TinyAgent: Function Calling at the Edge

Lutfi Eren Erdogan[*1] **Nicholas Lee**[*1] **Siddharth Jha**[*1] **Sehoon Kim**[1]
Ryan Tabrizi[1] **Suhong Moon**[1] **Coleman Hooper**[1]
Gopala Anumanchipalli[1] **Kurt Keutzer**[1] **Amir Gholami**[1,2]
[1]UC Berkeley [2]ICSI
{lerdogan, nicholas.lee, sidjha, sehoonkim, rtabrizi, suhong.moon, chooper, gopala, keutzer, amirgh}@berkeley.edu

Abstract

Recent large language models (LLMs) have enabled the development of advanced agentic systems that can integrate various tools and APIs to fulfill user queries through function calling. However, the deployment of these LLMs on the edge has not been explored since they typically require cloud-based infrastructure due to their substantial model size and computational demands. To this end, we present TinyAgent, an end-to-end framework for training and deploying task-specific small language model agents capable of function calling for driving agentic systems at the edge. We first show how to enable accurate function calling for open-source models via the LLMCompiler framework. We then systematically curate a high-quality dataset for function calling, which we use to fine-tune two small language models, TinyAgent-1.1B and 7B. For efficient inference, we introduce a novel tool retrieval method to reduce the input prompt length and utilize quantization to further accelerate the inference speed. As a driving application, we demonstrate a local Siri-like system for Apple's MacBook that can execute user commands through text or voice input. Our results show that our models can achieve, and even surpass, the function-calling capabilities of larger models like GPT-4-Turbo, while being fully deployed at the edge. We open-source our dataset, models, and installable package[1] and provide a demo video for our MacBook assistant agent[2].

1 Introduction

The ability of LLMs to execute commands through plain language (e.g. English) has enabled agentic systems that can complete a user query by orchestrating the right set of tools (e.g. ToolFormer (Schick et al., 2024), Gorilla (Patil et al., 2023)). This, along with the recent multi-modal efforts such as the GPT-4o (OpenAI, 2024) or Gemini-1.5 (Google, 2024), has expanded the realm of possibilities with AI agents. However, the large model size and computational requirements of these models often requires their inference to be performed on the cloud. This can create several challenges for their widespread adoption. First, uploading data such as video, audio, or text documents to a third-party vendor on the cloud, can result in privacy issues. Second, this requires cloud/Wi-Fi connectivity which is not always possible. For instance, a robot deployed in the real world may not always have a stable connection. Besides that, latency could also be an issue as uploading large amounts of data to the cloud and waiting for the response could slow down response time, resulting in unacceptable time-to-solution. These challenges could be solved if we deploy the LLM models locally at the edge.

Current LLMs like GPT-4o (OpenAI, 2024) or Gemini-1.5 (Google, 2024) are too large for local deployment. One contributing factor is that a lot of the model size ends up memorizing general information about the world into its parametric memory which may not be necessary for a specialized downstream application. For instance, if you ask a general factual question to these models like a historical event or well-known figures, they can produce the results using their parametric memory, even without having additional context in their prompt. This implicit memorization of training data into the parametric memory might be correlated with "emergent" phenomena in LLMs such as in-context learning and complex reasoning, which has been the driving force behind scaling the model size.

This leads to an intriguing research question:

Can a smaller language model with significantly less parametric memory emulate such emergent ability of these larger language models?

*Equal contribution
[1]https://github.com/SqueezeAILab/TinyAgent
[2]https://www.youtube.com/watch?v=0GvaGL9IDpQ

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing
System Demonstrations, pages 80–88
November 12–16, 2024

In this work, we demonstrate that this is feasible by training smaller models with specialized, high-quality data that does not require recalling generic world knowledge. Our goal is to develop Small Language Models (SLMs) that can be securely and privately deployed at the edge while maintaining the complex reasoning capability to understand natural language queries and orchestrate tools and APIs to accomplish user commands.

To achieve this, we first explore enabling small open-source models to perform accurate function calling, a key component of agentic systems. Off-the-shelf SLMs often lack sophisticated function calling capabilities and require fine-tuning. Next, we discuss systematically curating high-quality function calling datasets to train these SLMs, using a specialized Mac assistant agent as our primary application. We demonstrate that fine-tuning the models on this curated dataset can enable SLMs to exceed GPT-4-Turbo's function calling performance. Finally, we enhance the inference efficiency of these fine-tuned models using a novel Tool RAG method and quantization, allowing for efficient edge deployment with real-time responses.

2 Related Work

2.1 Function Calling LLMs

The sophisticated reasoning capabilities of recent LLMs have enabled them to call functions (i.e., tools), where LLMs determine which function to invoke among user-provided functions along with the associated arguments. This allows LLMs to use external functions (e.g. calculators or search engines) to provide more accurate answers to user queries than by responding directly. A pioneering work in this area is Toolformer (Schick et al., 2024), which has inspired various tool-calling frameworks (Ruan et al., 2023; Shen et al., 2024; Liang et al., 2024). ReAct (Yao et al., 2022) introduced a reasoning-and-action process that improved LLMs' interaction with external environments, which has become a back-bone for different open-source frameworks (Liu, 2022; Langchain). More recently, Gorilla (Patil et al., 2023) and ToolLLM (Qin et al., 2023) have demonstrated that an open-source LLM can be fine-tuned to obtain function-calling capabilities in diverse real-world use cases. One noticeable work is Octopus (Chen et al., 2024) which introduces on-device LLMs that invoke software APIs. TinyAgent pushes this boundary by enabling efficient inference via parallel function calling (Kim et al., 2023) as well as a novel tool retrieval method, similar to (Moon et al., 2024). Furthermore, our method does not require any architectural changes, making it compatible with a wider range of open-source models.

2.2 Dataset Synthesis

To address the problem of not having enough data for finetuning, a popular method has emerged to use LLMs to synthesize new training datapoints (Deng et al., 2023; Prasad et al., 2023; Fu et al., 2023; Dai et al., 2023; Ubani et al., 2023; Fang et al., 2023; Liu et al., 2023; Yu et al., 2023; Kumar et al., 2020; Yoo et al., 2021; Wang et al., 2022; Lee et al., 2024b). While these techniques create very good results, they often generate a significant amount of training data. Recent advancements have shown that by filtering these datasets or generating smaller, higher quality datasets, one can achieve similar or better performance (Chen et al., 2023; Cao et al., 2023; Wei et al., 2023; Zhou et al., 2023). TinyAgent builds on these works by constructing a pipeline that systematically generates high-quality, task-specific function-calling datasets, ensuring efficient training and robust performance even with smaller, curated datasets.

2.3 Device Control

Recent advancements in device control have introduced large-scale benchmarks and datasets focused on the Android environment (Rawles et al., 2024b; Zhang et al., 2024b; Rawles et al., 2024a; Lee et al., 2024a), which explore UI-based agents with low-level controls such as typing, scrolling, and tapping. They are primarily concerned with mobile device interactions in simulated environments, but they do not address the challenges of deploying small language models directly on the device, which is crucial for real-world applications where cloud resources are unavailable or impractical. More recently, UFO (Zhang et al., 2024a) introduced a dual-agent framework that leverages vision and language to enable UI-focused agents to operate within Windows OS applications. However, similar to earlier works, UFO also focuses on low-level control mechanisms and does not address the deployment of small language models directly on the device. TinyAgent pushes this boundary by formulating device control as a high-level function-calling problem instead

of low-level UI actions, utilizing task-specific abstractions that allow for more robust and efficient execution of commands. By running fully locally on MacOS, TinyAgent offers a more realistic and practical solution for device control, making it well-suited for real-life scenarios where on-device deployment is necessary.

3 TinyAgent

3.1 Teaching LLMs to do Function Calling

As mentioned above, our main interest is applications where the AI agent translates the user query into a sequence of function calls to complete the tasks. In such applications, the model does not need to write the function definition itself since the functions (or APIs) are mostly pre-defined and already available. Therefore, what the model needs to do is to determine (i) which functions to call, (ii) the corresponding input arguments, and (iii) the right order of calling these functions (i.e. function orchestration) based on the required interdependency across the function calls.

The first question is to find an effective way to equip SLMs to perform function calling. Large models such as GPT-4 are able to perform function calling, but how can this be achieved with open source models? LLMCompiler (Kim et al., 2023) is a recent framework that enables this by instructing the LLM to output a function calling plan that includes the set of functions that it needs to call along with the input arguments and their dependencies (see the example in Figure 1). Once this function calling plan is generated, we can parse it and call each function based on the dependencies.

The critical part here is how to teach the model to create this function calling plan with the right syntax and dependency . The original LLMCompiler (Kim et al., 2023) only considered large models, such as LLaMA-2 70B (Touvron et al., 2023), which have complex reasoning capabilities to create the plan when provided with sufficient instructions in their prompts. Unfortunately, our initial experiments showed that off-the-shelf small models such as TinyLlama-1.1B (Zhang et al., 2024c) (or even the larger Wizard-2-7B model (Vince, 2024)) are not able to output the correct plans when prompted the same way. The errors ranged from problems such as using the wrong set of functions, hallucinated names, wrong dependencies, and inconsistent syntax.

This is rather expected because these small models have been trained on generic datasets and primarily targeted to achieve good accuracy on general benchmarks which mostly test the model's world knowledge and general reasoning or basic instruction following capability. To address this, we explored if fine-tuning these models on a high-quality dataset specially curated for function calling and planning can improve the accuracy of these small language models for a targeted task, potentially outperforming larger models. In Section 3.2, we first discuss how we generated such a dataset, and then we discuss the fine-tuning approach in Section 3.3.

3.2 Dataset Generation

As a driving application, we consider a local agentic system for Apple's Macbook that solves user's day-to-day tasks. Particularly, the agent is equipped with 16 different functions that can interact with different applications on Mac, which includes:

- **Email:** Compose a new email or reply to/forward emails
- **Contacts:** Retrieve phone numbers or email addresses from the contacts database
- **SMS:** Send text messages to contact(s)
- **Calendar:** Create calendar events with details such as title, time, attendees, etc.
- **Notes:** Create, open, or append content to notes in various folders
- **Reminder:** Set reminders for various activities and tasks
- **File management:** Open, read, or summarize documents in various file paths
- **Zoom meetings:** Schedule and organize Zoom meetings

Predefined Apple scripts exist for each of these functions/tools, and all that the model needs to do is to take advantage of the predefined APIs and determine the right function calling plan to accomplish a given task, such as in Figure 1. However, as discussed previously, we need a dataset for training and evaluating SLMs since their off-the-shelf function calling capability is subpar.

Creating handcrafted data with diverse function calling plans is both challenging and not scalable. However, we can curate synthetic data using a powerful LLM like GPT-4-Turbo. Such an

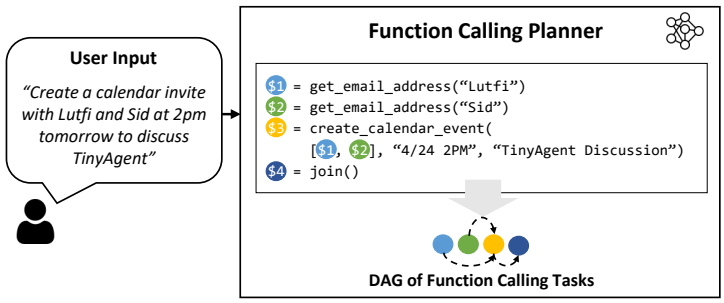

Figure 1: Overview of the LLMCompiler Function Calling Planner. The Planner understands the user query and generates a sequence of tasks with their inter-dependencies. These tasks are then dispatched by the LLMCompiler framework to accomplish the user command. In this example, Task $1 and $2 are fetched together to retrieve the email addresses of Sid and Lutfi independently. After each task is performed, the results are forwarded to Task $3 which creates the calendar event. Before executing Task $3, LLMCompiler replaces the placeholder variables (e.g., the variable $1 and $2 in Task $3) with actual values.

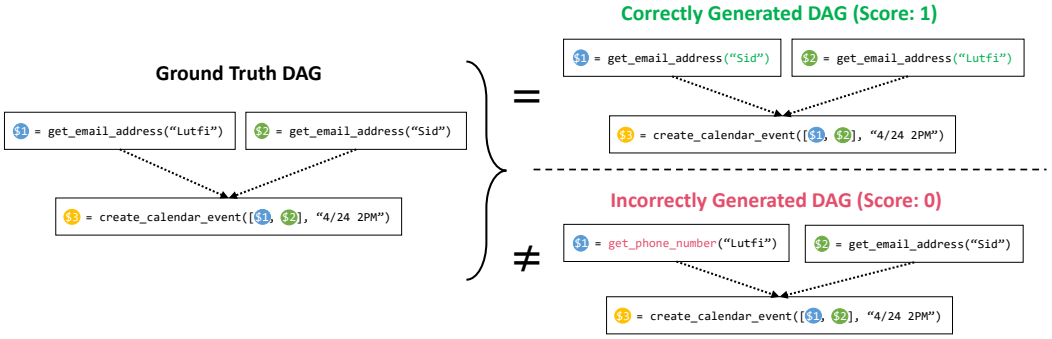

Figure 2: Graph Isomorphism Success Rate. The model scores a success rate of 1 only if the DAG of its generated plan is isomorphic to the DAG of the ground truth plan; and 0 otherwise. In the above example, for the top case, although the order of the `get_email_address` calls are different from the ground truth plan (the ground truth plan gets the email address of Lutfi before Sid, and the generated plan gets the email address of Sid before Lutfi), since the two DAGs are isomorphic to each other, the plan gets 1 success rate. For the bottom case, since the predicted DAG contains a wrong node, corresponding to a wrong function call, the plan gets 0 success rate.

approach is becoming a common method where a capable LLM is instructed to generate data similar to a given set of sample examples or templates. In our work, we used a similar approach, but instead of providing the LLM with generic user queries as templates, we provide it with various sets of functions and instruct it to generate realistic user queries that require those functions to accomplish the task, along with the associated function calling plan and input arguments, like the example shown in Figure 1. To verify the validity of the generated data, we incorporated sanity checks on the function calling plan to make sure that they form a feasible graph, and that the function names and input argument types are correct. With this approach, we created 80K training data, 1K validation data, and 1K testing data, with a total cost of only ~$500.

3.3 Fine-tuning for Improved Function Calling Reasoning

With our dataset in place, we can now proceed to fine-tune off-the-shelf SLMs to enhance their function calling capability. We started with two base small models: TinyLlama-1.1B (instruct-32k) and Wizard-2-7B. For fine-tuning these models, we first need to define a metric to evaluate their performance. Our objective is for these models to accurately generate the right plan, i.e., to select the right set of functions *and* to orchestrate them in the right order. Therefore, we define a success rate metric that assigns 1 if both criteria are met, and 0 otherwise. Checking whether the model has selected the right set function calls is straightforward. To additionally ensure that the orchestration of these functions is correct, we construct a Directed Acyclic Graph (DAG) of the function calls based on the dependencies, as shown in Figure 2, where each node represents a function call and a directed edge from

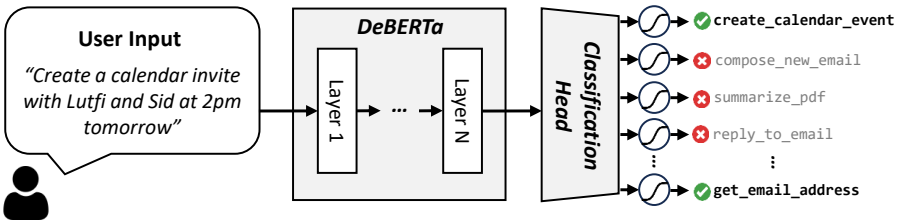

Figure 3: Overview of our Tool RAG scheme. We formulate tool retrieval as a multi-label classification problem. The user query is given as input to the fine-tuned DeBERTa-v3-small model, which outputs a 16-dimensional vector indicating tool probabilities. Tools with probabilities higher than 50% are selected, averaging 3.97 tools per query compared to 6 tools in basic RAG.

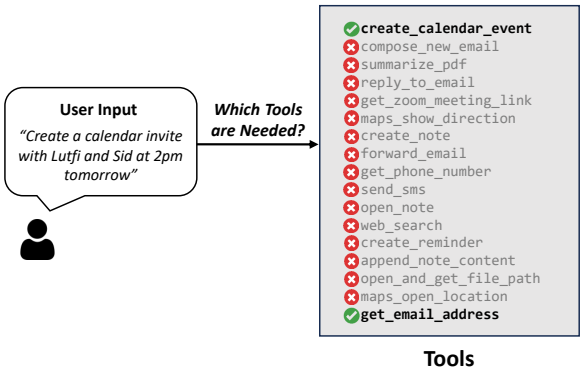

Figure 4: Efficient tool selection based on a user input. Not all user inputs require all available tools; hence, it is imperative to select the right set of tools to minimize the prompt size and increase performance. In this case, the LLM only needs the functions that get email addresses and create a calendar event to accomplish its task.

node A to B represents their interdependency (i.e. function B can only be executed after the execution of function A). Then we compare if this DAG is identical to that of the ground truth plan to verify the accuracy of the dependencies.

After defining our evaluation metric, we applied LoRA (Hu et al., 2021) to fine-tune the models for 3 epochs using a learning rate of 7e-5 over the 80K training examples, and selected the best checkpoint based on validation performance. For fine-tuning, our prompt included not only the descriptions of the ground truth functions (i.e. functions used in the ground truth plan) but also other irrelevant functions as negative samples. We found the negative samples to be particularly effective for teaching the model how to select appropriate tools for a given query, hence improving the post-training performance. Furthermore, we also include several in-context examples demonstrating how queries are translated into a function calling plans. These in-context examples are selected through a Retrieval Augmented Generation (RAG) process based on the user query from the data in the training dataset.

Using the above settings, we fine-tuned TinyLlama-1.1B/Wizard-2-7B models. After fine-tuning, the 1.1B model improved the success rate from 12.71% to 78.89%, and the 7B model performance improved from 41.25% to 83.09%, which is ∼4% higher than GPT-4-Turbo.

3.4 Efficient Inference with Tool RAG

Our primary goal is to be able to deploy the TinyAgent model locally on a Macbook, which has limited computational and memory resources available as compared to the GPUs that closed-source models like GPT are deployed on. To achieve efficient performance with low latency we need to ensure that not only is the model size small, but that the input prompt is as concise as possible. The latter is an important contributor to latency and computational resource consumption due to the quadratic complexity of attention on sequence length.

The fine-tuned TinyAgent model discussed previously was fine-tuned with the description of all available tools in its prompt. However, we can significantly reduce the prompt size by only including the description of relevant tools based on the user query. For instance, consider the example shown in Figure 4 above, where the user is asking to create a calendar invite with two people. In this case, the LLM only needs the functions that get email addresses and create a calendar event in its prompt.

To take advantage of this observation, we need to determine which functions are required to accomplish the user's command, which we refer to as Tool RAG given its similarity with how RAG works. However, the model performs poorly when we use a basic RAG method where we retrieve the relevant tools based on the embedding similarity of the user query and the tools. This is because completing a user's query often requires using several auxiliary tools which may be missed with a simple RAG method if the embedding of the

Table 1: Comparison of TinyAgent performance with DeBERTa to Basic RAG and no RAG settings. For Basic RAG, we retrieved top-3 most relevant tools. For our fine-tuned DeBERTa-v3-small model, we retrieved tools with a probability greater than 50%, which retrieves ∼3.97 tools per query.

Tool RAG Method	Tool Recall	Prompt Size (Tokens)	TinyAgent 1.1B Success Rate (%)	TinyAgent 7B Success Rate (%)
No RAG (all tools in the prompt)	1	2762	78.89	83.09
Basic RAG	0.949	1674	74.88	78.50
Fine-tuned DeBERTa-v3-small (Ours)	**0.998**	**1397**	**80.06**	**84.95**

Table 2: Latency, size, and success rate of TinyAgent models before and after quantization. Latency is the end-to-end latency of the function calling planner, including the prompt processing time and generation.

Model	Weight Precision	Latency (seconds)	Model Size (GB)	Success Rate (%)
GPT-3.5	Unknown	3.2	Unknown	65.04
GPT-4-Turbo	Unknown	3.9	Unknown	79.08
TinyAgent-1.1B	16	3.9	2.2	80.06
	4	**2.9**	**0.68**	**80.35**
TinyAgent-7B	16	19.5	14.5	84.95
	4	**13.1**	**4.37**	**85.14**

auxiliary tool is not similar to the user query. For instance, the example shown in Figure 4 requires calling get_email_address function even though the user query is just asking about creating a calendar invitation.

This can be addressed by treating the problem as a classification of which tools are needed. To that end, we fine-tuned a DeBERTa-v3-small (He et al., 2021) model on the training data to perform a 16-way classification as shown in Figure 3. The user query is given as an input to this model, and then we pass the CLS token at the end through a simple fully connected layer of size 768x16 to transform it into a 16 dimensional vector (which is the total size of our tools). The output of this layer is passed through a sigmoid layer to produce the probability of selecting each tool. During inference, we select the tools that have probably higher than 50%, and if so, we include their description in the prompt. On average we noticed that only 3.97 tools are retrieved with a recall of 0.998, whereas the basic RAG requires using the top 6 tools to achieve a tool recall of 0.968.

We evaluated the model performance after incorporating Tool RAG. The results are shown in Table 1, where we report the performance of the simple RAG system along with the fine-tuned DeBERTa approach. As one can see, the DeBERTa based Tool RAG method achieves almost perfect recall performance, improves the baseline accuracy, while reducing the prompt size by ∼2x tokens.

3.5 Fast Edge Deployment with Quantization

Deploying models at the edge, such as on consumer MacBooks, can still be challenging even for small models with O(1B) parameters, since loading the model parameters can consume a large portion of the available memory. A solution to these issues is quantization, which allows us to store the model at a reduced bit precision. Quantization not only reduces the storage requirements and model footprint, but also cuts down the time and resources needed to load model weights into memory, thereby reducing the overall inference latency as well. For more information on quantization, refer to (Gholami et al., 2022).

To more efficiently deploy the models, we quantized the models into 4-bit with a group size of 32, which is supported by the llama.cpp framework with quantization-aware training. As shown in Table 2, the 4-bit models result in 30% better latency, along with a 4x reduction in the model size. We also notice slight accuracy improvement which is due to the additional fine-tuning with simulated quantization.

4 Putting It All Together

We provide a demo video of the final TinyAgent-1.1B model deployed on a Macbook Pro M3[3], which can be downloaded and tested on Mac

[3] https://www.youtube.com/watch?v=0GvaGL9IDpQ

from the link[4]. It not only runs all of the model inference locally on your computer, but it also allows you to provide commands through audio. We process the audio locally as well using the Whisper-v3 (Radford et al., 2022) model from OpenAI deployed locally using the whisper.cpp framework. The greatest surprise for us was that the accuracy of the 1.1B model exceeds that of GPT-4-Turbo, and is markedly fast while deployed locally and privately on-device.

5 Conclusions

To summarize, we introduced TinyAgent and showed that it is indeed possible to train a small language model and use it to power a semantic system that processes user queries. In particular, we considered a Siri-like assistant for Mac as a driving application. The key components for enabling it is to (i) teach off-the-shelf SLMs to perform function calling through LLMCompiler framework, (ii) curate high quality function calling data for the task at hand, (iii) fine-tune the off-the-shelf model on the generated data, and (iv) enable efficient deployment by optimizing the prompt size through only retrieving the necessary tools based on the user query through Tool RAG, as well as quantized model deployment to reduce inference resource consumption. After these steps, our final models achieved 80.06% and 84.95% for the TinyAgent-1.1.B and 7B models which exceed GPT-4-Turbo's success rate of 79.08% on this task.

6 Ethics Statement

Deploying TinyAgent to operate agentic systems at the edge presents several ethical considerations that are integral to our design and operational philosophy.

Accessibility and Inclusivity: Ensuring that TinyAgent serves all users equitably, including those with disabilities, is a priority. We are committed to designing interfaces that are universally accessible, incorporating features such as voice recognition that can understand diverse speech patterns and text-to-speech technologies that are clear and easily comprehensible. Further, we are exploring adaptive technologies that can adjust to the specific needs of users with varying abilities, ensuring that everyone can benefit from TinyAgent's capabilities without barriers.

Human Oversight: While TinyAgent demonstrates robust capabilities in function calling, the risk of hallucination and erroneous responses by LLMs remains (Zhang et al., 2023). To mitigate this, it is essential to maintain human oversight throughout the operational loop, not just at the endpoint. This means integrating mechanisms for regular checks and balances where humans can review, override, or refine decisions made by TinyAgent. Future iterations of our system will aim to facilitate even more seamless human-agent collaboration to enhance decision accuracy and reliability.

Cultural and Bias Considerations: Synthetic datasets generated using simple or naive prompts often carry inherent biases, such as those related to regional or cultural specificity (Yu et al., 2024). Because task-specific agent systems like TinyAgent rely on synthetic data, their effectiveness and impartiality can be impacted when operating across different demographic landscapes. In response, we integrate diverse cultural data and demographic groups in our data generation processes to mitigate these biases. Our aim is to ensure that the synthetic data fueling TinyAgent is as inclusive and unbiased as possible, supporting a function-calling system that is culturally aware and equitably serves a global user base.

Acknowledgements

We would like to thank Sunjin Choi for his insights in energy cost associated with local and cloud deployment. We acknowledge gracious support from Intel, Furiosa, Apple, and NVIDIA. We also appreciate the support from Microsoft through their Accelerating Foundation Model Research, including great support from Sean Kuno. Furthermore, we appreciate support from Google Cloud, the Google TRC team, and specifically Jonathan Caton, and Prof. David Patterson. Prof. Keutzer's lab is sponsored by the Intel corporation, Intel One-API, Intel VLAB team, the Intel One-API center of excellence, as well as funding through BDD and BAIR. We appreciate great feedback and support from Ellick Chan, Saurabh Tangri, Andres Rodriguez, and Kittur Ganesh. Sehoon Kim would like to acknowledge the support from the Korea Foundation for Advanced Studies (KFAS). Our conclusions do not necessarily reflect the position

[4] https://github.com/SqueezeAILab/TinyAgent/raw/main/TinyAgent.zip

or the policy of our sponsors, and no official endorsement should be inferred.

References

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. 2023. Instruction mining: When data mining meets large language model finetuning. *Preprint*, arXiv:2307.06290.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2023. Alpagasus: Training a better alpaca with fewer data. *Preprint*, arXiv:2307.08701.

Wei Chen, Zhiyuan Li, and Mingyuan Ma. 2024. Octopus: On-device language model for function calling of software apis. *Preprint*, arXiv:2404.01549.

Haixing Dai, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Wei Liu, Ninghao Liu, Sheng Li, Dajiang Zhu, Hongmin Cai, Lichao Sun, Quanzheng Li, Dinggang Shen, Tianming Liu, and Xiang Li. 2023. Auggpt: Leveraging chatgpt for text data augmentation. *Preprint*, arXiv:2302.13007.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. 2023. Rephrase and respond: Let large language models ask better questions for themselves. *Preprint*, arXiv:2311.04205.

Luyang Fang, Gyeong-Geon Lee, and Xiaoming Zhai. 2023. Using gpt-4 to augment unbalanced data for automatic scoring. *Preprint*, arXiv:2310.18365.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning. *arXiv preprint arXiv:2301.12726*.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC.

Google. 2024. Google gemini: Next generation model. Accessed: 2024-07-29.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

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

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. In *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*, pages 18–26.

Langchain. https://github.com/langchain-ai/langchain.

Juyong Lee, Taywon Min, Minyong An, Changyeon Kim, and Kimin Lee. 2024a. Benchmarking mobile device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipali, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024b. Llm2llm: Boosting llms with novel iterative data enhancement. *arXiv preprint arXiv:2403.15042*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing*, 3:0063.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. 2023. Tinygsm achieving 80% on gsm8k with small language models. *Preprint*, arXiv:2312.09241.

Jerry Liu. 2022. LlamaIndex.

Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. 2024. Efficient and scalable estimation of tool representations in vector space. *arXiv preprint arXiv:2409.02141*.

OpenAI. 2024. Hello gpt-4o. Accessed: 2024-07-29.

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*.

Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. 2023. Rephrase, augment, reason: Visual grounding of questions for vision-language models. *Preprint*, arXiv:2310.05861.

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*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *Preprint*, arXiv:2212.04356.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo

Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2024a. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2024b. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Solomon Ubani, Suleyman Olcay Polat, and Rodney Nielsen. 2023. Zeroshotdataaug: Generating and augmenting training data with chatgpt. *arXiv preprint arXiv:2304.14334*.

Amazing Vince. 2024. Not-wizardlm-2-7b. Accessed: 2024-07-29.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Lai Wei, Zihao Jiang, Weiran Huang, and Lichao Sun. 2023. Instructiongpt-4: A 200-instruction paradigm for fine-tuning minigpt-4. *Preprint*, arXiv:2308.12067.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyeong Park. 2021. Gpt3mix: Leveraging large-scale language models for text augmentation. *arXiv preprint arXiv:2104.08826*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *Preprint*, arXiv:2309.12284.

Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2024. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in Neural Information Processing Systems*, 36.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024a. Ufo: A ui-focused agent for windows os interaction. *Preprint*, arXiv:2402.07939.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024b. Android in the zoo: Chain-of-action-thought for gui agents. *Preprint*, arXiv:2403.02713.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024c. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment. *Preprint*, arXiv:2305.11206.