Octo-planner: On-device Language Model for Planner-Action Agents

Wei Chen[1⋆], Zhiyuan Li[1⋆], Zhen Guo[2⋆], and Yikang Shen[3⋆]

[1] Nexa AI & Stanford, USA
{alexchen, zack}@nexa.ai
https://www.nexa.ai
[2] MIT EECS, USA
zguo0525@mit.edu
https://www.mit.edu
[3] MIT-IBM Watson AI Lab, USA
yikang.shen@ibm.com
https://research.ibm.com

Abstract. AI agents have become increasingly significant in various domains, enabling autonomous decision-making and problem-solving. To function effectively, these agents require a planning process that determines the best course of action and then executes the planned actions. In this paper, we present an efficient on-device Planner-Action framework that separates planning and action execution into two components: a planner agent, or Octo-planner, optimized for edge devices, and an action agent using the Octopus model for function execution. Octo-planner first responds to user queries by decomposing tasks into a sequence of sub-steps, which are then executed by the Octopus action agent. To optimize performance on resource-constrained devices, we employ model fine-tuning instead of in-context learning, reducing computational costs and energy consumption while improving response times. Our approach involves using GPT-4 to generate diverse planning queries and responses based on available functions, with subsequent validations to ensure data quality. We fine-tune the Phi-3 Mini model on this curated dataset, achieving a 97% success rate in our in-domain test environment. To address multi-domain planning challenges, we develop a multi-LoRA training method that merges weights from LoRAs trained on distinct function subsets. This approach enables flexible handling of complex, multi-domain queries while maintaining computational efficiency on resource-constrained devices. To support further research, we have open-sourced our model weights at https://huggingface.co/NexaAIDev/octopus-planning. For the demo, please refer to https://www.nexa4ai.com/octo-planner#video.

Keywords: On-device AI · Multi-LoRA training · Large Language model · AI agent

⋆ Equal contribution.

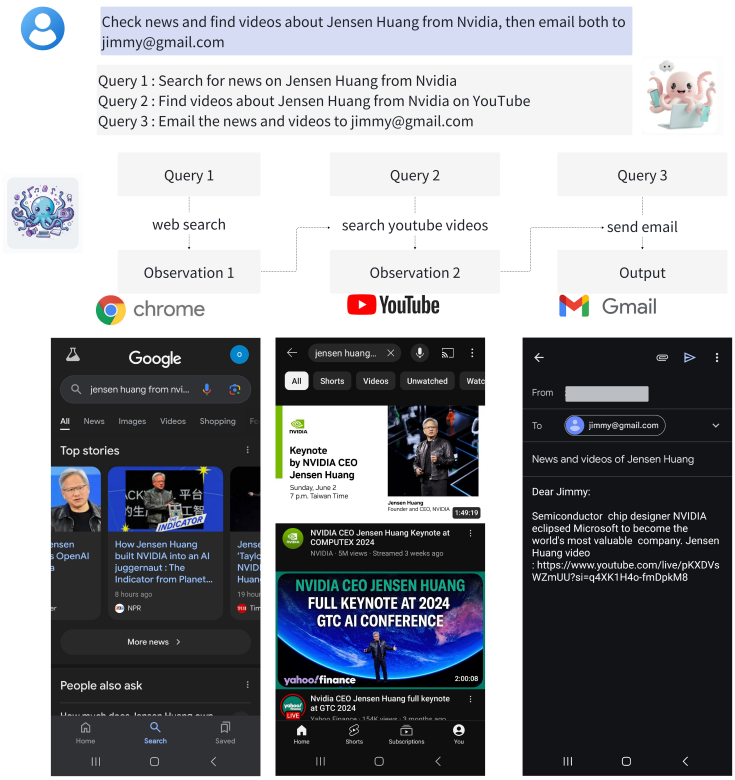

Fig. 1. Planner-Action Agent in smartphone using Octopus models

1 Introduction

Artificial intelligence (AI) agents [1,2] have significantly transformed various industries by enabling autonomous decision-making and improving operational efficiencies [3,4,5,6,7,8,9,10]. These agents rely on a critical planning process that involves determining the optimal course of action, executing the planned actions, and summarizing the outcomes. Large Language Models (LLMs) such as Gemini-Pro [11] and GPT-4 [12] have shown potential in this domain. While these models face challenges in executing complex planning tasks at a level comparable to human performance [13,14], they remain effective in addressing simpler tasks, thereby facilitating practical applications.

One such application is the emergence of AI assistant tools from companies like MultiOn [15], Simular AI [16], and Adept AI [17], which leverage the capabilities of LLMs to provide intelligent assistance across various domains. Additionally, consumer-oriented AI hardware products, such as Rabbit R1 [18], Humane AI Pin [19], and Limitless Pendant [20], integrate LLMs into user-friendly devices, making intelligent assistance more accessible and driving significant traction.

The success of AI agents depends on the performance of the underlying LLMs. Agents using pre-trained models without fine-tuning on tasks demonstrations have relatively low success rates, ranging from 12% on desktop applications [21] to 46% on mobile applications [22], while those leveraging fine-tuned models can achieve up to 80% success rate on tasks similar to their training data [23,24]. However, using LLMs for AI agents is costly due to high computational demands and infrastructure expenses, limiting widespread adoption. The lack of on-device AI agents restricts applications requiring real-time processing, offline functionality, or enhanced privacy.

On-device AI agents offer advantages including reduced latency, offline operation, lower costs, and improved data security [25,27,28,26]. While action models like Octopus V2 achieve over 95% accuracy for function calling [29], an on-device planning model is still missing. General agent frameworks use single-model in-context learning, requiring lengthy function descriptions and planning instructions in each prompt. This approach is impractical for on-device models with limited context lengths, causing high latency and battery consumption on edge devices.

In this paper, we introduce Octo-planner, an on-device planning agent that addresses the key challenges of efficiency, adaptability, and resource constraints. Our Planner-Action framework separates planning and action execution into two components: a planner agent, or Octo-planner, optimized for edge devices, and an action agent using the Octopus model for function execution. By prioritizing fine-tuning over few-shot prompting, we reduce computational costs and minimize key-value (KV) cache requirements. Our approach uses GPT-4 to generate and validate planning data, which is then used to fine-tune Phi-3 Mini for on-device deployment. In-domain tests demonstrate that this fine-tuning improves planning success rates to 97%. To address multi-domain planning challenges, we develop a multi-LoRA training method that merges weights from LoRAs trained on distinct function subsets. This enables flexible handling of complex, multi-domain queries while maintaining computational efficiency on resource-constrained devices. By focusing on pre-defined functions for simpler tasks and leveraging fine-tuning, we aim to make AI agents more practical, accessible, and cost-effective for real-world applications.

This work aims to contribute to the ongoing efforts to make AI more accessible and practical for everyday use. By bridging the gap between AI agent potential and edge computing constraints, we seek to facilitate the adoption of intelligent, on-device assistants across various domains. Through open-sourcing our approach, we hope to inspire further innovations in on-device AI, expanding the reach of advanced planning capabilities to a broader range of applications.

2 Related Works

2.1 Planner Agent

Language models have become essential in planning agent systems. Proprietary models like OpenAI's assistant API [52] excel in generating strategies based on

user queries and available functions. Recent advancements have further expanded the capabilities of language models in planning. The ReAct framework [33] integrates planning and acting for limited action spaces, while research from Alibaba Group [34] highlights the effectiveness of separate planning and action models for complex tasks. In robotics, language models are also increasingly applied to task-level planning [35,36]. Notable examples include SayCan [37], which uses LLMs to break high-level tasks into concrete sub-tasks, and Video Language Planning (VLP) [38], which enhances long-horizon planning through a text-to-video dynamics model.

2.2 Fine-Tuning to Replace Long Context

Fine-tuning language models to internalize specific prompts or context information reduces input length and improves efficiency [39,40]. This approach involves training models on carefully curated, task-specific datasets. For models with limited context windows, this technique is particularly valuable as it enables more efficient query processing without sacrificing response quality. The success of fine-tuning largely depends on the use of diverse, high-quality datasets, which ensure the model can generalize across various prompt phrasings [41,42,43,44]. When implemented effectively, fine-tuning streamlines application-specific interactions, addressing both context length limitations and computational challenges in practical deployments.

2.3 LoRA and Multi-LoRA

Low-Rank Adaptation (LoRA) efficiently adapts pre-trained language models to specific tasks [45]. Unlike fine-tuning, which updates all parameters, LoRA freezes pre-trained weights and adds trainable low-rank matrices to each layer, significantly reducing trainable parameters and computational demands. Multi-LoRA extends this concept by enabling multiple task-specific adapters to be trained, combined, or switched during inference, allowing a single base model to handle various tasks efficiently [46]. Building on these approaches, researchers have developed several related variants of the original LoRA to address different aspects of model adaptation: LoRA+ optimizes learning rates [47], VeRA uses random projections [48], AdaLoRA implements adaptive rank [49], DoRA decomposes weights [50], and Delta-LoRA updates pretrained weights [51]. These variations aim to further refine efficiency or performance in specific scenarios.

3 Method

This section presents our framework for on-device Planner-Action agents. We first describe the integration of planning and action agents for efficient problem-solving. We then detail our approach to dataset design and the training process for the planning agent, including support for extensive functions and a plug-and-play capability for additional function sets. Finally, we outline our benchmark used to evaluate agent performance.

3.1 Planner and Action Agents Framework

Our Planner-Action approach distinguishes itself from general agent frameworks by separating the planning and action execution processes into two components. This separation improves modularity and allows for specialized optimization of each component. The framework operates as follows.

Planner Phase: Given a user query q, our planning model π_{plan} decomposes the task into a sequence of sub-steps. Formally:

$$\{\tau_1, \tau_2, \ldots, \tau_n\} = \pi_{\text{plan}}(q; F) \tag{1}$$

where F is the set of available function descriptions, and τ_i is the i^{th} execution step. π_{plan} internalizes F during instruction fine-tuning.

Action Phase: For each step in the execution sequence, we employ an action model π_{action}. At step i, given the observation of the current state O_i, the action model performs:

$$O_{i+1} = \pi_{\text{action}}(\tau_i, O_i) \tag{2}$$

where O_{i+1} and τ_{i+1} are passed to the next step for continued execution.

For the action model, we utilize the Octopus model, which is specifically designed for on-device function calling. Figure 2 illustrates the difference between our Planner-Action framework and the single-model approach for LLM agents.

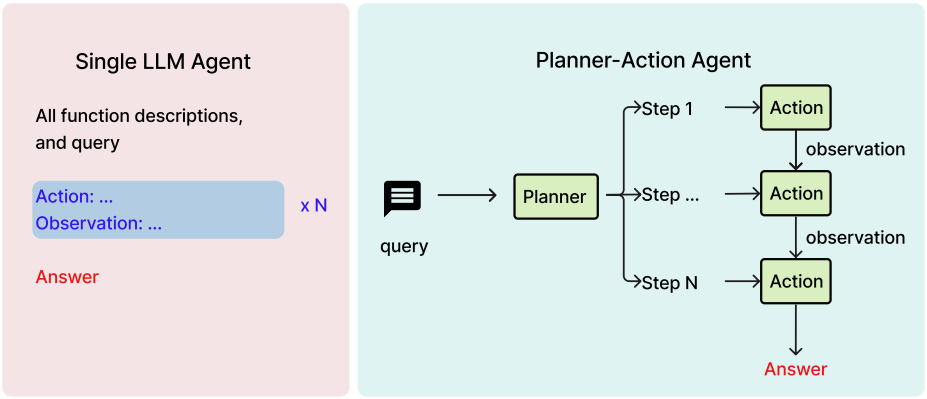

Fig. 2. Comparison of Single LLM Agent and Planner-Action Agent frameworks. (Left) Single LLM Agent: A unified model performs both task planning and action execution. (Right) Planner-Action Agent: A specialized planner model decomposes the task into subtasks, while a separate action model executes each subtask sequentially.

The modular design of our framework offers several advantages:

– **Specialization**: Separating planning and action execution allows optimization of each model for its specific role, enhancing performance in complex tasks.

- **Scalability**: Independent scaling of planning and action capabilities efficiently accommodates varying task complexities.
- **Interpretability**: Explicit separation of phases improves transparency in the decision-making process.
- **Adaptability**: Easier integration of domain-specific knowledge or constraints into either phase without system-wide changes.

3.2 Planning Dataset

Our framework uses the Octopus model as the action model, requiring training only for the planner agent. We fine-tune the planner agent with the following dataset format:

<|user|>{user's query}<|end|>

<|assistant|> `{query`$_1$`}`<nexa_split>`{query`$_2$`}`<nexa_split>...
<nexa_split>`{query`$_n$`}`.<|end|>

Special tokens like `<|user|>` and `<|assistant|>` are used for chat model pretraining but are optional otherwise. We set n as 1-5, based on our finding that most of tasks on mobile app consist of fewer than 5 steps. The dataset generation and curation process includes:

1. **Dataset collection**: Given the available functions F, we use a large language model (GPT-4) to generate diverse queries answerable by these functions. We increase the model's temperature setting to ensure query variety. Responses are then generated in the specified dataset format. Importantly, while function descriptions are used during generation, they are not included in the final dataset. Instead, the planner model internalizes this function information during training.
2. **Data validation**: We employ the same language model as a validation tool to assess the correctness of query-response pairs. Despite some errors in the initial generation process, we found that the model effectively classifies generated content as valid or invalid, allowing us to filter out incorrect outputs and maintain dataset quality.

Example data points with different number of sub-steps are shown below:

Planning Dataset example:

query: Tell me today's stock market.
response: Search for today's stock market news and latest updates<|end|>

Two-step example:

query: Find and email me Jensen Huang's news in English.
response: Find articles about Jensen Huang in English<nexa_split> Email the first article found to myself<|end|>

Three-step example:

query: Find and increase the screen brightness by 20% to better view the quarterly report document before connecting to the office printer via Bluetooth.
response: Find the quarterly report document in the system<nexa_split> Increase screen brightness by 20% to improve visibility<nexa_split> Connect to the office printer via Bluetooth to print the quarterly report<|end|>

Four-step example:

query: Find and email the project proposal document to project.manager@company.com, then look up the submission deadline and schedule it in my calendar with a reminder.
response: Locate the project proposal document<nexa_split> Send the project proposal via email to the project manager at project.manager@company.com<nexa_split> Find the project proposal submission deadline<nexa_split> Schedule a calendar event for the project proposal submission deadline with a reminder<|end|>

Five-step example:

query: I have a meeting tomorrow morning, please find my presentation then connect to the conference room projector via Bluetooth. After that, increase the screen brightness then take a screenshot of the final summary slide and email it to all participants.
response: Find the presentation for the meeting tomorrow<nexa_split> Connect to the conference room projector via Bluetooth<nexa_split> Increase screen brightness by 20%<nexa_split> Take a screenshot of the final summary slide<nexa_split> Email the screenshot to all meeting participants<|end|>

For the visualization of the dataset collection, please see Figure 3. Example function descriptions are in Appendix 8.1.

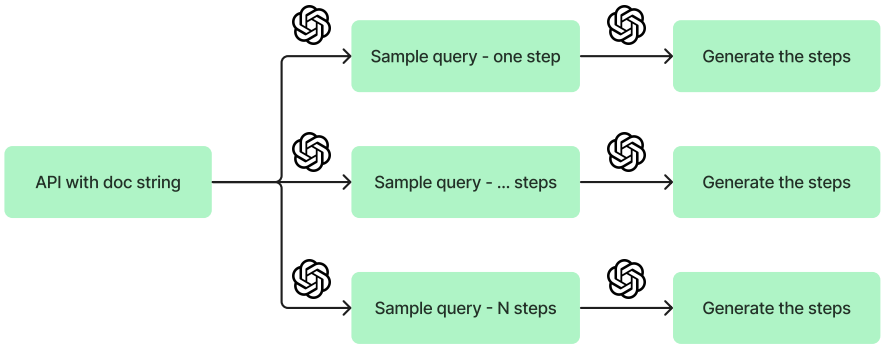

Fig. 3. Dataset Collection Process for Planner Model Training. We generate queries that require between 1–5 steps, verify correctness, and retain only high-quality samples.

3.3 Benchmark Design

Our evaluation relies on a carefully constructed test dataset. This dataset is designed to represent the complexities of real-world planning, employing a multi-stage approach that integrates automated generation, expert validation, and empirical testing.

The process begins with the automated generation of an initial dataset comprising 1,000 data points using GPT-4. These data points then undergo a rigorous quality assurance process to ensure their integrity and relevance. The quality assessment criteria are as follows:

- Each step must correspond to an existing function;
- The sequential order of steps must be correct.

To ensure the reliability of our evaluation, we incorporate an additional phase of manual verification. This phase involves selecting a subset of examples for end-to-end model execution, thereby validating the accuracy of our results and providing a comprehensive assessment of our model's performance.

For the evaluation of our proposed planning model, we employ GPT-4 as an oracle to determine the correctness of the generated plans. This choice is based on empirical observations indicating GPT-4's high proficiency in our specific use case.

4 Experimental Design

Our experimental design assesses the Octo-planner's performance for on-device AI agent planning. We aim to determine the optimal configuration for deploying

efficient, accurate planning models on resource-constrained devices while maintaining adaptability to new domains and functions. Our experiments focus on four key areas:

1. Performance and efficiency trade-offs between full fine-tuning and LoRA.
2. Multi-LoRA accuracy in handling different function sets simultaneously.
3. Performance comparison across various base models and sizes.
4. The impact of dataset size on accuracy, ranging from 100 to 1000 training examples.

We conduct supervised fine-tuning on our curated dataset, using Phi-3 Mini and a few other alternatives as the base model. Training includes both full fine-tuning and LoRA techniques. For all experiments, we set the dataset size to be 800 times the number of available functions, and the multiplier was determined empirically through initial pilot studies. We perform fine-tuning on an NVIDIA A100 GPU using optimized hyperparameters for both approaches: a learning rate of 5×10^{-6}, batch size of 4, a warm-up ratio of 0.2, and 2 training epochs. For LoRA, we set the `target_modules` to be `all-linear`.

5 Results

5.1 Full Fine-Tuning vs. LoRA

Table 1 presents a detailed comparison of full fine-tuning and LoRA approaches for our planning model. Our experiments reveal significant differences in performance across these methods. Full fine-tuning achieves the highest accuracy at 98.1%, demonstrating superior performance. In contrast, LoRA performance depends on rank size. With rank 64 and alpha 256, LoRA achieves 85.1% accuracy, while reducing to rank 16 and alpha 32 drops accuracy to 72.9%. These results highlight the trade-off between model performance and computational efficiency when using LoRA. While full fine-tuning provides better accuracy, LoRA offers a more resource-efficient alternative, with performance varying based on rank configuration.

Table 1. Full Fine-Tuning vs. LoRA

Training Configuration	Benchmark Accuracy
Full model training	98.1%
LoRA (rank=64, alpha=256)	85.1%
LoRA (rank=16, alpha=32)	72.9%

5.2 Multi-LoRA Training and Merging

While LoRA-based training is effective for specific function sets, real-world applications often require handling new or expanded sets of functions. To address this challenge, we propose merging multiple LoRA weights, each trained on distinct function subsets, with the same base model. This approach creates a composite model that combines knowledge from various function sets, offering a scalable solution for complex, multi-domain queries in resource-constrained environments.

To evaluate this method, we constructed a benchmark dataset by randomly selecting functions from each LoRA's domain with equal probability, combining them into workflows. Queries and plans were generated using GPT-4. For example, when testing two merged LoRAs, queries may involve Android functions, E-commerce functions, or both, with equal likelihood.

The code block below shows sample queries from our benchmark dataset and the corresponding inference results from the multi-LoRA merged model:

Android functions only:

query: Find the quarterly report for Q3 and print it for the upcoming board meeting.
response: Locate the quarterly report document for Q3<|nexa_split|>Print the located quarterly report document<|end|>

E-commerce functions only:

query: Find ergonomic office chairs and then buy the selected one using my default shipping address and credit card
response: Search for ergonomic office chairs<nexa_split>Purchase the selected ergonomic office chair using the default shipping address and credit card<|end|>

Android and E-commerce functions:

query: Find ergonomic office chair list doc and print it for my shopping
response: Search for ergonomic office chair list doc<nexa_split>Print the ergonomic office chair list doc <|end|>

Table 2 presents the performance results of our multi-LoRA merging technique. Each individual LoRA was trained with consistent hyperparameters: rank 64, lora_alpha 256, and target_modules set to "all-linear". The single-domain Android function set LoRA achieves 85.1% accuracy. When merging LoRAs from two domains (Android and E-Commerce), accuracy slightly decreases to 82.2%. Further merging yields lower accuracies: 78.9% for three domains (adding Video Streaming), and 69.7% for four domains (adding Travel). These results reveal a pattern of gradual accuracy decline as we integrate more function sets, with a steeper drop occurring after the third domain is added.

Table 2. Multi-LoRA Benchmark

Training Configuration	Benchmark Accuracy (%)
LoRA for Android	85.1
Merged for Android, E-Commerce	82.2
Merged for Android, E-Commerce, video streaming	78.9
Merged for Android, E-Commerce, video streaming, travelling	69.7

5.3 Full Fine-Tuning with Different Base Models

We also tested multiple base models in full fine-tuning. Table 3 shows that Google Gemma 2b achieved 85.6% accuracy, Gemma 7b reached 99.7%, and Microsoft Phi-3 Mini got 98.1%. The results indicate that while smaller models can be adapted quickly, larger models generally achieve higher accuracy.

Table 3. Different Base Model Benchmark

Base Model	**Benchmark Accuracy**
Google Gemma 2b	85.6%
Google Gemma 7b	99.7%
Microsoft Phi-3 Mini	98.1%

5.4 Full Fine-Tuning with Different Dataset Sizes

Our default training dataset has 1,000 data points, with an even distribution across 1–5 step sequences. We investigate how dataset size affects performance, balancing the cost of synthetic data generation with accuracy. Table 4 shows the results. Accuracy grows with dataset size, suggesting that at least 1,000 data points is recommended for robust performance.

Table 4. Dataset Size Benchmark

Training Dataset Size	**Benchmark Accuracy**
1,000 data points	98.1%
500 data points	92.5%
250 data points	85.3%
100 data points	78.1%

6 Conclusion

We presented the Octo-planner, an on-device planning agent designed to work alongside action agents like Octopus V2. By separating planning and action

execution, we improve specialization and adaptability. Our approach fine-tunes Phi-3 Mini (a 3.8B parameter LLM) to serve as a planning agent capable of running locally on edge devices, achieving a 97% success rate in in-domain tests. We reduce computational demands, improve latency, and introduce a multi-LoRA approach for easily expanding model capabilities without full retraining.

This contribution addresses key deployment concerns—data privacy, latency, and offline functionality—moving toward practical, sophisticated AI agents that function entirely on-device. By open-sourcing our model weights, we hope to spur innovation in on-device AI, advancing efficient and privacy-protecting applications that serve everyday needs.

7 Limitations and Future Work

While our model is effective for specific mobile phone use cases, it lacks the iterative refinement seen in frameworks like ReAct [33], which alternate between planning steps and real-time execution based on feedback. Our upfront planning approach is efficient for straightforward tasks but may be less adaptable to changing conditions mid-execution.

Future work will explore incremental plan refinement using real-time observations to handle dynamic environments more effectively. We also intend to integrate our planning model with a broader range of action models, extending beyond mobile applications to IoT and robotics. These steps aim to address current limitations and broaden the versatility of on-device planning.

Acknowledgments. We thank the Nexa AI and MIT community for valuable feedback and support.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Jennings, N.R., Wooldridge, M.: Applications of intelligent agents. In: Agent Technology. Springer, pp. 3–28 (1998)
2. Poole, D.L., Mackworth, A.K.: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press (2010)
3. Kim, B., Gao, J., He, J., et al.: Language models are multilingual chain-of-thought reasoners. arXiv preprint arXiv:2309.04958 (2023)
4. Deng, B., Xiao, W., Lin, X., et al.: Mind2web: Toward a generalist agent for the web. arXiv preprint arXiv:2308.09230 (2023)
5. Yan, S., Wang, W., Fang, Y., et al.: Exploring gpt-4v as a multi-modal agent for vision-based tasks. arXiv preprint arXiv:2310.12979 (2023)
6. Zheng, W., Song, W., Liu, X., et al.: GPT-4V in the loop: Multimodal chain-of-thought reasoning for web navigation tasks. Under review (2024)
7. Koh, S., Wang, Y., Xu, A., et al.: VisualWebArena: A benchmark for vision-based web manipulation. Under review (2024)

8. Chen, W., Li, Z., Ma, M.: Octopus: On-device language model for function calling of software APIs. arXiv preprint arXiv:2404.01549 (2024)
9. Xu, J., Li, Z., Chen, W., et al.: On-Device Language Models: A Comprehensive Review. arXiv preprint arXiv:2409.00088 (2024)
10. Chen, W., Li, Z.: Octopus v4: Graph of language models. arXiv preprint arXiv:2404.19296 (2024)
11. Gemini Team: Gemini: Next-generation large language model from Google. Under review (2024)
12. OpenAI: GPT-4 Technical Report. (2024) https://openai.com/research/gpt-4
13. Xie, Q., Zhu, W., Li, D., et al.: Evaluating LLM-based agent for real-world travel itinerary planning. Under review (2024)
14. Zheng, X., Chen, Y., Wu, S., et al.: The limit of LLM's planning ability for real-world tasks. arXiv preprint arXiv:2401.01234 (2024)
15. MultiOn Team: MultiOn: An AI assistant for multi-step tasks. https://www.multion.ai (2024)
16. Simular AI Team: Simular: AI in everyday workflows. https://simular.ai (2024)
17. Adept Team: Adept AI. https://www.adept.ai (2024)
18. Rabbit Team: Rabbit R1: On-device AI phone. https://www.rabbitphone.com (2024)
19. Humane: Humane AI Pin. https://hu.ma.ne (2024)
20. Limitless Lab: Limitless Pendant. https://www.limitlessai.com (2024)
21. Xie, Z., Gao, X., Yuan, Y., et al.: OSWorld: Evaluating LLM-based agent for OS-level tasks. Under review (2024)
22. Bishop, W., Song, Y., Chen, R.: Latent planning in LLM-based mobile agents. Under review (2024)
23. Nakano, R., Hilton, J., Balaji, S., et al.: WebGPT: Browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332 (2022)
24. Gur, I., Friedman, E., Heilbron, F.C., et al.: Real-world agent tasks: A fine-tuned LLM approach. Under review (2024)
25. Yu, H., Wang, M., Chen, J., et al.: A survey on security issues of on-device AI. IEEE Communications Surveys & Tutorials (2017)
26. Lin, X., Gan, Y., Xu, A., et al.: AWQ: Activation-aware weight quantization for LLMs. Under review (2024)
27. Alwarafy, A., Barnawi, A., et al.: A survey on IoT-based edge computing for streaming data analytics. IEEE Access 8 (2020)
28. Ranaweera, P., Ramasamy, S., Jayarathna, S.: A survey on security in edge computing. IEEE Access 9 (2021)
29. Chen, W., Li, Z., Guo, Z., Shen, Y.: Octopus v2: On-device function calling with 95% accuracy. Under review (2024)
30. Chen, W., Li, Z.: Octopus v3: Technical Report for On-device Sub-billion Multimodal AI Agent. arXiv preprint arXiv:2404.11459 (2024)
31. Chen, W., Li, Z., Xin, S.: OmniVLM: A Token-Compressed, Sub-Billion-Parameter Vision-Language Model for Efficient On-Device Inference. arXiv preprint arXiv:2412.11475 (2024)
32. Chen, W., Li, Z., Xin, S., et al.: Squid: Long Context as a New Modality for Energy-Efficient On-Device Language Models. arXiv preprint arXiv:2408.15518 (2024)
33. Yao, S., Yu, P.S., Wang, S.: ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2303.04671 (2023)
34. Shen, Y., Chen, W., Zhuang, T., et al.: Small LLM, Big Skills: A multi-agent approach for complex tasks. Under review (2024)

35. Hu, Z., Zhu, Y., Farn, C., et al.: General-purpose robotics via large language models. arXiv preprint arXiv:2307.01233 (2023)
36. Firoozi, T., Lan, R., Magnus, G., et al.: Foundation models in robotics: A survey. arXiv preprint arXiv:2306.03306 (2023)
37. Ahn, M., Chen, D., et al.: Do as I can, not as I say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691 (2022)
38. Du, Y., Zhang, S., Kemp, C., et al.: VLP: Video language planning for long-horizon tasks. arXiv preprint arXiv:2309.10024 (2023)
39. Lester, B., Al-Rfou, R., Constant, N.: The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691 (2021)
40. Li, X.L., Liang, P.: Prefix-tuning: Optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190 (2021)
41. Paul, M., Kolesnikov, A., Reinhard, E., et al.: Deep training data curation. arXiv preprint arXiv:2305.01012 (2023)
42. Cao, J., Giannini, S., Guo, D., et al.: Instruction-tuning data generation for domain-specific tasks. arXiv preprint arXiv:2306.14175 (2023)
43. Xia, T., Wei, Y., Huang, H., et al.: Less is more: Efficient sample generation for large instruction tuning. Under review (2024)
44. Wang, Y., Zhang, T., Kong, Y., et al.: On the importance of diverse synthetic data for instruction tuning. Under review (2024)
45. Hu, E.J., Shen, Y., et al.: LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685 (2021)
46. Wang, R., Zhu, Y., Zhang, R.: Multi-LoRA: Fine-tuning large language models across domains. arXiv preprint arXiv:2305.12345 (2023)
47. Hayou, S., Gurram, A., Ingold, R.: LoRA+: Learning rate aware LoRA for efficient adaptation. Under review (2024)
48. Kopiczko, P., Foltyn, R., et al.: VeRA: Very efficient random adaptation. Under review (2024)
49. Zhang, X., Jiang, Y., et al.: AdaLoRA: Fine-tuning large language models with adaptive rank. arXiv preprint arXiv:2303.11527 (2023)
50. Liu, K., Hayes, T., Barnett, M.: DoRA: Decomposed rank adaptation for parameter-efficient tuning. Under review (2024)
51. Zi, W., Wei, J., Lin, Z., et al.: Delta-LoRA: Revisiting the tradeoff between model adaptation and stability. arXiv preprint arXiv:2310.14257 (2023)
52. OpenAI: OpenAI Assistant API Overview. `https://platform.openai.com/docs/guides/gpt` (2023)

8 Appendix

8.1 Function description examples

```
def get_trending_news(query, language):
    """
    Retrieves a collection of trending news articles
    relevant to a specified query and language.

    Parameters:
    - query (str): Topic for news articles.
    - language (str): ISO 639-1 language code. The default
      language is English ('en'), but it can be set to any
      valid ISO 639-1 code (e.g., 'es', 'fr').

    Returns:
    - list[str]: A list of strings, where each string
      represents one news article (title and URL).
    """

def get_weather_forecast(location):
    """
    Provides a weather forecast for a specified location.

    Parameters:
    - location (str): Location for the forecast (city name,
      ZIP code, etc.).

    Returns:
    - list[str]: A list of daily forecasts, each containing
      the date and a brief description of the weather.
    """

def send_email(recipient, title, content):
    """
    Sends an email to a specified recipient.

    Parameters:
    - recipient (str): Recipient's email address.
    - title (str): The subject line of the email.
    - content (str): The body content of the email.

    Returns:
    """

def search_youtube_videos(query):
    """
    Searches YouTube for videos matching a query.

    Parameters:
```

```
    - query (str): Search query.

    Returns:
    - list[str]: Each element includes the video name and URL.
    """

def find_route_google_maps(origin, destination, mode):
    """
    Computes a route using Google Maps from origin to destination.

    Parameters:
    - origin (str): Starting location.
    - destination (str): Target location.
    - mode (enum): 'driving', 'walking', 'bicycling', or 'transit'.

    Returns:
    - list[str]: Route details, including directions and distances.
    """

def send_text_message(contact_name, message):
    """
    Sends a text message to a specified contact.

    Parameters:
    - contact_name (str): The name of the recipient contact.
    - message (str): The content of the message.

    Returns:
    """

def create_contact(name, phone_number):
    """
    Creates a new contact in the device's address book.

    Parameters:
    - name (str): Full name of the contact.
    - phone_number (str): Phone number (preferably in E.164 format).

    Returns:
    """

def set_timer_alarm(time, label):
    """
    Sets a timer or alarm for a specified time.

    Parameters:
    - time (str): Alarm time in "HH:MM" (24-hour format).
    - label (str): Custom label (default is "alarm").

    Returns:
```

```
    """

def create_calendar_event(title, start_time, end_time):
    """
    Schedules a new event in the calendar.

    Parameters:
    - title (str): Event title.
    - start_time (str): ISO 8601 format (YYYY-MM-DD-HH-MM).
    - end_time (str): ISO 8601 format; must be after start_time.

    Returns:
    """

def set_volume(level, volume_type):
    """
    Sets the volume level for a specified type ('ring', 'media', 'alarm').

    Parameters:
    - level (int): 0 (mute) to 10 (max).
    - volume_type (enum): One of 'ring', 'media', 'alarm'.

    Returns:
    """
```