LLM Tool-Use for Travel Booking

Yuriy Sosnin

July 2025

1 Introduction

Many modern open-source LLMs can already handle simple tool-use tasks out of the box, especially when tools are defined using OpenAI-style function schemas [6]. In particular, larger instruction-tuned models like Qwen2.5-7B-Instruct [9] show near-perfect accuracy on single-turn cases. That said, performance drops quickly for smaller models, and it's unclear how much of that is due to scale vs. alignment training.

In this report, we explore this gap by evaluating two related model families - Qwen2 [13] and Qwen2.5 [9] - at different sizes (0.5B to 7B) on a narrow function-calling task. We limit scope to single-turn queries with all arguments specified, and compare zero-shot performance across models. To make things more interesting, we also fine-tune Qwen2-0.5B-Instruct on a tool-use dataset (APIGen [4]), to see if we can recover accuracy with minimal training.

We evaluate on a subset of Berkeley Function-Calling Leaderboard [7] (a standard tool-use benchmark) and a small synthetic dataset for booking queries. Accuracy is measured as exact match with ground truth JSON calls. While not a full agent setup, this gives us a clean way to isolate tool-use behavior without reasoning, dialogue, or orchestration.

2 Related Works

Early work on tool use with LLMs mostly focused on showing that it's possible. Toolformer [10] used self-supervised data collection to train a model to call APIs mid-generation. ReAct [14] and WebGPT [5] demonstrated that reasoning and decision-making with tools was viable, especially with access

to external state (like search results or environments). But these approaches were more agent-style and didn't isolate tool-calling as a standalone behavior.

Later work focused more directly on structured tool use and introduced proper benchmarks. Gorilla [8] and ToolLLM [**?**] both evaluated open models on JSON-style function calling, and Gorilla introduced what is now the Berkeley Function-Calling Leaderboard (BFCL) [7], which has become one of the standard benchmarks for the task. More datasets later appeared, like Nexus [11] or APIGen [4].

Closed models like GPT-4 [6], Claude [1], and Gemini [2] arguably led the way in terms of real-world function calling, especially in production tools and agents. OpenAI's method of using a standard JSON schema is increasingly becoming standard, with most models turning to follow it instead of inventing something new.

In the newest generation of open models, tool use has become a part of instruction tuning. Qwen2.5 [9], LLaMA-3 [**?**], and others explicitly advertise tool-use capabilities, often include it in their system prompts, and report benchmark performance. But details vary - some models support multi-call or tool choice, others don't — and there's still a lot of variation in how robust the behavior is, especially at smaller scales.

3 Task Setup & Evaluation Metrics

In this project, we strictly focus on single-turn, zero-shot, OpenAI-style [6] function calling, without CoT [12], ReAct [14], or any other form of orchestration. We also do not consider underspecified user queries, i.e. where some required field is missing in the context and the LLM call should result in *rejection* (see Discussion). Therefore, we define the following interaction protocol: the system prompt specifies tools available to the model, in JSON Schema format; the user asks something that necessitates the use of at least one of the tools; the model is expected to produce a structured JSON object with function names and parameters.

Although our target application is flight and hotel booking, we believe it is valuable to evaluate on a broad and general tool-use benchmark. We choose Berkeley Function Calling Leaderboard (BFCL) [7]. It consists of multiple different subsets, corresponding to different flavors and difficulty levels of the task. We only use the "simple" and "multiple" subsets, corresponding to one function + one call and multiple functions + one call setups, respectively.

On top of that, we also follow BFCL format to design our own small benchmark focused on the booking domain and our concrete functions. By design, it always has only 2 available functions, and one of 2 intents: hotel or flight. To collect the dataset, we use a common technique of synthetic data collection by reversing the task. Basically, we prompt GPT-4.1-mini with a seed example and the function definition, and ask it to generate both a plausible query and correct ground truth JSON call. There are 20 examples in total, half for hotel booking and the other for flights.

The primary metric used to evaluate this task is tool-calling accuracy: for each example, we compare the model's predicted JSON call(s) to the ground truth, available in advance. Following BFCL, we use AST-based validation - that is, we compare function name, argument structure, and values semantically, rather than relying on exact string match.

4 Experimental Setup

To comply with the constraints of the assignment, we limit our study to sub-7b open-source language models. Within this range, we select two related model families: Qwen2 [13] and Qwen2.5 [9], each released in 0.5b, 1.5b, and 7b variants.

These families provide a useful basis for systematic comparison. First, they allow us to investigate the scaling behavior of tool-use performance as a function of model size. Second, they differ in their alignment strategies: Qwen2.5 models have been explicitly instruction-tuned on tool-use, while Qwen2 models have not (or at least no such training is disclosed). This divergence allows us to isolate the impact of tool-specific alignment, in particular at smaller scales where zero-shot performance is less robust.

In the first stage of our study, we evaluate these models as is, with the default Qwen2.5 system prompt that includes tool definitions in OpenAI-style JSON Schema format. Since Qwen2 does not have a recommended tool-use prompt format, we use the same prompt for it too, for consistency.

We notice that Qwen2-0.5B-Instruct significantly underperforms, nearing 0 accuracy, and decide to fine-tune it, to assess how effective SFT fine-tuning would be for tool use on such a small scale. We use the APIGen Function-Calling Dataset [4] - we choose this dataset over OpenFunctions [8] or Nexus [11] due to its compatibility with OpenAI-style Schema out of the box. APIGen consists of 60k examples of various Python functions, with multiple

available tools and multiple possible calls in each example.

We train a simple LoRA adapter [3] for one epoch, with the effective batch size of 32 and learning rate of 2e-4. One training run on 1 × A100 40GB in bf16 precision takes about 40 minutes.

5 Results

Table 1: Model Performance Comparison: Qwen2 vs Qwen2.5 Across Model Sizes

Dataset	**Subset**	**Qwen2**			**Qwen2.5**			**Qwen2-SFT**
		0.5B	**1.5B**	**7B**	**0.5B**	**1.5B**	**7B**	**0.5B**
bfcl	simple	0.045	0.6225	0.935	0.78	0.84	0.9625	0.7475
	multiple	0.15	0.825	0.955	0.7	0.85	0.965	0.705
booking	multiple	0.1	0.85	0.85	0.35	0.8	0.75	0.5

With the model families and datasets in place, we proceed to evaluate performance across two experimental stages: (1) zero-shot inference using the base instruction-tuned models, and (2) supervised fine-tuning of a smaller model (Qwen2-0.5B-Instruct) to assess learnability under low-scale alignment. All evaluations use the task and metrics setup described in Section 3. Our experimental results can be examined in Table 1.

Looking at zero-shot performance, one can see that 7B models already perform nearly perfectly - interestingly, both models trained on tool use and not. Additionally, performance drops with model size, but only slightly on Qwen2.5 and significantly on Qwen2. At the smallest model level, 0.5B, performance drops considerably, nearly to zero. However, fine-tuning this model on actual tool-use dataset improves the score dramatically, allowing it to be on par even with Qwen2.5.

While both benchmarks show similar dynamics, our booking benchmark has lower scores and higher variance mostly due to date formatting, which turns out to be difficult for the models to do right.

6 Limitations & Discussion

There are several important setups we explicitly leave out in this report. First, we do not test parallel tool use, where multiple tools are invoked in

a single step. While BFCL includes this subset, and we could expand our benchmark with this form too, we focus only on one-call setups for clarity.

More critically, we do not consider multi-turn dialogue, i.e. feeding the tool result back into the model and expecting either further tool use or natural language output. We also do not test reasoning before calling (e.g., CoT or ReAct-style prompting), nor any form of state tracking.

We also leave multilingual evaluation out. Many recent models claim some level of non-English coverage (especially Qwen), and testing, for example, Russian inputs could have been an interesting axis.

Finally, a major open question - in our work and in the literature - is evaluating *rejections*. That is: can a model recognize when it lacks context to call a tool, and choose to do nothing (or ask a follow-up), especially in ambiguous cases? Anecdotally, the Qwen models we tested tend to hallucinate missing parameters instead. One way to deal with the problem would be to define a synthetic "reject" tool explicitly - something we did not explore, but which would make rejection measurable under the same metric.

7 Conclusion

We explored tool-use capabilities of sub-7B instruction-tuned models in a simplified single-turn setting. By comparing Qwen2 and Qwen2.5 families across sizes and running a small fine-tuning experiment on Qwen2-0.5B-Instruct, we showed how scale and alignment both affect performance.

Out-of-the-box results on BFCL and a custom booking benchmark confirmed that tool-specific alignment matters most at smaller scales, but becomes less important at 7B. A single LoRA fine-tune was enough to recover most of the gap for a small unaligned model.

References

[1] Anthropic. Tool use with claude. https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview, 2025. Accessed: 2025-07-16.

[2] Google AI for Developers. Function calling with the gemini api. https://ai.google.dev/gemini-api/docs/function-calling, 2025. Accessed: 2025-07-16.

[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[4] 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. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets, 2024.

[5] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.

[6] OpenAI. Function calling. https://platform.openai.com/docs/guides/function-calling, 2025. Accessed: 2025-07-16.

[7] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[8] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

[9] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

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

[11] Nexusflow.ai team. Nexusraven: Surpassing the state-of-the-art in open-source function calling llms, 2023.

[12] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

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

[14] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

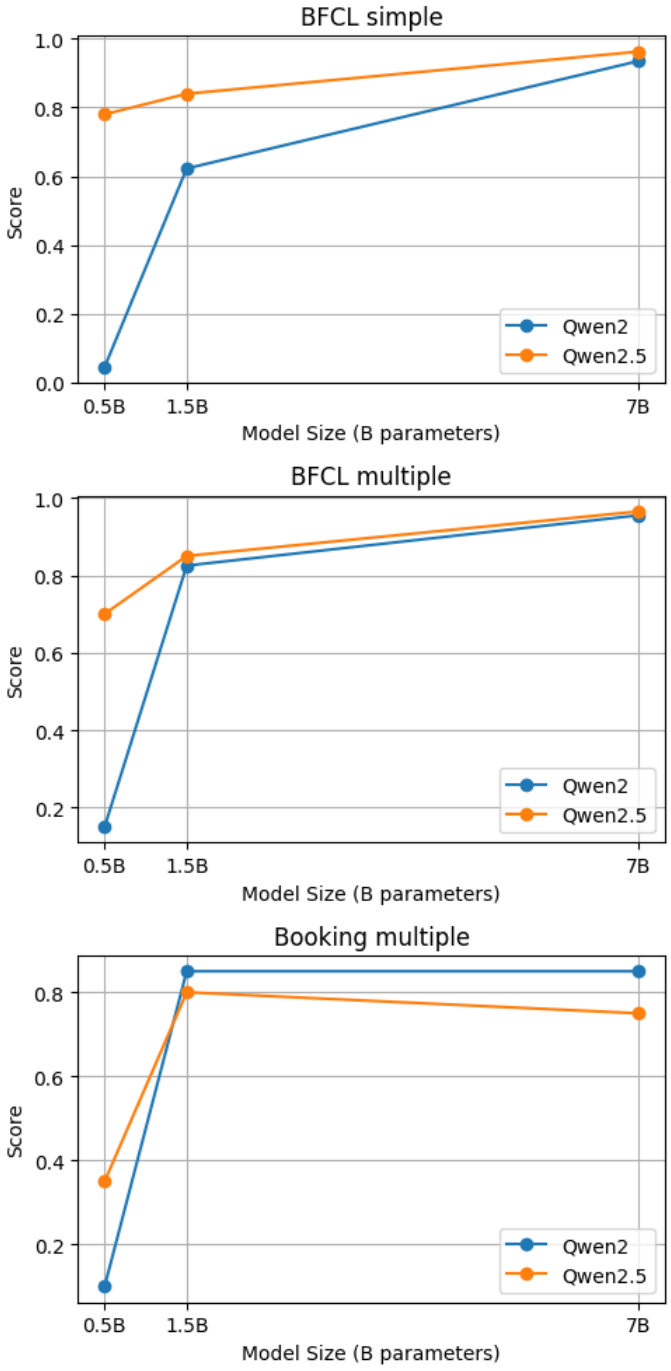

Figure 1: Model Performance vs Size: Qwen2 vs Qwen2.5 Comparison