

CAMPHOR: Collaborative Agents for Multi-input Planning and High-Order Reasoning On Device

Yicheng Fu*
Stanford University

Raviteja Anantha†
Amazon

Jianpeng Cheng†
Meta AI

Abstract

While server-side Large Language Models (LLMs) demonstrate proficiency in function calling and complex reasoning, deploying Small Language Models (SLMs) directly on devices brings opportunities to improve latency and privacy but also introduces unique challenges for accuracy and memory. We introduce CAMPHOR, an innovative on-device SLM multi-agent framework designed to handle multiple user inputs and reason over personal context locally, ensuring privacy is maintained. CAMPHOR employs a hierarchical architecture where a high-order reasoning agent decomposes complex tasks and coordinates expert agents responsible for personal context retrieval, tool interaction, and dynamic plan generation. By implementing parameter sharing across agents and leveraging prompt compression, we significantly reduce model size, latency, and memory usage. To validate our approach, we present a novel dataset capturing multi-agent task trajectories centered on personalized mobile assistant use-cases. Our experiments reveal that fine-tuned SLM agents not only surpass closed-source LLMs in task completion F1 by 35% but also eliminate the need for server-device communication, all while enhancing privacy.

1 Introduction

Server-side Large Language Models (LLMs) are powerful semantic parsers that interpret user intent and map queries to executable function calls. To ground a query within a personal environment such as an open toolbox, retrieval-augmented generation (RAG) (Borgeaud et al., 2022) can be adopted to pre-fill the LLM prompt with external knowledge relevant to the user query, such as top-K most relevant tools. An orthogonal strategy is long-context language modeling (Beltagy et al., 2020; Zaheer

et al., 2020), which pre-loads the prompt with all available external knowledge, taking advantage of a larger context window up to 128K tokens (Dubey et al., 2024). However, a server-side LLM is not optimal for a mobile assistant due to privacy and latency concerns.

Privacy. User queries to mobile assistants are often ambiguous, making it crucial to ground them in personal information, such as contacts, installed tools, and past activities. While the assistant needs access to personal data to improve understanding, it must also prioritize user privacy by keeping sensitive information on the device. Even private cloud solutions are not ideal for this, as they commonly avoid storing user-specific data, which prevents KV caching in multi-turn dialogues (Li et al., 2024).

Latency. In addition to the limited flexibility of KV caches due to privacy concerns, server-side LLMs introduce extra latency between understanding (which occurs on the server) and execution (which happens on the user’s device). This latency can degrade the user experience, particularly for solutions requiring multiple server-device round trips. For instance, the ReAct framework (Yao et al., 2023), which breaks down the understanding task into multiple steps and reasons over intermediate execution results, is hindered by this added latency.

A more effective solution to address latency and privacy concerns is to deploy a small language model (SLM) on-device, allowing it to process personal data and interpret user queries locally. However, SLMs introduce new challenges related to accuracy due to tighter budgets on prompts and KV caches. While a RAG approach is scalable across personal databases, the retriever is a separate model which gates the performance of the language model since it is impossible to achieve a perfect recall given the fixed prompt budget (Fan et al., 2024). The problem becomes worse for compositional queries which are naturally harder for

* Work done as an intern at Apple.

† Research done while at Apple.

retrieval. Consider the following example:

Can you show me the cheapest flight options to Barcelona next month and add it to my calendar? Also, let my travel buddy know about our trip plan.

The query is both compositional and includes a personal entity (my travel buddy). Successfully retrieving the appropriate tool requires task decomposition, which depends on the granularity of the available toolbox (i.e., the tools present on the user’s device). While long-context modeling allows the language model to consider all relevant tools and personal entities during query parsing, it also rapidly exhausts the prompt budget, increases KV cache overhead, and introduces a "needle-in-a-haystack" challenge (Liu et al., 2024) for the SLM.

We argue that a key to successful on-device understanding is an agentic approach that decomposes the understanding task in functional space, proactively seeks personal information and runtime feedback before generating the final function call. Specifically, we propose CAMPHOR, an SLM-based collaborative agent understanding system for grounded query parsing. CAMPHOR features a hierarchical architecture, with a high-order reasoning agent that incrementally solves a query by orchestrating sub-agents. Each sub-agent is responsible for a unique functionality, such as retrieving personal entities, fetching latest user activities, and calling task-completion APIs. We apply tool compression techniques that enable the SLM to reason over device-specific toolboxes while satisfying prompt and memory budgets. Overall, CAMPHOR is designed to offer optimal accuracy and speed, while protecting user data on device.

We release the CAMPHOR dataset¹, which consists of multi-agent execution trajectories focused on mobile assistant use cases (as shown in Figure 1). The dataset is unique in how it simulates user device states and ground query understanding to personal information. We fine-tune the SLM-based CAMPHOR agent on this dataset, and prove that fine-tuned SLMs can outperform closed-source LLMs in terms of task completion metrics, while eliminating server-device communication costs and safeguarding user privacy.

¹The dataset can be found at <https://github.com/sofyc/CAMPHOR>.

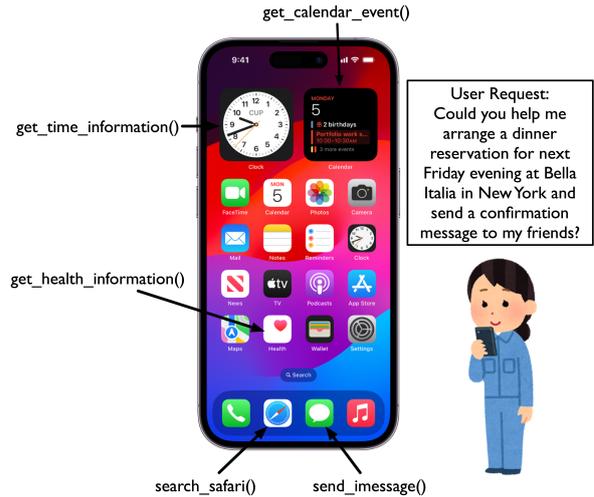


Figure 1: CAMPHOR dataset simulates a user’s smartphone environment, encompassing diverse personal information stored across multiple apps on the device.

2 Related Work

Small Language Models and On-device Agents demonstrate benefits of faster inference, lower latency, and enhanced privacy protection. Studies show that SLMs like Mistral (Jiang et al., 2023), Phi (Abdin et al., 2024), TinyLlama (Zhang et al., 2024), MobileLLM (Liu et al.), MiniCPM (Hu et al., 2024), and Gemma (Team et al., 2024), when fine-tuned for specific tasks, can outperform prompting Large Language Models (LLMs). In particular for the function calling task, the Octopus series (Chen et al., 2024b) has achieved remarkable accuracy, exceeding 97% for function calling on device. Most related to our work is the on-device Octo-planner (Chen et al., 2024a), which breaks down a query into multiple subqueries for function call generation. However, we argue that query decomposition in natural language space is an unconstrained optimization problem, as the granularity of decomposition depends on the available toolset.

Multi-agent Planning is the process where multiple agents, each with unique capabilities, knowledge, and objectives, work together towards shared or interrelated goals. The rise of large language models (LLMs) has significantly advanced the development of multi-agent planning, as tasks for each agent can potentially be solved through prompts. Agentic frameworks like ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2024), LATS (Zhou et al.), SwiftSage (Lin et al., 2024), and AUTOACT (Qiao et al.) continuously prompt LLMs to reflect on and critique their past ac-

tions, sometimes incorporating additional external information, such as environmental observations and feedback. In this work we focus on multi-agent that solves user queries while understanding user data on a device. Examples of such data includes past user actions, personal entities and installed toolsets (Wu et al., 2024).

Retrieval Augmented Generation and Long-context Language Models are two orthogonal approaches to ground a fine-tuned LM with external data sources, which in this work include the dynamic set of personal entities and tools (Borgeaud et al., 2022). A standard workflow of RAG includes possibly a query generation step (or a query decomposition step for compositional utterances) (Ma et al., 2023; Rackauckas, 2024) followed by sparse and/or dense retrieval. The retrieval model is commonly a separate set of parameters which can be trained either separately or jointly with the LLM. A major limitation of RAG is that a sub-optimal retrieval model will gate the performance of the LLM which has access to more contextual information. In contrast, long-context LLMs allow for the direct incorporation of more external data into the prompt (Beltagy et al., 2020; Zaheer et al., 2020; Kitaev et al., 2019; Ding et al., 2023). However, this comes at the cost of increased size of prompt and KV caches, making it impractical for small language models (SLMs) and on-device deployment.

Prompt Compression is an optimization to reduce the number of prompt tokens at least at the inference time. We adopt the technique to enable SLMs to retrieve directly from a dynamic toolbox. Related to this work are the work of Gist tokens (Mu et al., 2024), Parallel Context Encoder (Yen et al., 2024), and Squid which compresses a piece of long text into a single embedding (Chen et al., 2024c). These approaches differ in terms of how the compressed embedding is learned and incorporated with the base LLM, as either prompt tokens or late fusions in the attention layer.

3 Methodology

3.1 Agents Overview

CAMPHOR is a collaborative agent framework that performs grounded query parsing on a user device. It consists of the following agents, including an orchestrator:

- *High-order reasoning agent* plans the process of understanding a user query, solving the

query by determining the order in which other expert agents are invoked, effectively using the expert skills to complete sub-tasks.

and various experts:

- *Personal context agent* generates function calls to search relevant personal context that would be helpful in resolving entity ambiguities and under-specified queries. **The set of function calls that can be invoked by the agent is unique for each user device**, as the databases of personal entities are linked to the apps installed on a user device.
- *Device information agent* generates generic function calls to retrieve device information including current location, time and screen entities.
- *User perception agent* represents a single function call to fetch the recent user activities on device.
- *External knowledge agent* generates generic function calls to seek information from external sources including web search, Wikipedia and calculator.
- *Task completion agent* generates function calls to represent the user intent towards task completion. **The set of function calls that can be invoked by the agent is unique for each user device**, as the capabilities are determined by the apps installed on a user device.

Take the following query as an example

Can you show me the cheapest flight options to Barcelona next month and add it to my calendar? Also, let my travel buddy know about our trip plan.

The high-order agent proactively gathers personal information to understand the user intent. This includes *Device information agent* to obtain the current location and *Personal context agent* to look up the entity *travel buddy*. The *Task completion agent* is finally invoked to generate the task completion function calls. Figure 2 shows a high-level overview of all CAMPHOR agents.

3.2 Dynamic Prompt Construction

We model all agents in CAMPHOR with the same underlying SLM. A general formulation of all agents is that they take as input an agent-specific

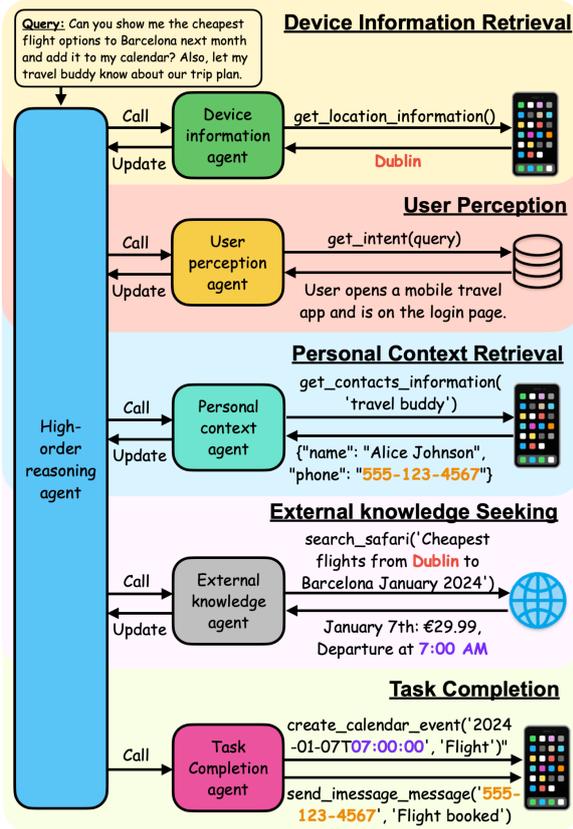


Figure 2: An overview of multiple agents in CAMPHOR. The figure includes all agents for completeness. In practice, a subset of the agents can be invoked in arbitrary order until task completion.

prompt and produces a function call which can be executed. The execution result of each expert agent is sent back to instruct the higher-order agent and the next expert agent. An agent prompt p_a is generated by a template formatting function $f(i_a, h_a, t_a)$, where i_a is an agent-specific task instruction and h_a stands for the message history (i.e., the past agent actions and observations) that the agent has access to.

t_a is an optional parameter representing function definitions that go into the prompt. Note that most agents actually use a static set of functions that are shared across user devices. The static set of function calls and parameters can be directly memorized by the model without definitions revealed in the prompt. However, two agents—the personal context agent and the task completion agent—interact with a dynamic set of device-specific functions. This is because the entity databases and capable tools are dependent on the apps installed on each user’s device. For these two agents, we need to present a dynamic set of

function definitions in the prompt.

3.3 Prompt Compression

As discussed earlier and demonstrated through experiments in Section 5.5, a RAG-based approach is sub-optimal because the retriever gates the performance of an SLM. Additionally, it is impractical to include the entire set of function definitions in the prompt, as this would quickly exceed the prompt token limit. To address this, we compress each function definition into a single token, which is then appended to the beginning of the prompt. This prompt compression approach is reminiscent of the cross-modality token used in multi-modal language modeling. By doing so, the agent can still access and reason over the full set of function definitions, while significantly reducing the number of input tokens—by a factor corresponding to the average length of function definitions in the prompt.

We opt for the SLM itself as a text encoder to obtain the single-token embedding for each function definition, by taking the output embedding of the last token therein, as illustrated in Figure 3. The choice is motivated by the fact that the language model is already pre-trained to encode text, offering meta-learning generalization. During fine-tuning, gradients will not be back-propagated through the function tokens. Comparing to gist tokens (Mu et al., 2024) which also leverage a pre-trained language model to encode texts as KV caches, our approach significantly reduces the cache size since only a single embedding is needed for each function definition, whose KV caches are computed on the fly of language model inference.

Positional Embeddings. We set custom position indices for the computation of the Rotary Positional Embeddings (Su et al., 2024). Every function token in the prompt shares the same position index 0 while the first token in the formal prompt starts with with position index 1. Function tokens are restricted from attending to each other, but each prompt token can attend to all function tokens, reasoning over the toolbox jointly.

4 CAMPHOR Dataset

A central focus of CAMPHOR is personalized planning and query understanding on device. However, existing function calling datasets (Patil et al., 2023; Qin et al.) only provide task completion annotations for user queries but not incorporating personal knowledge for understanding. On the other hand,

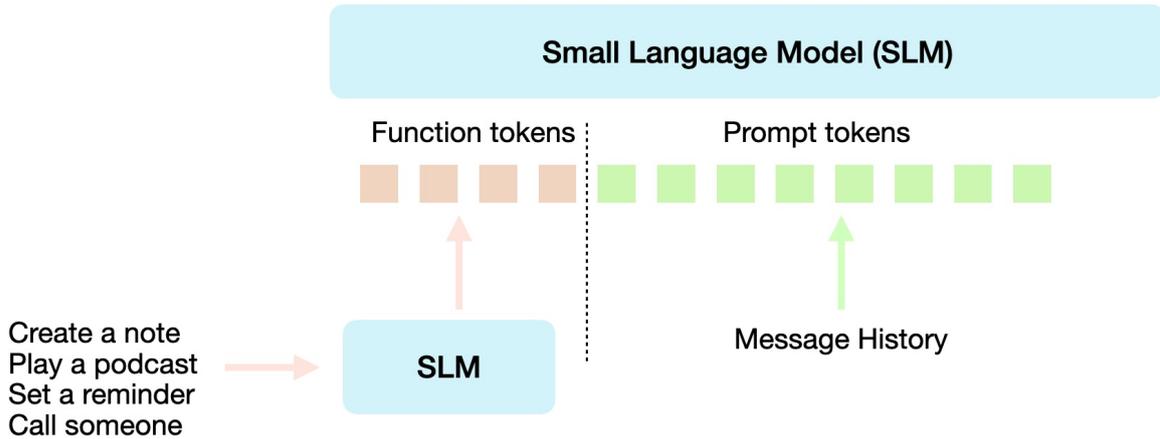


Figure 3: Prompt compression technique. We use the pre-trained SLM itself as a text encoder to generate a single-token embedding for each function description, by taking the output embedding of the final token therein. The compressed function tokens are appended to the beginning of the prompt.

there exist a few datasets on agent planning but they largely focus on mathematics (Cobbe et al., 2021; Mishra et al., 2022; Lu et al.) and common sense reasoning (Talmor et al., 2019; Geva et al., 2021) instead of query parsing.

To this end, we created the CAMPHOR dataset by annotating each query with a trajectory of function calls that demonstrate how a multi-agent system proactively fetches personal information to solve a user query by breaking down the understanding task into smaller actions. The dataset is developed by assigning a personal device state to each query, which includes a randomly sampled history of user activities, as well as the personal entities and tools available on the device. Each query in the dataset is generated by GPT-4o based on a device state and a set of global function definitions. The GPT-4o is also instructed to annotate the query in a multi-step fashion. The execution results are fetched from the device state for each function calling, which are then used to guide the next step of annotation. The final solution path is reviewed and verified with human oversight. Overall, the CAMPHOR dataset contains 3,410 queries, which are split into 2,728 for training and 682 for test. The dataset is flattened, resulting in 35,444 prompt-completion pairs for SLM fine-tuning, with an average of 10.39 pairs per query.

5 Experiments

We consider two SLM candidates for fine-tuning the CAMPHOR agents: Phi-3.5 and Gemma-2. The sequence of prompt and completion pairs associated with each query is obtained by unrolling the

ground truth trajectories in the dataset. Fine-tuning details are provided in Appendix A.

5.1 Evaluation Metrics

We consider three end-to-end evaluation metrics on task completion:

- *Tool F1* measures the accuracy of the function names used in task completion function calls. F1 is selected as the metric because it not only accounts for true-positive predictions within the ground truth set, but also penalizes false-positive predictions outside of it.

Example: `send_imessage_message`

- *Delexicalized Plan F1* measures the accuracy of both function names and parameters in task completion function calls. A true-positive prediction must not include any parameter hallucinations. The prediction is measured at the abstract syntax tree level, disregarding the order of parameters.

Example: `send_imessage_message(receiver, content)`

- *Plan F1* measures the accuracy of both function names, parameters and their values in task completion function calls. A subset of the parameters does support an open set of values, instead of closed-set enums. We adopt a lenient match rule to evaluate open-ended values: a match is incurred if the SentenceBERT (Reimers, 2019) embedding similarity between target and predicted values is higher than threshold 0.7.

```
Example:      send_imessage_message(
receiver = "555-123-4567", content =
"Flight Booked")
```

5.2 LLM Baseline Experiments

Before presenting results for the fine-tuned SLM agents, we first evaluated the performance of state-of-the-art LLMs on the CAMPHOR test set as baselines. We choose Claude-3.5 as the LLM for evaluation to avoid any potential label leakage as the CAMPHOR dataset is generated with GPT-4 in the loop.

A key difference between instruction-based inference and fine-tuning is that the former relies fully on the prompt instructions which must be clear and often framed with specific structure to guide the pre-trained model. In comparison (as we will show in Section 5.3), the prompts used in fine-tuning can be more concise and tailored according to prompt budgets, as the model is tuned to act for certain pattern of inputs. Given the requirement of instruction-based inference, we evaluated a wide range of prompting strategies and aim to pick the best for the comparison with SLM fine-tuning. The prompting strategies include:

- *Static* employs a consistent prompt template which contains all available function definitions for all CAMPHOR agents. The LLM agent is tasked to generate a sequence of function calls for each CAMPHOR query. The prediction history is also appended to the prompt of each turn.
- *ReAct* is similar to the *Static* baseline, but additionally has the option to perform an explicit reasoning step before generating a function call.
- *Reflexion* is similar to *ReAct*, but additionally incorporates a reflection step to examine the generated function calls and provide feedback. *Reflexion* inherently requires multiple trials, the number of which is set to 3.
- *AUTOACT* employs three distinct prompt templates that respectively handle function call generation, parameter filling and reflection of the results. Similar to other baselines, each CAMPHOR query is parsed as a sequence of function calls with parameter values. The prediction history is also appended to the prompt of each turn.

- *CAMPHOR Agents* employ distinct prompt templates for each CAMPHOR agent. The prompt of each agent contains agent-specific task descriptions, function calls of that agent and in-context examples. The prediction history is also appended to the prompt of each turn. This setting is closer to the dynamic prompt construction adopted in fine-tuning. Examples for the prompts can be found in the Appendix B.

Evaluation results of various LLM prompting strategies are presented in Table 1. The table demonstrates the importance of dynamic, agent-specific prompting strategies in solving CAMPHOR queries. One conclusion is that it is sub-optimal to overload all agent instructions and function definitions into a static prompt template. Given the results, we select *CAMPHOR-agent* as the LLM baseline to compare with fine-tuned SLM agents.

5.3 SLM Fine-tuning Experiments

Remember that we consider two base SLMs, Phi-3.5 and Gemma-2, for fine tuning. A key question we aim to answer is how to formulate the prompt such that the SLM maintains high accuracy while satisfying the prompt budget of on-device deployments.

We start with the dynamic prompt formatting function described in Section 3.2. Each agent prompt contains an agent-specific task description, the prediction history and optionally function descriptions. Compared to the agent-specific prompts in LLM experiments, there are two differences in the SLM fine-tuning. First we only append definitions for the dynamic set of functions for the personal context agent and the task completion agent, since the static functions (and their parameters) can be memorized via fine-tuning. Second, we removed in-context examples for each agent from the prompt, considering the prompt budget and also because the model can be trained to react to input patterns without in-context learning. In the fine-tuning process, we employ parameter sharing across agents so that all of them are trained jointly in a single, unified model.

Table 2 shows the results comparing the fine-tuned SLMs with the best LLM prompting strategy. The fine-tuned SLMs, including both Phi-3.5 and Gemma-2, outperform the LLM result in task completion metrics. Meanwhile, the Phi-3.5 model without fine-tuning does poorly in task comple-

Metric	Prompting Strategy				
	CAMPHOR	AUTOACT	Reflexion	ReAct	Static
Tool F1 / %	51.31	48.59	48.94	36.72	35.7
Delexicalized Plan F1 / %	30.07	25.77	24.49	19.16	18.78
Plan F1 / %	27.96	24.77	22.53	18.36	17.18

Table 1: Claude 3.5 Sonnet performance with different prompting strategies.

Metric	Model			
	Phi-3.5 (fine-tuning)	Gemma-2 (fine-tuning)	Claude-3.5 (LLM baseline)	Phi-3.5 (no fine-tuning)
Tool F1 / %	67.84	63.72	51.31	12.94
Delexicalized Plan F1 / %	44.85	41.57	30.07	10.39
Plan F1 / %	38.77	37.43	27.96	9.72

Table 2: Performance of the fine-tuned SLM compared to the untuned LLM baseline and Phi-3.5 (without fine-tuning). All experiments use the CAMPHOR LLM and CAMPHOR SLM prompts according to model type.

tion. The results highlight the effectiveness of fine-tuning an SLM for specialized agent tasks, showing it to be more powerful than simply prompting a pre-trained LLM with task instructions. Moreover, the performance of fine-tuning is not compromised by prompt simplification since the model is trained to learn fixed input-output mappings patterns.

To further optimize the prompt, we remove system instructions from each agent prompt and only reveal the prediction history, based upon which the SLM is fine-tuned to predict the next function call in the trajectory. Surprisingly we found that the prompt simplification leads to only marginal degradation of the task completion, with a plan F1 38.3% compared to 38.7% in the original setting. The result demonstrates that fully non-instruction tuning is also a promising direction to further improve on-device efficiency without sacrificing much accuracy.

5.4 Prompt Compression

Even though we only append definitions for the dynamic function set in the prompt, they still consume a significant amount of prompt space for large toolboxes. We further experiment with the prompt compression technique described in Section 3.3 where each function definition is represented as a single token in the prompt.

As shown in Table 3, applying the prompt compression technique only leads to marginal changes in the task completion F1, from 39.89 % to 38.45 %.

But it should be noted that the prompt compression technique reduces the number of static prompt tokens (without message history which dynamically grows) further by 96.00% for the personal context agent and 95.02% for the task completion agent.

Table 4 shows a summary of all prompt optimisation steps we have taken for SLM fine-tuning. Comparing to the prompt used for the CAMPHOR LLM baseline, the fine-tuning prompt is made much shorter and concise. We optimised the prompt by removing agent-specific instructions and employing compression technique to represent each function definition as a single token. The total static token reduction rate compared to the baseline is 98.3%, meaning that the fine-tuned SLM requires much shorter prompts than instruction-based inference in LLM.

5.5 Comparison with RAG

One could argue that an alternative approach to generalize to a dynamic toolbox is retrieval-augmented generation (RAG). However, we showcase here that RAG creates a performance bottleneck for the SLM when handling CAMPHOR queries due to sub-optimal retrieval recall. Figure 4 shows the retrieval recall at K for the CAMPHOR test queries using a Sentence-BERT (Reimers, 2019) as the retriever. The recall at K=5 is only 0.5 for the personal context agent and 0.8 for the task completion agent. Unsurprisingly the end-to-end plan F1 for a RAG approach with top 5 function definitions in

Metric	No Prompt Compression	Prompt Compression	Relative Δ
Tool F1 / %	66.95	65.91	-1.55%
Delexicalized Plan F1 / %	44.68	44.29	-0.87%
Plan F1 / %	39.89	38.45	-3.61%
# of PC Tool Tokens	575	23	-96.00%
# of TC Tool Tokens	261	13	-95.02%

Table 3: Applying prompt compression significantly reduces the prompt size with marginal changes in accuracy. The prompt size reduction is measured for personal context (PC) and task completion (TC) agents respectively.

	CAMPBOR LLM	CAMPBOR SLM	-instructions	+compression
Prompt	{task description} {agent instructions} {tool definition} {message history} {few-shot examples}	{agent instructions} {tool definition} {message history}	{tool definition} {message history}	{compressed tokens} {message history}
Static token reduction %	0%	48.6%	50.3%	98.3%

Table 4: A summary of prompt optimisation for CAMPBOR agents. Prompts in SLMs are simplified by virtue of fine-tuning. We further experimented with removing task instructions from the prompt, and compression techniques. Static token reduction rate measures the reduction of static prompt tokens (excluding message history) compared to the prompt used in instruction-based inference.

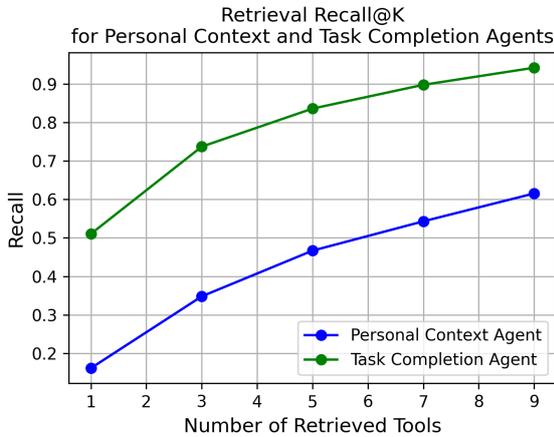


Figure 4: Retrieval recall at K computed with an external retrieval model for personal context agent and task completion agent.

the prompt is only 32.5% due to error propagation, compared to 38.7% when all function definitions are present in the prompt.

Why is RAG not working well? Queries in CAMPBOR are compositional with multiple task completion function calls. The average number of task completion function calls for each query is 3. However, given a tight prompt budget of $K=5$, it is rather difficult to make sure the retriever is able to

fetch all function calls into the $K=5$ bucket. As a direct consequence, the language model will not see the correct function definitions in the prompt, conditioned on which it is trained to generate the completion. The following shows an example where retrieval failed to find all ground truth:

Query

Can you show me the cheapest flight options to Barcelona next month and add it to my calendar? Also, let my travel buddy know about our trip plan.

Ground truth functions

create_calendar_event, send_message

Retrieved functions

send_mail, send_message,
download_appstore_app, play_podcasts,
create_reminders

While admitting that the performance of the retriever on such queries can be improved with customized fine-tuning and more advanced dense retrieval techniques (Santhanam et al., 2022), we contend that the pre-trained language model itself holds greater potential for selecting the appropriate function call in an end-to-end manner, when it has full visibility of the toolbox enabled by prompt

compression techniques.

6 Conclusion

This work introduces CAMPHOR, a collaborative, SLM-based agent framework designed for personalized query parsing on user devices. CAMPHOR proactively retrieves on-device information and decomposes the understanding tasks into multiple steps of function calls. Our results show that a fine-tuned SLM outperforms instruction-based LLMs in this task. By employing advanced prompt compression techniques, CAMPHOR strikes an optimal balance between accuracy and efficiency, while safeguarding user data directly on the device.

Limitations

The personalized user query parsing task studied in this work is restricted to single interactions. While many user queries can indeed be resolved in one interaction, this approach oversimplifies the problem space. In practice, many real-world tasks—especially those requiring user disambiguation or confirmation—still depend on multi-turn interactions between the user and the assistant. In such cases, system policies play a critical role in guiding the conversation and triggering the next agent. Future work should focus on extending CAMPHOR to handle multi-turn conversations, incorporating system policies and user follow-ups.

The simulated device environment in this work also primarily focuses on the "happy path" of personal information retrieval. It does not account for more complex runtime feedback and error-handling logic, such as conflicting information and disambiguation requests for multiple search results, which would need to be communicated back to the user before task continuation. In future, we aim to scale our data simulation approach to handle more complex runtime feedback and in multi-turn conversational settings, as discussed in the first paragraph.

Although CAMPHOR is designed for on-device deployment—a point we emphasize in the paper—this work does not include a dedicated evaluation of privacy guarantees, security safeguards, or latency/resource trade-offs. Real-world on-device systems must balance inference speed, energy consumption, and data isolation to meet user expectations and platform constraints. Future experiments will benchmark end-to-end latency, measure energy usage on representative hardware, and analyze how

local data processing impacts privacy and threat models.

References

- Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.
- Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.
- Wei Chen, Zhiyuan Li, Zhen Guo, and Yikang Shen. 2024a. Octo-planner: On-device language model for planner-action agents. *arXiv preprint arXiv:2406.18082*.
- Wei Chen, Zhiyuan Li, and Mingyuan Ma. 2024b. Octopus: On-device language model for function calling of software apis. *arXiv preprint arXiv:2404.01549*.
- Wei Chen, Zhiyuan Li, Shuo Xin, and Yihao Wang. 2024c. Dolphin: Long context as a new modality for energy-efficient on-device language models. *arXiv preprint arXiv:2408.15518*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#).
- Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501.

- Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.
- Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.
- 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*.
- Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2019. Reformer: The efficient transformer. In *International Conference on Learning Representations*.
- Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*.
- Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*.
- Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations*.
- Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315.
- Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, et al. 2022. Lila: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832.
- Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.
- 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*.
- Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Huajun Chen, et al. Autoact: Automatic agent learning from scratch for qa via self-planning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.
- Zackary Rackauckas. 2024. Rag-fusion: a new take on retrieval-augmented generation. *International Journal on Natural Language Computing (IJNLC) Vol*, 13.
- N Reimers. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.
- Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.
- Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024. [Avatar: Optimizing llm agents for tool-assisted knowledge retrieval](#).

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Howard Yen, Tianyu Gao, and Danqi Chen. 2024. [Long-context language modeling with parallel context encoding](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2588–2610, Bangkok, Thailand. Association for Computational Linguistics.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

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

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning*.

A Fine-tuning details

Table 5 provides hyper-parameters for fine-tuning the SLMs, including both Phi-3.5 and Gemma-2.

B Prompts

The CAMPHOR baseline prompt used to instruct the LLM are shown below. The prompt serves as the starting point to simplify and derive various fine-tuning prompts.

The prompt is constructed from the chat template with a system role and a user role, each of which contains a few variables. We present the templates for the system role and the user role respectively.

Hyper-parameters	Phi-3.5	Gemma-2
Optimization		
Batch Size	128	32
Training Steps	443	600
Warmup Ratio	0.03	0.03
LR	1×10^{-4}	1×10^{-4}
Seed	42	42
Hardware		
Data Type	bfloat16	bfloat16
Accelerator	A100 80G	A100 80G
LoRA Config		
LoRA Alpha	16	16
LoRA Dropout	0.05	0.05
LoRA Rank	16	16
Target Proj Modules	qkv, o, gate_up, down	q, k, v, o, gate, up, down
Quantization		
Quantization Type	nf4	nf4
Double Quantization	true	true
Computation Type	bfloat16	bfloat16

Table 5: Hyper-parameters for fine-tuning Phi-3.5 and Gemma-2.

The following are the templates for the system role and user role:

System-role Template

{task description} (Optional)
{agent architecture description} (Optional)
Here are available API calls:
{tool definition} (Optional)

User-role Template

{agent specific instruction} (Optional)
Here is the message history:
{message history}
{few shot examples} (Optional)

The following is the task description in the system role:

Task Description

You are a helpful digital assistant. An iPhone user has issued a query to you. Your ultimate goal is to provide an accurate and helpful response and complete any related tasks. This may involve utilizing additional context, such as personal contexts and relevant facts, to enhance the user experience.

The following are the agent descriptions in the system role:

Agent Descriptions

To successfully complete a complex task, collaboration among the following types of agents is required:

1. High Order Reasoning Agent. This agent is used to plan the specific execution process of the task, solving a given task by determining the order in which other expert agents are invoked. Also, this agent will be responsible for overseeing the communication between the expert agents, effectively using their skills to complete sub-tasks.
2. Information Agent. This agent is responsible for providing direct information including location information, time information or screen information. Location Information: Detailed current location information of user.
Time Information: Detailed current time information of user.
Screen Information: A detailed textual description of the user's screen content. When calling this agent, please select one type of information to retrieve.
3. Perception Agent. This agent is responsible for translating the onscreen context into a high-level understanding of the user's intent. Note that this intent is abstract; if more detailed textual information is needed, the Information Agent would be a better choice.
4. Personal Context Retrieval Agent. This agent is responsible for actively seeking relevant personal context that would be helpful in more accurate and personalized response.
5. Tool Calling Agent. This agent is responsible for calling useful tools. Tools can include external tools like web searches, Wikipedia, and calculators.
6. Answer Agent. This agent is responsible for generating tentative responses and task completion API calls based on the message history. These responses can then be reviewed by Reflection Agent and polished for the final answer.
7. Reflection Agent. This agent evaluates the proposed final response and execution history to

determine whether the suggested textual response and task completion API calls are appropriate for the given query and provides recommendations for improving the response.

8. Response Submit Agent. This agent is similar to Answer Agent, and is also responsible for generating tentative responses and task completion API calls based on the message history. The output from this agent will be directly submitted to the user.

These agents will communicate by sending messages and sharing a message history.

The following are the tool definitions for each agent in their system role:

Tool Definition for Device Information Agent

`get_screen_information()`: Get a detailed textual description of the user's screen content.

`get_location_information()`: Get detailed current location information of user.

`get_time_information()`: Get detailed current time information of user.

Tool Definition for User Perception Agent

`get_intent()`: Get a high-level understanding of the user's intent.

Tool Definition for Personal Context Agent

`get_settings_cellular()`: Retrieve user's cellular data usage summary.

`get_settings_notifications(keyword)`: Retrieve user's notifications containing a specific keyword.

`get_health_records()`: Retrieve user's health records.

`get_health_medications()`: Retrieve user's medication list.

`get_fitness_summary()`: Retrieve user's fitness summary and activity.

`get_safari_history(keyword)`: Retrieve browsing history of Safari containing a specific keyword.

`get_news_history(keyword)`: Retrieve browsing history of News containing a specific keyword.

`get_podcasts_history(keyword)`: Retrieve listening history of Podcasts containing a specific keyword.

`get_notes_content(keyword)`: Retrieve notes containing a specific keyword.

`get_reminders_content(keyword, time_range)`: Retrieve reminders containing a specific keyword or/and within a specific time range.

`get_calendar_event(theme, time_range)`: Retrieve calendar events related to a theme or/and within a specified time range.

`get_mail_event(theme, time_range)`: Retrieve mail invitation or confirmation for events related to a theme or/and within a specified time range.

`get_imessage_history(keyword)`: Retrieve chatting history of iMessage containing a specific keyword.

`get_music_playlist(keyword)`: Retrieve songs in user's music playlist containing a specific keyword.

`get_voice_recording(keyword)`: Retrieve recordings from the user's voice memos with titles containing a specific keyword.

`get_books_library()`: Retrieve user's reading books.

`get_contacts_information(keyword)`: Retrieve contact information, including `person_id`, `name`, `phone_number`, `relationship`.

`get_appstore_history()`: Retrieve the purchase and download history of apps.

`get_maps_places(keyword)`: Retrieve user's saved places containing a specific keyword.

`get_amazon_information()`: Retrieve user's Amazon account information.

`get_amazon_orders(keyword)`: Retrieve user's Amazon orders containing a specific keyword.

`get_instagram_information()`: Retrieve user's Instagram account information.

get_instagram_post(keyword): Retrieve user's Instagram post containing a specific keyword.

Tool Definition for External Knowledge Agent

search_safari(query): Perform a search in Safari app using the specified query, which can include searches for information, weather forecasts, available items on Amazon, and other types of information.

Tool Definitions for Task Completion Agent

play_podcasts(title): Play a podcast with the specified title.
create_notes(content): Create a note with the specified content.
create_reminders(time, content): Set a reminder with the specified content at the specified time.
create_calendar_event(time, event_title): Create a calendar event with the specified event_title at the specified time.
cancel_calendar_event(event_title): Cancel the calendar event with the specified event_title.
send_mail(receiver, content): Send an email to the receiver with the specified content.
send_imessage_message(receiver, content): Send a message to the receiver with the specified content via iMessage.
play_music(title): Play music with the specified title.
call_contacts(person): Call the specified person.
download_appstore_app(app_name): Download the specified app.
show_maps_place(name): Show the location of the specified place in the Maps app.
show_amazon_item(name): Show the page of the specified item on Amazon.
create_instagram_post(content): Create a new post with the specified content on Instagram.

The following are the agent-specific instructions in the user role:

Agent Specific Instruction for Device Information Agent

Now your task is to generate accurate and helpful API calls to retrieve device information based on the message history.

Agent Specific Instruction for User Perception Agent

N/A (Just one tool in the toolbox and there are no parameters; no need to call the model to determine the tool.)

Agent Specific Instruction for Personal Context Agent

Now your task is to generate accurate and helpful API calls to retrieve personal context based on the message history.

Agent Specific Instruction for External Knowledge Agent

Now your task is to generate accurate and helpful API calls to retrieve relevant facts or public information based on the message history.

Agent Specific Instruction for Task Completion Agent

Now your task is to generate accurate and personalized textual response and task completion API calls for user based on the message history.

The following is an example for the message history. This history gets updated dynamically after an agent is invoked and the execution result is sent back.

Example of Message History

[User]: Can you show me the cheapest flight options to Barcelona next month and add it to my calendar? Also, let my travel buddy know about our trip plan.

[High Order Reasoning Agent]: [Device Information Agent]

[Device Information Agent]: ['get_location_information()']

[Execution Result]: [{"latitude": 53.3478, "longitude": -6.2597, "city": "Dublin", "country": "Ireland", "postal_code": "D01 V902", "formatted_address": "Ryanair Head Office, Airside Business Park, Swords, Co. Dublin, Ireland"}]

[High Order Reasoning Agent]: [Personal Context Agent]

[Personal Context Agent]: ["get_contacts_information(keyword='travel buddy')"]

[Execution Result]: [{"person_id": "003", "name": "Alice Johnson", "phone_number": "555-123-4567", "relationship": "Travel Buddy", "is_self": "False"}]

[High Order Reasoning Agent]: [External Knowledge Agent]

[External Knowledge Agent]: [search_safari(query='Cheapest flights from Dublin to Barcelona January 2024')]

[Execution Result]: [Cheapest flights from Dublin to Barcelona in January 2024:
- Tuesday, January 7th: €29.99, Departure at 7:00 AM, Arrival at 10:30 AM. -
Thursday, January 16th: €32.50, Departure at 6:45 AM, Arrival at 10:15 AM. -
Friday, January 10th: €31.00, Departure at 8:00 AM, Arrival at 11:30 AM.]

[High Order Reasoning Agent]: [Task Completion Agent]

[Task Completion Agent]:

Textual Response:

Sure! Here are some of the cheapest flight options to Barcelona next month from various sources:
From Ryanair:

1. Tuesday, January 7th at 7:00 AM, arriving at 10:30 AM - €29.99.
2. Thursday, January 16th at 6:45 AM, arriving at 10:15 AM - €32.50.
3. Friday, January 10th at 8:00 AM, arriving at 11:30 AM - €31.00.

I will add the cheapest flight, which departs on January 7th at 7:00 AM and costs €29.99, to your calendar and notify your travel buddy.

Task Completion API Calls:

```
[ "create_calendar_event(time='2024-01-07T07:00:00', event_title='Flight to Barcelona - Departure from Dublin at 7:00 AM')", "send_imessage_message(receiver='555-123-4567', content='We have a flight to Barcelona on January 7th at 7:00 AM. Please be ready!')"]
```

The following is an example of few-shot examples in the user prompt:

Few Shot Examples

Here is an example output:

```
[get_maps_places(keyword="Apple"); get_imessage_history(keyword="Pear")]
```

Strictly use only the available API calls and separate each API call by semicolons in a list.

The following is an example of the full prompt:

An example of the Full Prompt

You are a helpful digital assistant. An iPhone user has issued a query to you. Your ultimate goal is to provide an accurate and helpful response and complete any related tasks. This may involve utilizing additional context, such as personal contexts and relevant facts, to enhance the user experience.

To successfully complete a complex task, collaboration among the following types of agents is required:

1. High Order Reasoning Agent. This agent is used to plan the specific execution process of the task, solving a given task by determining the order in which other expert agents are invoked. Also, this agent will be responsible for overseeing the communication between the expert agents, effectively using their skills to complete sub-tasks.
2. Information Agent. This agent is responsible for providing direct information including location information, time information or screen information. Location Information: Detailed current location information of user.
Time Information: Detailed current time information of user.
Screen Information: A detailed textual description of the user's screen content. When calling this agent, please select one type of information to retrieve.
3. Perception Agent. This agent is responsible for translating the onscreen context into a high-level understanding of the user's intent. Note that this intent is abstract; if more detailed textual information is needed, the Information Agent would be a better choice.
4. Personal Context Retrieval Agent. This agent is responsible for actively seeking relevant personal context that would be helpful in more accurate and personalized response.
5. Tool Calling Agent. This agent is responsible for calling useful tools. Tools can include external tools like web searches, Wikipedia, and calculators.
6. Answer Agent. This agent is responsible for generating tentative responses and task completion API calls based on the message history. These responses can then be reviewed by Reflection Agent and polished for the final answer.
7. Reflection Agent. This agent evaluates the proposed final response and execution history to determine whether the suggested textual response and task completion API calls are appropriate for the given query and provides recommendations for improving the response.
8. Response Submit Agent. This agent is similar to Answer Agent, and is also responsible for generating tentative responses and task completion API calls based on the message history. The

output from this agent will be directly submitted to the user.
These agents will communicate by sending messages and sharing a message history.

You are the Personal Context Agent that is responsible for actively seeking relevant personal contexts that would be helpful in more accurate and personalized response. The High Order Reasoning Agent has assigned a task to you. Could you please generate a sequence of personal context retrieval API calls to retrieve relevant personal context in various smartphone apps based on the available calls and the message history?

Here are available API calls to retrieve relevant personal information for each app:

`get_settings_cellular()`: Retrieve user's cellular data usage summary.
`get_settings_notifications(keyword)`: Retrieve user's notifications containing a specific keyword.
`get_health_records()`: Retrieve user's health records.
`get_health_medications()`: Retrieve user's medication list.
`get_fitness_summary()`: Retrieve user's fitness summary and activity.
`get_safari_history(keyword)`: Retrieve browsing history of Safari containing a specific keyword.
`get_news_history(keyword)`: Retrieve browsing history of News containing a specific keyword.
`get_podcasts_history(keyword)`: Retrieve listening history of Podcasts containing a specific keyword.
`get_notes_content(keyword)`: Retrieve notes containing a specific keyword.
`get_reminders_content(keyword, time_range)`: Retrieve reminders containing a specific keyword or/and within a specific time range.
`get_calendar_event(theme, time_range)`: Retrieve calendar events related to a theme or/and within a specified time range.
`get_mail_event(theme, time_range)`: Retrieve mail invitation or confirmation for events related to a theme or/and within a specified time range.
`get_imessage_history(keyword)`: Retrieve chatting history of iMessage containing a specific keyword.
`get_music_playlist(keyword)`: Retrieve songs in user's music playlist containing a specific keyword.
`get_voice_recording(keyword)`: Retrieve recordings from the user's voice memos with titles containing a specific keyword.
`get_books_library()`: Retrieve user's reading books.
`get_contacts_information(keyword)`: Retrieve contact information, including `person_id`, `name`, `phone_number`, `relationship`.
`get_appstore_history()`: Retrieve the purchase and download history of apps.
`get_maps_places(keyword)`: Retrieve user's saved places containing a specific keyword.
`get_amazon_information()`: Retrieve user's Amazon account information.
`get_amazon_orders(keyword)`: Retrieve user's Amazon orders containing a specific keyword.
`get_instagram_information()`: Retrieve user's Instagram account information.
`get_instagram_post(keyword)`: Retrieve user's Instagram post containing a specific keyword.

Now your task is to generate accurate and helpful API calls to retrieve personal context based on the message history.

Here is the message history:

[User]: Can you show me the cheapest flight options to Barcelona next month and add it to my calendar? Also, let my travel buddy know about our trip plan.

[High Order Reasoning Agent]: [Device Information Agent]

[Device Information Agent]: ['get_location_information()']

[Execution Result]: [{"latitude": 53.3478, "longitude": -6.2597, "city": "Dublin", "country": "Ireland", "postal_code": "D01 V902", "formatted_address": "Ryanair Head Office, Airside Business Park, Swords, Co. Dublin, Ireland"}]

[High Order Reasoning Agent]: [Personal Context Agent]

Here is an example output:

[get_maps_places(keyword="Apple"); get_imessage_history(keyword="Pear")]

Strictly use only the available API calls and separate each API call by semicolons in a list.