Granite-Function Calling Model: Introducing Function Calling Abilities via Multi-task Learning of Granular Tasks

Ibrahim Abdelaziz[⋆†], Kinjal Basu[⋆†], Mayank Agarwal[⋆†],
Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav,
Maxwell Crouse, Chulaka Gunasekara, Shajith Ikbal, Sachin Joshi, Hima Karanam,
Vineet Kumar, Asim Munawar, Sumit Neelam, Dinesh Raghu, Udit Sharma,
Adriana Meza Soria, Dheeraj Sreedhar, Praveen Venkateswaran,
Merve Unuvar, David Cox, Salim Roukos, Luis Lastras, Pavan Kapanipathi[†]

IBM Research

Abstract

An emergent research trend explores the use of Large Language Models (LLMs) as the backbone of agentic systems (e.g., SWE-Bench, Agent-Bench). To fulfill LLMs' potential as autonomous agents, they must be able to identify, call, and interact with a variety of external tools and application program interfaces (APIs). This capability of LLMs, commonly termed *function calling*, leads to a myriad of advantages such as access to current and domain-specific information in databases and the outsourcing of tasks that can be reliably performed by tools. In this work, we introduce GRANITE-20B-FUNCTIONCALLING[1], a model trained using a multi-task training approach on seven fundamental tasks encompassed in function calling. Our comprehensive evaluation on multiple out-of-domain datasets, which compares GRANITE-20B-FUNCTIONCALLING to more than 15 other best proprietary and open models, shows that GRANITE-20B-FUNCTIONCALLING has better generalizability on multiple tasks across seven different evaluation benchmarks. Moreover, GRANITE-20B-FUNCTIONCALLING shows the best performance among all open models and ranks among the top on the Berkeley Function Calling Leaderboard (BFCL).

1 Introduction

Function calling provides a means for language models to leverage external tools and resources. These tools can make available to an LLM specific, up-to-date information that would otherwise be inaccessible (e.g., stored in a dynamic knowledge base) and thus reduce its proclivity for hallucinating responses (Schick et al., 2023). This is particularly crucial in enterprise use cases where a significant portion of relevant data is stored in a structured format accessible only via storage engines. In addition to knowledge access, function calling can allow an LLM to outsource tasks that are out of scope for a generalized language model. Most commonly, these tasks involve compute-heavy operations, e.g., program execution (Shinn et al., 2023), numerical calculation, or retrieval (Schick et al., 2023), and are otherwise a frequent source of LLM hallucinations (Li et al., 2023a). The importance of function calling has spurred the development of several recent data generation efforts for fine-tuning (Basu et al., 2024; Guo et al., 2024; Qin et al., 2023; Yan et al., 2024; Tang et al., 2023) and evaluation of models (Li et al., 2023b; Muennighoff et al., 2023). However, the fine-tuned models from datasets like ToolLLM (Qin et al., 2023), ToolAlpaca (Tang et al., 2023), and Gorilla (Patil et al., 2023) fall short in one (or more) of three key dimensions: (a) **Generalizability:** While the datasets are generated using diverse sets of APIs (e.g., ToolLLama uses

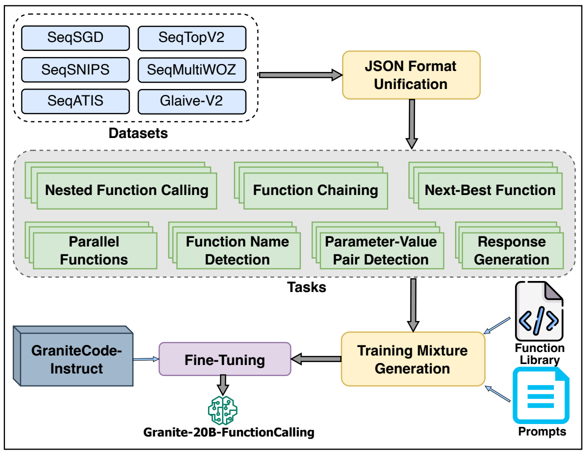

Figure 1: Step-by-step building process of GRANITE-20B-FUNCTIONCALLING.

[⋆]These authors contributed equally to this work

[†]Corresponding Authors: {ibrahim.abdelaziz1, kinjal.basu, mayank.agarwal}@ibm.com, kapanipa@us.ibm.com

[1]The model is available at: https://huggingface.co/ibm-granite/granite-20b-functioncalling

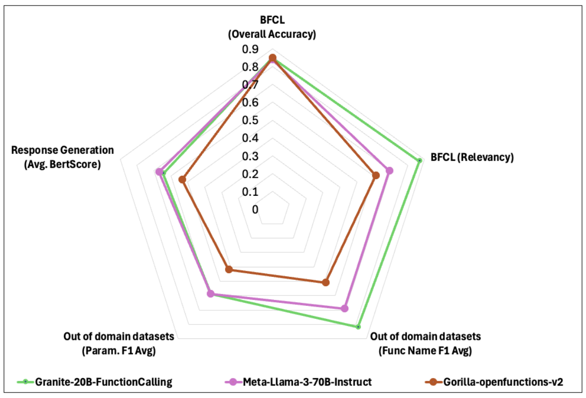

Figure 2: Evaluation of GRANITE-20B-FUNCTIONCALLING against the best *open* function calling models (according to BFCL)

RapidAPIs [2], ToolAlpaca uses public APIs[3], and Gorilla uses TensorFlow Hub, PyTorch Hub, and Hugging Face Hub), Basu et al. (2024) has shown that models trained on these datasets have difficulty generalizing to out-of-domain datasets. (b) **Ability to handle granular tasks:** Function calling, as an umbrella term, encompasses multiple granular sub-tasks such as function-name detection, slot filling[4] or parameter-value pair detection, and detecting the ordered sequence of functions needed to be called. Existing models trained to perform function calling lack the ability to handle these granular tasks independently, and hence, perform poorly on such sub-tasks. (c) **Openness:** The best performing models are proprietary and the ones that have open licenses (e.g., Gorilla (Patil et al., 2023)) are trained using data generated from OpenAI models.

Our work addresses these limitations, and centers on introducing function-calling abilities to models with an inherent focus on granular tasks. Figure 1 shows an overview of how GRANITE-20B-FUNCTIONCALLING was trained. Our work draws largely on data obtained from API-Blend (Basu et al., 2024), which comprises the following tasks: function name detection, slot filling, parallel functions, multiple functions, sequencing,[5] and calling APIs[6] using multiple programming languages. We build upon Granite code models (Mishra et al., 2024) by instruction tuning them for function calling using the datasets for granular tasks with a multi-task learning approach. We perform a comprehensive evaluation of the open and proprietary models using BFCL, four Function Calling Academic Benchmarks, and the Response Generation Benchmark (Li et al., 2023b) to evaluate the generalizability of function-calling models. GRANITE-20B-FUNCTIONCALLING is on par with the best open model on BFCL and ranks fourth overall. Furthermore, GRANITE-20B-FUNCTIONCALLING exhibits superior generalizability over other models on the out-of-domain datasets. Figure 2 shows how GRANITE-20B-FUNCTIONCALLING compares to the top two open models (according to BFCL) on various tasks where despite only having 20B parameters, it performs as well or better than Meta-Llama-3-70B-Instruct which has 70B parameters.

2 Related Work

2.1 Instruction Tuning

Our work is an instantiation of *instruction tuning* (Wei et al., 2021), a fine-tuning method that improves an LLM's ability to solve natural language tasks (Mishra et al., 2022; Wang et al., 2023). It involves taking a large collection of NLP datasets, reformulating those datasets into a set of instruction-following tasks, and then fine-tuning an LLM on the modified data. While the earliest versions of instruction tuning straightforwardly combined large datasets together, the most recent iterations use more sophisticated mixtures of tasks to achieve the best results (Li et al., 2024; Sudalairaj et al., 2024). Our work draws largely on instruction API datasets. Some examples of datasets that lie within this category are API-Blend (Basu et al., 2024), a diverse corpora of multiple API datasets focused on various API related tasks (e.g., slot filling and API intent detection), and API Pack (Guo et al., 2024), which focuses on API call code generation covering multiple programming languages.

2.2 Function Calling by LLMs

Recently, many language models with function-calling capabilities have been introduced. They broadly fall into two categories: pre-trained models with function-calling capabilities (Reid et al., 2024; CodeGemma Team et al., 2024; CohereForAI, 2024; AI@Meta, 2024; Jiang et al., 2023), and models specifically fine-tuned for function-calling (Qin et al., 2023; Tang et al., 2023; MeetKai, 2024; Patil et al., 2023; Nous-Research, 2023; Nexusflow.ai, 2023). While the pre-trained mod-

[2] https://rapidapi.com/hub
[3] https://github.com/public-apis/public-apis
[4] *Slot*, *parameter*, and *argument* are used interchangeably.
[5] *Sequencing* and *chaining* are used interchangeably.
[6] *Function* and *API* are used interchangeably.

els enable function-calling using a combination of supervised and preference fine-tuning, details of the datasets used to train models for these tasks are not generally available. In contrast, specialized function-calling models mostly rely on synthetic data generated from proprietary state-of-the-art models. For example, Gorilla (Patil et al., 2023), ToolLlama (Qin et al., 2023), ToolAlpaca (Tang et al., 2023), and the NousResearch Hermes series of models (Nous-Research, 2023) utilize GPT-4 or ChatGPT to generate synthetic instruction tuning data to fine-tune a base model (e.g., Llama, Mistral) for function-calling tasks. The NexusRaven models (Nexusflow.ai, 2023) are some of the few open-source function-calling models designed for commercial use that avoid synthetic data generation through proprietary models.

3 Multi-Task Training Data

In this section, we outline our comprehensive approach to curate multi-task function calling data to fine-tune GRANITE-20B-CODE-INSTRUCT (Mishra et al., 2024) model, thereby creating our robust GRANITE-20B-FUNCTIONCALLING model specifically designed for function-calling. Our training data mixture draws largely on API-Blend (Basu et al., 2024), which compiles five datasets (SeqSGD, SeqSNIPS, SeqTopV2, SeqATIS, and SeqMultiWOZ) totalling about 160K training examples. In addition, we also use the Glaive-V2[7] dataset. As a data pre-processing step, we unify the format of all these datasets.

A key contribution to the process of building GRANITE-20B-FUNCTIONCALLING is multi-task training, where we reuse the same data in different formats with distinct instructions for different function-calling related tasks. We identified six underlying sub-tasks for function calling and divided them into two broad categories based on their respective difficulty levels: (A) *Low-Level Function Calling Tasks* which are simpler tasks for an LLM and relate to either function names or only parameter-value pairs; and (B) *High-Level Function Calling Tasks* which are complex tasks for an LLM and typically handle multiple functions; To excel in High-Level function calling tasks, it is crucial for any LLM to master the low-level foundational sub-tasks. We have included *"Response Generation"* as the seventh task in our training data since producing natural language responses is one of the fundamental goals of an LLM. Table 1 demonstrates the task-wise mapping of each dataset. Below, we briefly describe each task.

3.1 Low-Level Function Calling Tasks

Next-Best Function In this task, given the function library along with the user query and the partial function sequence, the models are supposed to select the next most suitable function from the function library. It only requires the model to choose one function name without any parameters.

Function Name Detection This task expects the model to produce only the sequence of function names (without parameters) from the function library that are required to answer the user query. This task closely resembles Function Chaining (a High-Level task), with the sole distinction being it does not necessitate the model to populate the function's arguments.

Parameter-Value Pair Detection In this task, when provided with a user query or a user-agent conversation along with a list of parameters and their descriptions, the model must identify all the parameters for which the values are present in the query or conversation.

3.2 High-Level Function Calling Tasks

Nested Function Calling The main characteristic of this task is in the function sequence, where one function's output becomes an input to the next function. So, the answer to a user query is a sequence of nested function calls selected from the function library. Furthermore, the parameters of these function calls need to be filled by extracting the values from the user query.

Function Chaining In this task, a model needs to call multiple functions in a sequence to answer a user query. However, unlike Nested Function Calling, these functions do not have to be nested. Also, for each function, the parameters present in the user query must be passed as arguments.

Parallel Functions Similar to the Function Chaining task, here, the answer to a user query requires the same function to be called multiple times (in parallel). Similarly, parameters and their values have to be extracted from the user query.

[7] https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

	High-Level Function Calling Tasks			Low-Level Function Calling Tasks			
Datasets	**Nested Func. Calling**	**Func. Chaining**	**Parallel Func.**	**Next-Best Func.**	**Func. Name Detection**	**Param-Val Pair Detection**	**Response Generation**
SeqSGD		✓	✓	✓	✓	✓	
SeqSNIPS		✓	✓	✓	✓	✓	
SeqTopV2	✓	✓		✓	✓	✓	
SeqATIS		✓	✓	✓	✓	✓	
SeqMultiWOZ		✓		✓	✓	✓	
Glaive-V2		✓					✓

Table 1: Training Datasets with Task mapping

3.3 Response Generation

Natural language response generation is a crucial feature of any LLM. In this task, the model must comprehend an ongoing conversation between a user and an AI assistant. Then, it generates a natural language response, answering the most recent user utterance. Such responses are needed to chit-chat with the user, ask clarifying questions, or synthesize a function call's output into a natural language response.

4 Instruct Tuning

4.1 Training Data Mixture Creation

After generating the data for various tasks, the next step is to create a training data mixture including all the data. We programmatically generate the mixture of data by following a weighted configuration for datasets and tasks. Following is an *example* of the weighted configuration, where the total mixture samples will be divided between Function Chaining and Next-Best Function in a 3:5 ratio. Within the Function Chaining portion, the allocation is split between SeqSGD and Glaive-V2 in a 2:3 ratio. Similarly, the Next-Best Function chunk will be divided in a 2:1 ratio between SeqTopV2 and SeqSNIPS.

```
[{
    "instruction_name": "Function Chaining",
    "datasets": {
        "SeqSGD": 2,
        "Glaive-V2": 3
    },
    "weight": 3
},
{
    "instruction_name": "Next-Best Function",
    "datasets": {
        "SeqTopV2": 2,
        "SeqSNIPS": 1
    },
    "weight": 5
}]
```

Also, in this step, the training data is embedded with the instructions. Below is our instruction template:

```
SYSTEM: You are a helpful assistant with access to
     the following function calls. Your task is to
     produce a sequence of function calls necessary
     to generate response to the user utterance. Use
      the following function calls as required.\n<|
     function_call_library|>\n{API_SPEC_INSTRUCTION}
     \n\nUSER: {QUERY}\nASSISTANT:
```

Here, the "<|function_call_library|>" tag has been used for the function library that is demonstrated in the prompt with the placeholder named - {API_SPEC_INSTRUCTION}. As the name suggests, the {QUERY} serves as a proxy for the user query.

4.2 Training

GRANITE-20B-FUNCTIONCALLING is instruct-tuned version of GRANITE-20B-CODE-INSTRUCT (Mishra et al., 2024)[8]. For training data, we created a mixture of 142K examples spanning all the tasks' datasets discussed above. We then trained our model using QLoRA fine-tuning (Dettmers et al., 2023) based on our multi-task training mixture discussed above. In particular, we trained GRANITE-20B-FUNCTIONCALLING a QLoRA rank of 8, alpha of 32 and a dropout of 0.1. We also used a learning rate of 5e-5 and ApexFusedAdam as our optimizer with a linear learning rate scheduler. Training was done using a single node of 8 A100_80GB GPUs with 800GB of RAM for a total of 3 epochs.

5 Experimental Setup and Evaluation

In the section below, we detail our extensive evaluation on various evaluation datasets and public leaderboard. We provide a comprehensive comparison of our GRANITE-20B-FUNCTIONCALLING to other open and proprietary function calling models.

5.1 Datasets

To evaluate the model's generalizability, we evaluated GRANITE-20B-FUNCTIONCALLING on a variety of function calling benchmarks, all of which are out-of-domain evaluation for our model. It is worth noting that some of these datasets;

[8] https://huggingface.co/ibm-granite/granite-20b-code-instruct

Dataset	Test Instances	Testing tasks	Metrics
BFCL	1,700	Function Calling	AST, Execution Accuracy
		Relevancy	Accuracy
ToolLLM	491	Function Calling	Func. matching (F1)
RestGPT	157	Function Calling	Func. matching (F1)
API-Bank	473	Function Calling	Func. and Param. matching (F1)
	478	Response Generation	BERTscore, ROUGE, BLEU
ToolBench	214	Function Calling	Func. and Param. matching (F1)
ToolAlpaca	100	Function Calling	Func. and Param. matching (F1)
NexusRaven	318	Function Calling	Func. and Param. matching (F1)

Table 2: Evaluation Datasets

e.g. ToolAlpaca and ToolLLM, have training data releases. However, we *did not use* any of these benchmarks to train GRANITE-20B-FUNCTIONCALLING and we only used the datasets mentioned in Table 1.[9] Table 2 depicts the details of the evaluation datasets we used.

5.2 Evaluation Metrics

Below, we define the metrics we adopted for specific tasks in function calling.

BFCL Metrics[10]: BFCL evaluates multiple tasks using the following four metrics.
(1) *AST summary* compares the abstract syntax tree of the function output to the ground truth and the function definition. It captures the correctness of the functions called, their parameters (required or not), and the parameter types.
(2) *Execution Summary* compares the execution output from generated and ground-truth function calls. This metric is used to evaluate REST APIs and non-REST data samples.
(3) *Relevance* evaluates the model's ability to detect no function calls when the given list of functions is irrelevant to the user query. This inversely captures the hallucination rate of models.
(4) *Overall Accuracy* is the weighted average of all individual data splits in BFCL.

The same metrics described above cannot be used for our out-of-domain datasets because of missing information, varied formats, and response generation tasks. For example, ToolLLM datasets have missing arguments, ToolAlpaca has missing argument types, and API-Bank has a response generation task. Therefore, we use the following metrics to evaluate the models on other datasets:

F1 measure: Based on Basu et al. (2024), we opted for standard metrics like precision, recall, and F1 scores which focus on exactly matching API and parameters' names. The reason behind this is that APIs are very specific and unless everything (e.g., name, parameters, input/output format, etc.) matches the API specifications, executing such APIs will not be possible. We report F1 for matching function names as well as parameter names and values.

Longest Common Subsequence (LCS) and Exact match: We also used LCS from Basu et al. (2024) to capture the overlap between the gold and predicted sequences of APIs. This allows us to compute models' ability to predict APIs in the correct sequence as required by the user. Similarly, exact match score (Basu et al., 2024) checks if all APIs are predicted by the model and are in the same order.

BERTScore, ROUGE-L and BLEU: We follow the evaluation in API-Bank (Li et al., 2023b), a dialog dataset that also evaluates model responses based on language generation metrics such as Rouge-L (Lin, 2004), BertScore (Zhang et al., 2019), and BLEU (Papineni et al., 2002).

Hallucination Rate: We compute the hallucination rate as the number of samples where the model predicted an API not provided in the function library.

5.3 Evaluation Results

Tables 3, 4, 5, 6, and Figure 3 depict an extensive evaluation of GRANITE-20B-FUNCTIONCALLING in comparison to other state of the art function calling models. In order to detail this evaluation and analyses, below we categorize the results into (a) BFCL Evaluation, (b) Function calling academic benchmarks, and (c) Response Generation.

5.3.1 BFCL Leaderboard Evaluation Results

Table 3 shows that GRANITE-20B-FUNCTIONCALLING is ranked fourth on the overall accuracy metric among the top 15 models on BFCL and is highest among models with open licenses[11]. While it is tied with the Gorilla (Patil et al., 2023) model, it is important to note that the latter was finetuned on data that are (a) generated from ChatGPT, and (b) similar data to the test set and hasn't generalized well to other datasets as shown in Table 4 and

[9]We could not verify whether some (or all) of the out-of-domain datasets were used in other models' training sets.

[10]https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html#metrics

[11]We have picked the best performing version of each model. For example, Gemini-1.5-Pro-Preview-0514 (FC) and Gemini-1.5-Pro-Preview-0409 (FC) are both part of the leaderboard but for our evaluation, we consider the best one.

Model	Organization	License	AST Summary	Exec. Summary	Relevance	Overall Acc.
Claude-3.5-Sonnet-20240620 (Prompt)	Anthropic	Proprietary	91.31	89.50	85.42	90.00
GPT-4-0125-Preview (Prompt)	OpenAI	Proprietary	91.22	88.10	70.42	88.00
Gemini-1.5-Pro-Preview-0514 (FC)	Google	Proprietary	87.92	83.32	89.58	86.35
GRANITE-20B-FUNCTIONCALLING	IBM	Apache 2.0	84.11	86.50	87.08	84.71
Gorilla-OpenFunctions-v2 (FC)	Gorilla	Apache 2.0	89.38	81.55	61.25	84.71
Meta-Llama-3-70B-Instruct (Prompt)	Meta	MetaLlama 3	87.74	85.32	69.17	83.88
FireFunction-v2	Fireworks	Apache 2.0	86.44	80.26	56.67	81.88
Mistral-Medium-2312 (Prompt)	Mistral AI	Proprietary	83.76	73.47	88.33	81.35
Functionary-Medium-v2.4 (FC)	MeetKai	MIT	85.61	75.71	74.17	80.47
Command-R-Plus (Prompt) (Opt.)	Cohere	cc-by-nc-4.0	83.60	86.74	54.17	80.35
Functionary-Small-v2.4 (FC)	MeetKai	MIT	83.55	76.31	67.92	79.94
Mistral-large-2402 (FC Auto)	Mistral AI	Proprietary	64.73	60.01	84.17	68.76
Nexusflow-Raven-v2 (FC)	Nexusflow	Apache 2.0	65.19	73.89	57.50	67.35
DBRX-Instruct (Prompt)	Databricks	Databricks	66.62	74.92	55.83	65.88
Snowflake-arctic-Instruct (Prompt)	Snowflake	Apache 2.0	61.09	80.04	59.58	65.18

Table 3: Berkeley Function Calling Benchmark: Top 15 models by Overall Accuracy (as of 06/25/2024). All evaluations are done in a *zero-shot* manner.

	ToolLLM-G1			ToolLLM-G2			ToolLLM-G3			RestGPT			Average		
	Func. Match	LCS	Exact Score	Func. Match	LCS	Exact Score	Func. Match	LCS	Exact Score	Func. Match	LCS	Exact Score	Func. Match	LCS	Exact Score
Functionary-small-v2.4 (7B)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.30	0.06	0.07	0.07	0.02
Gorilla-openfunctions-v2 (7B)	0.59	0.59	0.28	0.48	0.48	0.22	0.51	0.52	0.24	0.21	0.21	0.01	0.44	0.45	0.19
Hermes-2-Pro-Mistral (7B)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.03	0.01	0.01	0.01	0.00
Mistral-Instruct-v0.3 (7B)	0.49	0.49	0.26	0.51	0.49	0.30	0.36	0.33	0.13	0.36	0.37	0.08	0.43	0.42	0.19
CodeGemma-Instruct (7B)	0.59	0.59	0.21	0.53	0.53	0.13	0.52	0.54	0.16	0.22	0.23	0.02	0.46	0.47	0.13
Nexusflow-Raven-v2 (13B)	0.65	0.65	0.39	0.73	0.72	0.43	0.68	0.66	0.27	0.39	0.41	0.06	0.61	0.61	0.28
C4AI-Command-R-v01 (35B)	0.65	0.64	0.39	0.73	0.71	0.45	0.69	0.68	0.23	**0.59**	**0.60**	**0.22**	<u>0.66</u>	<u>0.66</u>	<u>0.32</u>
Meta-Llama-3-70B-Instruct (70B)	0.61	0.61	0.31	0.59	0.58	0.21	0.65	0.64	0.23	0.22	0.22	0.01	0.52	0.51	0.19
GRANITE-20B-FUNCTIONCALLING	**0.86**	**0.85**	**0.63**	**0.84**	**0.82**	**0.58**	**0.76**	**0.73**	**0.35**	0.51	0.52	0.15	**0.74**	**0.73**	**0.43**

Table 4: Function Calling Academic Benchmarks: Function Name Detection. Best performance is highlighted in **bold**, second best is underlined. All evaluations are done in a *zero-shot* manner.

Figure 3. In the context of model sizes, GRANITE-20B-FUNCTIONCALLING is one of the smallest models in the list. Specifically, the ones better than GRANITE-20B-FUNCTIONCALLING in the ranking are all significantly larger in size.

For the BFCL evaluation dataset, we highlight concerns in certain categories, particularly the Java, JavaScript, and REST API evaluations. We are concerned with how the Java and JavaScript categories evaluate a function-calling model's capabilities to follow language-specific syntax, for instance how objects are instantiated and called in Java and JavaScript utilizing language-specific context and norms. For the REST API category, we observed significant brittleness in the evaluation due to issues with API availability and API call limits.

5.3.2 Function Calling Academic Benchmarks

Tables 4 and 5 focus on evaluating the models' performance on Function Matching using F1-measure, LCS, and Exact Match. In this experiment, we reuse the model handlers from the BFCL code base, including the optimized prompts for each model. However, since the Cohere Command-R-v01 and Mistral-Instruct-v0.3 handlers available in BFCL use the REST API interface for inference, we reimplement handlers for these models, utilizing local models using prompts suggested by the respective model developers for function calling.

Function Name Detection: On ToolLLM datasets (G1, G2, and G3) and RestGPT, GRANITE-20B-FUNCTIONCALLING performs the best on detecting function names given a natural language utterance with **8%** better F1 score than the next best function calling model, as shown in Table 4. Since these datasets have multiple functions in sequence, we also compute sequencing metrics; exact score and LCS. On this front, GRANITE-20B-FUNCTIONCALLING model also outperforms other function calling models by **7%** on LCS and **11%** on Exact Match scores.

Full Function Calling: Table 5 reports on the models' performance on the API-Bank, ToolBench, and ToolAlpaca datasets that are out-of-domain and evaluated in a zero-shot manner. No single model outperforms all other models across datasets. Note that datasets like ToolAlpaca and API-Bank come with training data split which we never used for training GRANITE-20B-FUNCTIONCALLING, but could not guarantee that the other models were not trained with it too. Averaging out the F1 scores across datasets shows that GRANITE-20B-FUNCTIONCALLING achieves an F1 score of 0.87 when predicting the function name; second best by 0.01 to Cohere's Command-R (a 35B model) which provides an F1 score of 0.88. When predicting the

	Func-Name+Args Det. (F1 Func-Name ǀ F1 Args)						F1 Average	
	API-Bank L-1	API-Bank L-2	ToolBench HS	ToolBench B	Tool-Alpaca	Nexus Raven	Func Name	Args
Functionary-small-v2.4 (7B)	0.78 ǀ 0.70	0.54 ǀ 0.45	0.73 ǀ 0.68	0.65 ǀ 0.33	0.88 ǀ **0.47**	0.82 ǀ 0.64	0.73	0.55
Gorilla-openfunctions-v2 (7B)	0.43 ǀ 0.41	0.12 ǀ 0.12	0.86 ǀ 0.69	0.41 ǀ 0.27	0.69 ǀ 0.39	0.81 ǀ 0.65	0.55	0.42
Hermes-2-Pro-Mistral (7B)	**0.93** ǀ **0.77**	0.54 ǀ 0.25	0.51 ǀ 0.40	0.56 ǀ 0.26	0.80 ǀ 0.26	0.90 ǀ 0.63	0.71	0.43
Mistral-Instruct-v0.3 (7B)	0.79 ǀ 0.69	0.69 ǀ 0.46	0.60 ǀ 0.47	0.04 ǀ 0.16	0.33 ǀ 0.33	0.71 ǀ 0.54	0.53	0.44
CodeGemma-Instruct (7B)	0.77 ǀ 0.57	0.59 ǀ 0.38	0.65 ǀ 0.50	0.54 ǀ 0.22	0.59 ǀ 0.31	0.84 ǀ 0.68	0.66	0.44
Nexusflow-Raven-v2 (13B)	0.51 ǀ 0.42	0.28 ǀ 0.22	**0.92** ǀ 0.65	0.89 ǀ 0.35	0.85 ǀ 0.37	0.92 ǀ **0.75**	0.73	0.46
C4AI-Command-R-v01 (35B)	**0.93** ǀ 0.76	0.77 ǀ 0.54	0.85 ǀ 0.77	0.88 ǀ 0.49	**0.90** ǀ 0.42	**0.93** ǀ 0.71	**0.88**	**0.62**
Meta-Llama-3-70B-Instruct (70B)	0.85 ǀ 0.67	0.69 ǀ 0.52	0.91 ǀ **0.86**	**0.91** ǀ **0.56**	0.78 ǀ 0.43	0.70 ǀ 0.52	0.81	<u>0.59</u>
GRANITE-20B-FUNCTIONCALLING	0.91 ǀ 0.71	**0.83** ǀ **0.60**	0.87 ǀ 0.71	0.82 ǀ 0.36	0.89 ǀ 0.44	0.92 ǀ 0.72	<u>0.87</u>	<u>0.59</u>

Table 5: Function Calling Academic Benchmarks: Full Function Calling. Best performance is highlighted in **bold**, second best is <u>underlined</u>. All evaluations are done in a *zero-shot* manner.

	API-Bank-Response-Level 1			API-Bank-Response-Level 2		
Models	BertScore	Rouge-L	BLEU	BertScore	Rouge-L	BLEU
Functionary-small-v2.4 (7B)	0.34	0.23	0.05	0.35	0.23	0.05
Gorilla-openfunctions-v2 (7B)	0.56	0.33	0.32	0.51	0.26	0.25
Hermes-2-Pro-Mistral (7B)	0.45	0.18	0.09	0.42	0.14	0.06
Mistral-Instruct-v0.3 (7B)	0.52	0.29	0.22	0.46	0.20	0.14
CodeGemma-Instruct (7B)	0.14	0.03	0.00	0.09	0.02	0.01
Nexusflow-Raven-v2 (13B)	0.41	0.16	0.11	0.38	0.11	0.06
C4AI-Command-R-v01 (35B)	0.39	0.15	0.07	0.39	0.15	0.06
Meta-Llama-3-70B-Instruct (70B)	**0.69**	**0.48**	**0.47**	**0.65**	**0.40**	**0.40**
GRANITE-20B-FUNCTIONCALLING	<u>0.68</u>	<u>0.47</u>	**0.47**	<u>0.61</u>	<u>0.36</u>	<u>0.37</u>

Table 6: API-Bank Response generation dataset evaluation. Results are averaged across each dataset per model. Best performance is highlighted in **bold**, second best is <u>underlined</u>. All evaluations are done in a *zero-shot* manner.

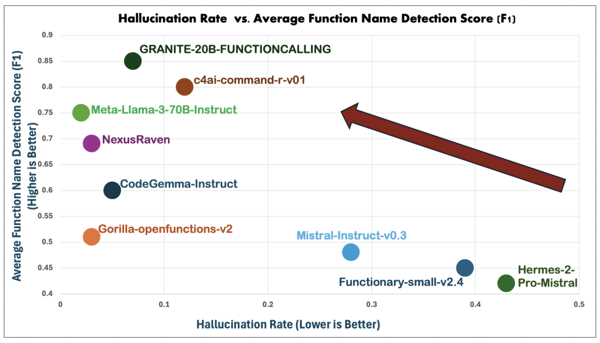

Figure 3: Performance vs. Hallucination rates for Out-of-Domain Function Calling

arguments, GRANITE-20B-FUNCTIONCALLING average F1 score lags behind the best model (Cohere's Command-R) by 0.03; 0.62 vs. 0.59.

Function Name Hallucination: Hallucinations have been a major drawback of LLMs. In the context of calling and executing APIs, hallucinations can have adverse consequences. In Figure 3, we compare the models' Function Name Detection Scores (average F1) over all the datasets (except BFCL, which uses AST-based metrics) and their hallucination rates. Ideally, we want models to have high performance and low hallucination rates (top left corner of the plot). GRANITE-20B-FUNCTIONCALLING has the highest performance with less than 0.1 hallucination rate.

5.3.3 Response Generation

Table 6 shows the models' performance on response generation task. In this experiment, we used API-Bank dataset and followed their response generation task evaluation with BertScore, Rouge-L, and BLEU. Meta-Llama-3-70B-Instruct has the best performance across the three metrics with GRANITE-20B-FUNCTIONCALLING coming in close second (performance difference ranged between 1-5%). Both models significantly outperform all other evaluated models. The gap widens when we compare GRANITE-20B-FUNCTIONCALLING to the ones specifically trained for function calling such as Functionary-small-v2.5 and Gorilla-openfunctions-v2.

6 Conclusion

In this paper, we introduced GRANITE-20B-FUNCTIONCALLING, a capable function calling open model with Apache 2.0 license. It is trained using a suite of datasets transformed from different domains and with a multi-task learning approach. We performed an extensive evaluation of our model in comparison to other state-of-the-art function calling models. On multiple out-of-domain datasets, including BFCL, our model outperform the other open models. Even compared to multiple proprietary models with much larger sizes, our model showed on-par and in some cases better performance on multiple datasets and tasks.

References

AI@Meta. 2024. Llama 3 model card.

Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A. Lastras. 2024. Api-blend: A comprehensive corpora for training and benchmarking api llms. *Preprint*, arXiv:2402.15491.

CodeGemma Team, Ale Jakse Hartman, Andrea Hu, Christopher A. Choquette-Choo, Heri Zhao, Jane Fine, and Hui. 2024. Codegemma: Open code models based on gemma.

CohereForAI. 2024. C4ai command-r: A 35 billion parameter generative model for reasoning, summarization, and question answering. *Hugging Face Models*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Preprint*, arXiv:2305.14314.

Zhen Guo, Adriana Meza Soria, Wei Sun, Yikang Shen, and Rameswar Panda. 2024. Api pack: A massive multi-programming language dataset for api call generation. *Preprint*, arXiv:2402.09615.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2023a. Chain of code: Reasoning with a language model-augmented code emulator. *arXiv preprint arXiv:2312.04474*.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, et al. 2024. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. Api-bank: A comprehensive benchmark for tool-augmented llms. *Preprint*, arXiv:2304.08244.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

MeetKai. 2024. Functionary-7b-v1.4: A language model for function interpretation and execution. *Hugging Face Models*.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. 2023. Crosslingual generalization through multitask finetuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15991–16111, Toronto, Canada. Association for Computational Linguistics.

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

Nous-Research. 2023. Nous-hermes-13b. `https://huggingface.co/NousResearch/Nous-Hermes-13b`. Model on Hugging Face.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shivchander Sudalairaj, Abhishek Bhandwaldar, Aldo Pareja, Kai Xu, David D Cox, and Akash Srivastava. 2024. Lab: Large-scale alignment for chatbots. *arXiv preprint arXiv:2403.01081*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.