated to other languages (e.g. Chinese Glaive³). However, given the cross-lingual generalization capabilities of LLMs, it is not clear how much impact non-English training data has on the tool-calling abilities of the LLMs. Further, though tool-calling training data demonstrate to an LLM all the required tool-calling steps, would tool-calling benefit from LLM supervised fine-tuning (SFT) on general-purpose tasks such as chat, summarization, or headline generation? Additionally, tool-calling training data can't cover all possible tools, and LLMs are expected to generalize to new tools. However, if a set of tools is important to a user or an organization, how much benefit would be observed if training samples for these specific calls are included in tool-calling training data?

This paper attempts to answer the three aforementioned research questions, namely:

1. When using tool-calling for non-English prompts (e.g., Arabic), do LLMs benefit from being fine-tuned on tool-calling datasets in that language?
2. What effect does post-training on general domain capability data have on the ability of LLMs to perform effective tool calling?
3. Though LLMs can generalize well beyond the examples in their training data, is there value for tool-specific fine-tuning?

The contributions of the paper are as follows:

- We conduct extensive experimentation on a public open-weights LLM, Fanar (Team et al., 2025) that is specifically trained for Arabic, to answer the above research questions.
- We contribute a large dataset of Arabic tool-calling training set composed of tens of thousands of examples, and a version of the Fanar

¹<https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2>

²<https://huggingface.co/datasets/NousResearch/hermes-function-calling-v1>

³https://huggingface.co/datasets/llamafactory/glaive_toolcall_zh

open-weight Arabic LLM that is fine-tuned for tool-calling.⁴

2 Related Work

Tool calling relies on a model's ability to detect user intent, decide when to invoke a tool, and translate the query into structured parameters aligned with the tool's schema. This process entails selecting the appropriate tool, adhering to its specification, extracting and formatting the input arguments, and generating responses that conform to the expected output format. To improve LLM performance in tool use, several works have built instruction-tuning datasets that expose models to a diverse set of tools and usage patterns across varied prompt scenarios and interaction contexts (Qin et al., 2024; Patil et al., 2024; Liu et al., 2024b; Abdelaziz et al., 2024; Liu et al., 2024a).

A key emphasis in these datasets is the breadth and complexity of tool coverage, with some efforts incorporating tens of thousands of real-world APIs spanning hundreds of domains (Qin et al., 2024; Liu et al., 2024a). Beyond API diversity, these datasets increasingly capture advanced usage scenarios, including parallel and dependent tool invocations (Liu et al., 2024b; Abdelaziz et al., 2024), support for nested and structured parameter types (Liu et al., 2024a), and multi-turn interactions that require contextual memory and dialogue state tracking (Tang et al., 2023; Liu et al., 2024b). Additionally, several datasets aim to strengthen the planning and reasoning abilities required for effective tool use (Huang et al., 2024; Tang et al., 2023; Li et al., 2023). While these datasets have advanced the tool-use capabilities of LLMs, an important open question is whether they enable sufficient generalization to non-English prompts and unseen domain-specific tools. We investigate this in the context of Arabic-language tool-use with a focus on a small set of real-world tools developed for deployment in culturally and linguistically specialized settings.

A more subtle and critical challenge is integrating tool use in a way that aligns with the model's internal reasoning capabilities. Ideally, a model should invoke a tool only when its own knowledge or inference abilities are insufficient to complete the task described in the user query, or when a tool is capable of performing a required step

⁴<https://huggingface.co/collections/QCRI/arabictoolcalling-68b82e0b8f0865d6e3b179e7>

with greater efficiency and effectiveness. In this sense, effective tool use should be selective and autonomous, minimizing unnecessary calls and the associated computational or latency costs (Chen et al., 2024). Achieving this balance requires careful design of the supervised fine-tuning and preference optimization stages, ensuring that general capabilities are calibrated to support—rather than compete with—tooling.

3 Datasets

To construct our training and evaluation data, we utilized four distinct datasets (shown in Table 1). We adapted two prominent open-source function-calling datasets, namely Glaive⁵ and xLAM (Liu et al., 2024b), where we translated them into Arabic using Gemini-2.5-Flash-no-thinking (Team et al., 2023) following the prompt templates described in Appendix A. In our experiment, we use the Arabic and English versions of the datasets in isolation or in combination. We split both datasets into training and test splits, where the English and Arabic train and test splits are direct translations of each other. To address specific use cases, we curated two novel datasets. The first, CustomTools, is a collection of unique tools synthetically generated using, again, Gemini. It includes both positive examples, where a function call is required, and negative examples, where a function call is not required or not present in the list of provided tools. We synthesized Arabic and English examples. The tools cover functions such as translation, image generation, speech generation, speech recognition, text diacritization, Islamic knowledge, recent news, and person biography lookup. We list the function definitions in Appendix C.

The second, IslamicRAGTool, was built from real question-answer pairs obtained from the Fanar Arabic and English Islamic question-answering service API⁶. IslamicRAGTool is different from the other calls in three ways, namely: the dataset is based on actual logs instead of being synthetic; it involves specific topic/genre classification; and, unlike the other tools the LLM needs to pass either the user input or sequence of interactions as is without argument extraction. We provide a comprehensive overview of the datasets and their statistical properties in Table 1, and examples from

⁵<https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2>

⁶<https://api.fanar.qa/docs>

the datasets in Appendix B.

Table 1: Summary of Function-Calling Datasets. Language denotes the language of the dataset (AR = Arabic, EN = English). FC indicates whether the examples include function calls (Y = Yes, N = No). Turns specifies whether interactions are single-turn (S) or multi-turn (M), while Calls denote whether a single (S) or multiple (M) function calls occur per turn. The Train and Test columns report the number of samples in each split. The datasets Glaive, xLAM, CustomTools, and IslamicRAGTool contain 972, 3,179, 8, and 1 unique tools, respectively, distributed across their examples.

Dataset	Language	FC	Turns	Calls	Train	Test
Glaive	AR	Y	M	S	37,684	1,953
	AR	N	M	S	38,678	1,000
	EN	Y	M	S	37,684	1,953
	EN	N	M	S	38,678	1,000
xLAM	AR	Y	S	M	58,999	1,001
	AR	N	S	M	19,361	1,077
	EN	Y	S	M	58,999	1,001
	EN	N	S	M	19,361	1,077
CustomTools	AR	Y	S	S	4,528	1,000
	AR	N	S	S	4,313	1,000
	EN	Y	S	S	5,133	1,000
	EN	N	S	S	5,983	1,000
IslamicRAGTool	AR	Y	S	S	10,000	1,000
	AR	N	S	S	10,000	1,000
	EN	Y	S	S	10,000	1,000
	EN	N	S	S	10,000	1,000

4 Experimental Design

4.1 Experiments

We designed five experiments to answer the three main research questions introduced in Section 1. Each experiment evaluates a different configuration of supervised fine-tuning and tool-calling training strategies.

- **Experiment 1:** Fine-tuning of the **base** Fanar model using English tool-calling data drawn from a combination of the Glaive and XLAM datasets.
- **Experiment 2:** A direct replication of Experiment 1, but using the translated Arabic versions of the tool-calling examples from Glaive and XLAM.
- **Experiment 3:** Continued fine-tuning of **instruction-tuned** Fanar using a mix of English tool-calling examples from Glaive and XLAM.
- **Experiment 4:** Similar to Experiment 3, but using bilingual tool-calling data (English and Arabic) from the Glaive and XLAM datasets.

- **Experiment 5:** Similar to Experiment 4, where we fine-tuned the instruction tuned Fanar model with the bilingual training sets of of Glaive and XLAM along with the training splits of the CustomTools and IslamicRAGTool datasets.

In Experiments 3–5, we use the instruction-tuned Fanar model that differs from the base pre-trained model used in Experiments 1 and 2. This model has undergone both supervised fine-tuning and preference learning in Arabic and English, allowing it to more effectively follow user intent across both languages (Team et al., 2025).

4.2 Fine-Tuning Setup

We fine-tuned all models using supervised learning with LLaMA-Factory (Zheng et al., 2024). The training setup is the same for all models: we use a cosine learning rate schedule with a peak learning rate of 5.0×10^{-7} and a minimum of 5.0×10^{-8} , and a batch size of 640. We fine-tune two public models: Fanar-1-9B, a pre-trained base model, and Fanar-1-9B-Instruct, its post-trained variant (Team et al., 2025) to measure the effect of SFT on tool calling capabilities.

4.3 Evaluation Methodology

We fine-tuned the models to produce one of two outputs: a dedicated `<no_tool_call>` tag when no action is required, or a function call, with tool name and arguments, encapsulated within `<tool_call></tool_call>` tags. For evaluation, each model is tested on all test splits detailed in Table 1. To ensure a fair comparison with single-turn datasets, we decompose the multi-turn conversations from the Glaive test set into individual turns. We report the weighted-average precision and recall across all available tools, where the weighting reflects the relative importance of each tool based on its frequency in the test set.

Our evaluation methodology employs two complementary approaches: function name detection and end-to-end argument accuracy. First, we calculate the precision (P_T) and the recall (R_T) for each tool T based on function name matching only. For each tool/class, precision measures the fraction of predicted tool calls that are correct, while recall measures the fraction of actual tool calls that are correctly identified. Notably, we treat the absence of a tool call as its own tool, representing cases where no function tool is invoked:

$$P_T = \frac{\text{True Positives}_T}{\text{True Positives}_T + \text{False Positives}_T}$$

$$R_T = \frac{\text{True Positives}_T}{\text{True Positives}_T + \text{False Negatives}_T}$$

These individual scores are then aggregated using a weighted average, where each tool’s contribution is weighted by its support (N_T)—the number of true instances in the test set. The final weighted-average metrics are defined as:

$$\text{Precision}_{\text{weighted}} = \sum_{T \in K} \frac{N_T}{N_{\text{total}}} \cdot P_T$$

$$\text{Recall}_{\text{weighted}} = \sum_{T \in K} \frac{N_T}{N_{\text{total}}} \cdot R_T$$

where K is the set of all tools and N_{total} is the total number of instances.

Beyond function name detection, we assess end-to-end performance through Argument Population Accuracy (ArgA), which quantifies the proportion of function calls where both the function name and all parameter values are correctly predicted. This comprehensive metric evaluates the model’s capacity to not only select the appropriate tool but also furnish it with accurate argument values:

$$\text{ArgA} = \frac{\text{Exact Matches}}{\text{Total Positive Cases}}$$

where Exact Matches denotes instances with perfect correspondence in both function name and arguments, and Total Positive Cases encompasses all cases requiring function calls (excluding `<no_tool_call>` instances). ArgA delivers a holistic evaluation of the model’s practical effectiveness in real-world function calling applications.

To ensure reliable ArgA computation, we implement standardized normalization protocols for both ground truth and predicted function calls prior to assessment. These normalizations include lowercase normalization, elimination of extraneous whitespaces, and standardization of date formats and numerical representations. This preprocessing is essential because models may generate semantically identical outputs with minor formatting discrepancies (e.g., “2024-01-15” versus “2024/01/15” for dates, or “John Smith” versus “john smith”). By applying uniform normalization rules to both reference and predicted outputs, we focus evaluation on semantic accuracy

rather than superficial formatting differences, yielding a more precise assessment of functional performance.

5 Results and Analysis

Table 2 presents the comprehensive results of all the experiments conducted. As expected, models achieve nearly perfect precision and recall when evaluated on test examples drawn from the same domain as the training data. This pattern is consistently observed across the Glaive and xLAM test sets, where all models were trained on the respective training portions of these datasets, regardless of whether they used Arabic, English, or bilingual training data. To address our three research questions, we turn our attention to the cross-domain evaluation results obtained from the remaining datasets, which provide insights into the models’ generalization capabilities beyond their training domains.

5.1 Cross-Lingual Knowledge Transfer in Tool Calling

We examine the transferability of tool-calling capabilities between English and Arabic by comparing the results of Experiment 1 and Experiment 2. The results indicate that models trained on tool-calling data in one language (English or Arabic) can effectively transfer this ability to the other language. This suggests that the base model’s translation capabilities are sufficiently robust to cross-lingually detect the correct tool calls. However, when evaluating on previously unseen tools, particularly domain-specific ones such as CustomTools and IslamicRAGTool, we observe a significant drop in recall, where the LLM should have invoked a tool but didn’t. This gap becomes more pronounced when moving from moderately custom tools (e.g., 0.66-0.82 for CustomTools) to highly specialized ones (e.g., 0.25-0.47 IslamicRAGTool). Notably, this decline occurs regardless of the training language (either Arabic, English, or both). This highlights a broader generalization gap in tool invocation for previously unseen tools, especially those with niche or specialized behavior. Interestingly, we find that training on Arabic tool-calling data yields slightly better generalization to English than the reverse, with a consistent performance gap of approximately 0.1–0.2, depending on the dataset. This asymmetry may stem from the domain-specific nature of the

custom tools, which are more richly represented in the Arabic fine-tuning datasets. As a result, the model benefits from exposure to these specialized contexts during training, which in turn enhances its ability to generalize to English inputs.

As for argument population accuracy (ArgA), the results show that a mismatch in the language of training versus testing data adversely affects the ability of the model to guess the correct arguments, particularly for unseen tools. For example, ArgA dropped for the Arabic test set from 0.78 to 0.69 and from 0.75 to 0.61 for Glaive and xLAM respectively when training with English versus Arabic. An even sharper decline was observed for CustomTools and IslamicRAGTool with a drop from 0.77 to 0.45 and from 0.36 to 0.14 respectively. This underscores that the model struggles not only with deciding when to call a tool, but also with correctly populating its arguments.

5.2 In-Language Fine-Tuning

The addition of Arabic tool-calling data to the English fine-tuning dataset (transitioning from Experiment 3 to Experiment 4) produces notable improvements in non-function-calling performance. For the CustomTools dataset, non-FC recall increases substantially from 0.74 to 0.89 for Arabic test sets and from 0.87 to 0.94 for English test sets. Low non-FC recall indicates that the LLM chose a wrong tool instead of returning `<no_tool_call>`. In contrast, function-calling cases show minimal improvement, with English recall increasing slightly from 0.80 to 0.81 while Arabic recall remains unchanged. The IslamicRAGTool dataset exhibits a similar pattern for non-FC cases, demonstrating consistent benefits from bilingual training data. However, an unexpected trend emerges in the FC cases, where performance actually decreases. This declining pattern is not isolated to IslamicRAGTool but occurs across approximately half of the individual tools within CustomTools when comparing Experiments 3 and 4. Despite these localized drops, the overall weighted average recall remains positive, indicating that the benefits of including Arabic data outweigh the drawbacks. A more significant trend is visible in argument population accuracy, which improves markedly for Arabic test cases in both CustomTools (from 0.58 to 0.80) and IslamicRAGTool (from 0.42 to 0.49), while slightly decreasing for the corresponding English cases.

Table 2: Performance evaluation across five training configurations showing precision (P) and recall (R) for the function call detection task (measuring whether function names match), and argument population accuracy (ArgA) for end-to-end correctness requiring both correct function names and argument values. Training setups: (1) English-only tool-calling data, trained with a random mix of Glaive EN and xLAM EN; (2) Arabic-only tool-calling data, trained with a random mix of Glaive AR and xLAM AR; (3) Supervised Fine-Tuning (SFT) followed by training on a random mix of Glaive EN and xLAM EN; (4) SFT followed by a bilingual (EN + AR) random mix of Glaive and xLAM; (5) SFT followed by a bilingual (EN + AR) random mix of Glaive and xLAM, IslamicRAGTool and CustomTools. Test sets are evaluated in Arabic (AR) and English (EN). Function Calling (FC) indicates whether the test set contains positive cases requiring function calls (Yes) or negative cases without function calls (No).

Dataset	Language	FC	Exp. 1			Exp. 2			Exp. 3			Exp. 4			Exp. 5		
			P	R	ArgA	P	R	ArgA	P	R	ArgA	P	R	ArgA	P	R	ArgA
Glaive	AR	Yes	1.00	0.99	0.69	1.00	0.99	0.78	1.00	1.00	0.71	1.00	0.99	0.77	1.00	0.99	0.77
		No	1.00	0.95	-	1.00	0.98	-	1.00	0.96	-	1.00	0.99	-	1.00	1.00	-
	EN	Yes	1.00	0.99	0.90	1.00	0.99	0.88	1.00	0.99	0.91	1.00	0.99	0.91	1.00	0.99	0.91
		No	1.00	0.99	-	1.00	0.98	-	1.00	0.99	-	1.00	0.99	-	1.00	0.99	-
xLAM	AR	Yes	0.97	0.97	0.61	0.98	0.98	0.75	0.98	0.98	0.62	0.99	0.98	0.76	0.98	0.98	0.76
		No	1.00	0.98	-	1.00	0.98	-	1.00	0.97	-	1.00	0.99	-	1.00	0.99	-
	EN	Yes	0.98	0.98	0.85	0.98	0.99	0.82	0.98	0.98	0.86	0.98	0.98	0.87	0.99	0.99	0.86
		No	1.00	0.98	-	1.00	0.97	-	1.00	0.98	-	1.00	0.99	-	1.00	0.99	-
CustomTools	AR	Yes	0.98	0.66	0.45	0.97	0.82	0.77	0.98	0.86	0.58	0.98	0.86	0.80	1.00	1.00	1.00
		No	1.00	0.97	-	1.00	0.90	-	1.00	0.74	-	1.00	0.89	-	1.00	1.00	-
	EN	Yes	0.97	0.70	0.56	0.96	0.80	0.56	0.96	0.80	0.64	0.96	0.81	0.63	1.00	0.99	1.00
		No	1.00	0.98	-	1.00	0.92	-	1.00	0.87	-	1.00	0.94	-	1.00	1.00	-
IslamicRAGTool	AR	Yes	1.00	0.25	0.14	1.00	0.47	0.36	1.00	0.69	0.42	1.00	0.63	0.49	1.00	0.99	0.99
		No	1.00	0.98	-	1.00	0.94	-	1.00	0.90	-	1.00	0.95	-	1.00	1.00	-
	EN	Yes	1.00	0.44	0.33	1.00	0.58	0.33	1.00	0.71	0.54	1.00	0.62	0.51	1.00	0.99	0.99
		No	1.00	0.97	-	1.00	0.95	-	1.00	0.95	-	1.00	0.95	-	1.00	1.00	-

5.3 Effect of General SFT

The effect of general SFT data is most evident when comparing Experiment 1 and Experiment 3, revealing contrasting impacts on function-calling (FC) and non-function-calling cases across different datasets. For function-calling cases, the General SFT data produces substantial improvements in recall performance. In the CustomTools dataset, recall increases significantly from 0.66 to 0.86 for Arabic and from 0.70 to 0.80 for English, with argument population accuracy also rising from 0.45 to 0.58 and 0.56 to 0.64, respectively. The improvements are even more pronounced in the IslamicRAGTool dataset, where Arabic recall jumps from 0.25 to 0.69 and English recall rises from 0.44 to 0.71, accompanied by a major boost in ArgA from 0.14 to 0.42 for Arabic and 0.33 to 0.54 for English.

However, non-function-calling cases show a concerning decline in performance after applying general SFT data. The CustomTools dataset experiences notable drops in recall, falling from 0.97 to 0.74 in Arabic and from 0.98 to 0.87 in English. The IslamicRAGTool dataset shows a more modest decline, with Arabic recall dropping from 0.98 to 0.90 and English recall decreasing from 0.97 to 0.95. The performance decline in non-function-calling (non-FC) cases is likely due

to the supervised fine-tuning (SFT) data enhancing the model’s generative abilities while diminishing its classification precision, leading the model to incorrectly predict function calls in cases where none are required. This suggests that the general training data may be introducing a bias toward function-calling behavior.

Notably, Experiment 4 demonstrates that adding Arabic tool-calling data can help recover some of the lost performance. The Arabic non-FC recall improves from 0.74 to 0.89, indicating that language-specific training data can help balance the model’s classification behavior and mitigate the negative effects of overly confident function-calling predictions.

5.4 Tool-Specific Fine-Tuning

To address whether fine-tuning LLMs on tool-specific data is necessary, Experiment 5 involves training on all available datasets simultaneously. This comprehensive approach accounts for the substantial performance gains observed when comparing Experiment 5 to all previous experiments. The CustomTools and IslamicRAGTool results exemplify this improvement, with both recall and argument population accuracy scores reaching 0.99 or higher in most cases. These results demonstrate the

effectiveness of fine-tuning on task-specific data, effectively eliminating classification and argument population errors. This behavior aligns with the fact that training on Glaive and XLAM data yields nearly perfect tool selection results when tested on their respective test splits.

To test generalization, we tested a publicly available multilingual LLM, namely the instruction-tuned Qwen2.5-7B (Team, 2024), which was tuned for tool-calling and is comparable in size to Fanar 9B, with and without additional instruction tuning using the training splits of CustomTools and IslamRAGTool. We tested on the CustomTools and IslamRAGTool only, because we cannot exclude the possibility that Qwen was trained Glaive and/or xLAM. Table 3 shows the Qwen results with and without additional finetuning (FT and Base respectively). The results show that additional finetuning for the tools of interest yields a very large boost in tool-calling effectiveness, with both recall and argument population accuracy showing dramatic improvement. For example, the recall for IslamicRAGTool for English when tool-calling was required improved from 0.66 to 0.91, while ArgA jumped from 0.46 to 0.91. Nonetheless, the results of Qwen with continued finetuning falls short of the best Fanar results (Experiment 5), particularly for IslamicRAGTool. We suspect that this is the result of Fanar being specifically pre-trained on Arabic and Islamic content.

Table 3: Performance comparison of the base Qwen2.5-7B-Instruct model versus a version fine-tuned on the CustomTools and IslamicRagTool datasets. Metrics reported are precision (P) and recall (R) for the function call detection task (measuring whether function names match), and argument population accuracy (ArgA) for end-to-end correctness requiring both correct function names and argument values. Test sets are evaluated in Arabic (AR) and English (EN). Function Calling (FC) indicates whether the test set contains positive cases requiring function calls (Yes) or negative cases without function calls (No).

Dataset	Language	FC	Base			FT		
			P	R	ArgA	P	R	ArgA
CustomTools	AR	Y	0.95	0.85	0.64	0.99	0.96	0.95
		N	1.00	0.71	-	1.00	1.00	-
	EN	Y	0.97	0.94	0.74	0.99	0.98	0.98
		N	1.00	0.81	-	1.00	1.00	-
IslamicRAGTool	AR	Y	1.00	0.70	0.45	1.00	0.92	0.91
		N	1.00	0.89	-	1.00	1.00	-
	EN	Y	1.00	0.66	0.46	1.00	0.91	0.91
		N	1.00	0.94	-	1.00	1.00	-

5.5 Deeper Analysis of Argument Population Accuracy

While precision and recall measure a model’s ability to *select* the correct tool, the Argument Population Accuracy (ArgA) metric evaluates the more challenging task of end-to-end correctness, requiring both the function name and all argument values to be perfect. Across all experiments, a significant gap exists between tool-calling recall and the corresponding ArgA score, indicating that correctly populating arguments is a primary bottleneck for performance.

To understand the sources of ArgA failures, we conducted a detailed error analysis focusing on cases where function names were correctly identified but argument values were erroneous. From a total of 7,211 errors from all the experiments, we randomly selected 249 cases (see breakdown in Appendix D) and systematically categorized them as follows:

- **W** (Wrong argument values): The model produced incorrect arguments;
- **T** (Translation discrepancy): Expected argument values are in one language while model outputs are in another;
- **P** (Paraphrasing variance): Predicted arguments are paraphrases of the expected ones;
- **I** (Incomplete context): The query originates from a multi-turn conversation and lacks essential information, rendering certain argument values unpopulatable without prior conversational context.

The most frequent error category was Paraphrasing Variance (P), accounting for 50.2% of all argument errors. This occurs when the model generates a semantically correct argument that is syntactically different from the ground truth (e.g., "Could you tell me what Islam is?" vs. "What is Islam?"). This error type was particularly dominant in the English test sets (73.6% of errors) and more so in the specialized IslamicRAGTool dataset (82.7% of errors). This finding directly explains the dramatic success of Experiment 5, where tool-specific fine-tuning on all datasets resulted in near-perfect ArgA scores. Training on exact target examples, the model learns the exact syntactic format required, effectively eliminating paraphrasing ambiguities. The second most common issue was Translation Discrepancy (T), making up 38.2% of errors. This

error was overwhelmingly concentrated in the Arabic test sets, where it was the leading cause of failure (53.1% of all Arabic errors). This insight is critical for interpreting the cross-lingual experiments. The low ArgA scores in Experiment 1, where an English-trained model was tested on Arabic, can be directly attributed to the model’s tendency to provide arguments in English instead of Arabic. In contrast, a significant improvement in ArgA when bilingual data was introduced in Experiment 4 (e.g., increasing from 0.58 to 0.80 for CustomTools AR) demonstrates that bilingual fine-tuning is essential for teaching the model the correct language of the expected argument.

Finally, Wrong Argument Values (W) and Incomplete Context (I) were less frequent (6.8% and 4.8%, respectively). The latter category refers to cases where the user’s query originates from a multi-turn conversation and lacks essential information from previous turns, making it impossible to populate certain arguments.

In summary, this deeper analysis reveals that the primary obstacles to achieving high end-to-end tool-calling accuracy are not necessarily comprehension, but rather adherence to specific formatting rules. Cross-lingual performance is hindered by a failure to translate arguments, while generalization to new tools is challenged by syntactic ambiguity. These findings suggest that such errors are best mitigated by reducing ambiguity. The most effective approach, demonstrated in our experiments, is providing direct, in-domain examples through tool-specific fine-tuning. An alternative would be to craft highly granular function and argument descriptions. By explicitly defining expected formats, such as date conventions or required languages, such descriptions could guide the model’s behavior through in-context learning, potentially reducing the need for extensive fine-tuning data.

6 Conclusion

We conducted a series of experiments to investigate how tool-calling performance is influenced by language and the ability to generalize to previously unseen tools. Our findings highlight the importance of training on bilingual datasets, performing instruction tuning, and providing explicit examples of tool usage during fine-tuning. Most notably, we find that when developing agentic frameworks tailored to specific custom tools, direct fine-tuning

on those tools is significantly more effective compared to relying on generalization alone. In practice, this may entail continued fine-tuning of an instruction tuned model that is capable of tool calling with training examples for the tools of interest.

Limitations

Our conclusions are primarily based on experiments using two stock datasets—Glaive and xLAM—which may not fully capture the diversity of tool-calling use cases, especially in domain-specific or low-resource settings. While these datasets provide valuable benchmarks, extending the analysis to additional datasets could offer a more comprehensive view of language and generalization effects. Furthermore, our evaluation focuses on recall-based metrics and does not account for downstream utility or correctness of tool execution in real-world agentic systems. Finally, although we consider English and Arabic, additional languages with different morphological and syntactic properties may exhibit different transfer dynamics, warranting further investigation.

References

- Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, G. P. Shrivatsa Bhargav, Maxwell Crouse, Chulaka Gunasekara, Shajith Ikbai, Sachindra Joshi, Hima Karanam, Vineet Kumar, Asim Munawar, Sumit Neelam, Dinesh Raghu, Udit Sharma, Adriana Meza Soria, and 2 others. 2024. [Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP), Industry Track*. Also available at arXiv:2407.00121.
- Zhiyuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. 2024. [Towards tool use alignment of large language models](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1382–1400. Code and data: <https://github.com/zhiyuanc2001/ToolAlign>.
- Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. [Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4200–4216. ArXiv:2401.17167.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. [Api-bank: A comprehensive benchmark for tool-augmented llms](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ArXiv preprint arXiv:2304.08244.

Weiwu Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2024a. [Toolace: Winning the points of llm function calling](#). arXiv preprint arXiv:2409.00920.

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. 2024b. [Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets](#). In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*. ArXiv:2406.18518.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. 2024. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. arXiv preprint arXiv:2404.11584.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. [Gorilla: Large language model connected with massive apis](#). In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*. ArXiv:2305.15334.

Baolin Qin, Yuxuan Wang, Yifan Xu, Yuxiang Meng, Yifan Wang, Zhiyang Teng, Jun Yan, Zhiqiang Wei, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2024. [Toollm: Facilitating large language models to master 16000+ real-world apis](#). In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024)*. OpenReview: dHng200Jjr.

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. ArXiv:2306.05301.

Fanar Team, Ummar Abbas, Mohammad Shahmeer Ahmad, Firoj Alam, Enes Altinisik, Ehsannedin Asgari, Yazan Boshmaf, Sabri Boughorbel, Sanjay Chawla, Shammur Chowdhury, Fahim Dalvi, Kareem Darwish, Nadir Durrani, Mohamed Elfeky, Ahmed Elmagarmid, Mohamed Eltabakh, Masoomali Fatehkia, Anastasios Fragkopoulos, Maram Hasanain, and 23 others. 2025. [Fanar: An arabic-centric multimodal generative ai platform](#).

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805.

Qwen Team. 2024. [Qwen2.5: A party of foundation models](#).

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. 2024. [Llamafactory: Unified efficient fine-tuning of 100+ language models](#). In *ACL*. Association for Computational Linguistics.

A Prompts

System Prompt for Translating Datasets (ex. Glaive)

You are a helpful assistant tasked with translating user queries and function argument values into **Arabic**, using the descriptions of the tools and their arguments as guidance.

Translation Guidelines

- **Do not** translate function names or argument keys – only translate the **values** inside the arguments.
- **Do not** modify any values that are clearly identifiers.
- Preserve the original **JSON structure** exactly as it is.

Expected Output Format

Always return a JSON object with the following two keys:

- "question": The user query, translated into Arabic.
- "function_calls": A list of function call objects, where only the argument values are translated into Arabic.

System Prompts for Synthetic Data Generation

We do this process in a couple of steps to have the LLM focusing on a few tasks at a time, which yielded better data in our case:

Step 1: Generating questions that can be answered by the given tools.

Generate N unique English example questions that can be answered using the tools listed below. Ensure the following conditions are met:

- Each question should be **distinct**—no repetitions or near-duplicates.
- Distribute the questions **equally** across all tools.
- Format the output as a list of **tuples**, where each tuple contains:
 - The question as a string.
 - The corresponding tool/function name as a string.

The output should be a tuple containing the question and the related tool: (query, function-name).

Step 2: Populate the related arguments of the function given query/function-name pairs.

You are given a list of tuples, where each tuple contains:

- A **natural language query** (i.e., a question or instruction), and
- The corresponding **function name** (i.e., the tool that should be invoked to fulfill the query).

Each function has a predefined schema specifying its **required arguments**.

Your task is to extract and populate all **required argument values** for that function based on the information provided in the query.

For each input tuple, return a new tuple in the following format: (query, function-name, arguments-dict)

Step 3: Generate an answer and a paraphrase for each answer for every query based on the given function schema

Generate a sample answer and a paraphrase of that answer based on the provided question and the corresponding function.

- Separate the **answer** and the **paraphrase** using a **double newline character**.
- Both the answer and the paraphrase must be written in **English**.
- **Do not mention or reference the tool or function** used in either the answer or the paraphrase.
- Assume that all tools operate correctly and return accurate, expected results without any errors or failures.
- **Do not generate or execute any code**; your output should only be natural language text.
- The paraphrase must **include the answer itself**, but rephrased with different wording and sentence structure.

B Dataset Examples

The following are examples from our translated xLAM and CustomTools datasets. For clarity and readability, we've formatted them for presentation. In the original data, however, the tools and function calls are represented in raw JSON format, including detailed information about their arguments.

Example 1: xLAM Translated Example

System:

You are a helpful assistant with access to the following functions. Use them if required; otherwise, return no-tool-call. If you need more clarification to be able to use a tool, ask for clarification.

Available Tools:

- `sort_numbers(numbers, descending)`: Sorts a list of numbers.
- `binomial_probability(n, k, p)`: Calculates binomial probability.
- `wire_resistance(length_m, area_sq_m, material)`: Calculates the resistance of a wire.

User:

قم بترتيب الأرقام ﴿١.٤، ٧.٢، ٥.١، ٢.٣﴾ بترتيب تصاعدي واحسب احتمال ثنائي الحصول على ٣ نجاحات في ٥ تجارب مع احتمال نجاح ٠.٦.

Assistant:

Tool Calls:

- `sort_numbers(numbers=[3.2, 1.5, 2.7, 4.1], descending=false)`
- `binomial_probability(n=5, k=3, p=0.6)`

Example 2: CustomTools Arabic Example

System:

You are a helpful assistant with access to the following functions. Use them if required; otherwise, return no-tool-call. If you need more clarification to be able to use a tool, ask for clarification.

Available Tools:

- `generate_image(prompt, size)`: Generate an image using an image generation API.
- `generate_speech(text, save_path, voice)`: Convert written text (Arabic or English) to speech.
- `islamic_knowledge(question)`: Use for questions specifically about Islam (الإسلام), including beliefs Aqeedah (العقيدة), Quran (القرآن), Hadith (الحديث), Sunnah (السنة), Fiqh (الفقه), Seerah (السيرة), Islamic

history, rulings, and practices.

- `recent_news(question)`: Answer questions about recent events or current affairs.
- `diacritize_text(text, process_for_tts)`: Add Arabic diacritics (تشكيل) to text.
- `person_biography_lookup(question)`: Answer biographical questions.
- `translate_text(text, source_lang, target_lang)`: Translate text between languages.
- `transcribe_audio(audio_file_path)`: Transcribe audio files using ASR.

User:

ما الفرق بين الإلهام والرؤيا الصالحة؟

Assistant:

Tool Calls:

- `islamic_knowledge(question=ما الفرق بين الإلهام والرؤيا الصالحة؟)`

Tool Response:

الفرق بين الإلهام والرؤيا الصالحة يكمن في مصدرهما وكيفية تلقيهما. الإلهام هو ما يلقيه الله في القلب من معانٍ وأفكار صالحة، ويكون خفياً لا يدرك مصدره الإنسان تحديداً، وقد يكون في اليقظة أو المنام. أما الرؤيا الصالحة فهي جزء من ستة وأربعين جزءاً من النبوة، وتكون في المنام فقط، وهي بشارة من الله أو إنذار أو تعليم. الإلهام قد يكون لأي مؤمن، بينما الرؤيا الصالحة غالباً ما تكون للمؤمنين الصادقين.

Assistant:

يُقال أن الفرق بين الإلهام والرؤيا الصالحة هو في منشئهما وكيفية إدراكهما، حيث أن الإلهام يأتي من الله مباشرة إلى القلب بشكل غير مباشر وواضح، في حين أن الرؤيا الصالحة تأتي في المنام وتعتبر جزءاً من النبوة.

C Custom Tools Definitions

We list below the definitions of the tools found in our CustomTools Dataset.

translate_text

Description: Translates text from one language to another using a reliable and efficient translation API.

Parameters:

- text (string): The text to be translated.
- source_lang (string): The source language code. e.g. en, ar, etc.
- target_lang (string): The target language code.

generate_speech

Description: Text-to-Speech (TTS) converts written text in either English or Arabic to speech.

Parameters:

- text (string): The text to convert to speech.
- save_path (string): Path to save the audio file. If None, creates a default path.
- voice (string): The voice to use. Defaults to 'default'.

generate_image

Description: Generate an image using an image generation API.

Parameters:

- prompt (string): Description of the image to generate.
- size (string): Image size. Defaults to '1024x1024'.

islamic_knowledge

Description: Use for questions specifically about Islam (الإسلام), including beliefs Aqeedah (العقيدة), Quran (القرآن), Hadith (الحديث), Sunnah (السنة), Fiqh (الفقه), Seerah (السيرة), Islamic history, rulings, and practices.

Parameters:

- question (string): The question about Islamic knowledge or teachings.

transcribe_audio

Description: Transcribe audio using an ASR API.

Parameters:

- audio_file_path (string): Path to the audio file.

person_biography_lookup

Description: Answer biographical question about a person.

Parameters:

- question (string): The biographical question.

diacritize_text

Description: Adds Arabic diacritics (tashkeel – تشكيل) to Arabic text.

Parameters:

- text (string): Arabic text to diacritize.
- process_for_tts (boolean): Whether to optimize diacritization for text-to-speech. Defaults to False.

recent_news

Description: Answer questions about recent events, news, and current affairs.

Parameters:

- question (string): The question about recent events or information.

D Detailed Breakdown of ArgA Error Categories

Table 4 shows the distribution of ArgA error categories across the five experiments, separated by Arabic and English.

Arabic (AR)					
Exp	Total	P	T	W	I
Exp. 1	33	7 (21.2%)	24 (72.7%)	1 (3.0%)	1 (3.0%)
Exp. 2	29	12 (41.4%)	13 (44.8%)	3 (10.3%)	1 (3.4%)
Exp. 3	44	15 (34.1%)	25 (56.8%)	0 (0.0%)	4 (9.1%)
Exp. 4	33	17 (51.5%)	13 (39.4%)	3 (9.1%)	0 (0.0%)
Exp. 5	23	10 (43.5%)	11 (47.8%)	1 (4.3%)	1 (4.3%)
Total	162	61 (37.7%)	86 (53.1%)	8 (4.9%)	7 (4.3%)
English (EN)					
Exp	Total	P	T	W	I
Exp. 1	16	14 (87.5%)	0 (0.0%)	1 (6.2%)	1 (6.2%)
Exp. 2	26	13 (50.0%)	9 (34.6%)	3 (11.5%)	1 (3.8%)
Exp. 3	23	18 (78.3%)	0 (0.0%)	4 (17.4%)	1 (4.3%)
Exp. 4	16	14 (87.5%)	0 (0.0%)	0 (0.0%)	2 (12.5%)
Exp. 5	6	5 (83.3%)	0 (0.0%)	1 (16.7%)	0 (0.0%)
Total	87	64 (73.6%)	9 (10.3%)	9 (10.3%)	5 (5.7%)

Table 4: Distribution of ArgA error categories across experiments and languages. Categories: **P** (paraphrasing variance), **T** (translation discrepancy), **W** (wrong values), **I** (incomplete context). Bolded values mark the most frequent error category