Cognitive Kernel: An Open-source Agent System towards Generalist Autopilots

Hongming Zhang[‡], Xiaoman Pan, Hongwei Wang, Kaixin Ma, Wenhao Yu, Dong Yu
Cognitive Kernel Team[*], Tencent AI Lab, Seattle
{hongmzhang, xiaomanpan, hongweiw, kaixinma, wenhaowyu, dyu}@global.tencent.com

Abstract

We introduce Cognitive Kernel, an open-source agent system towards the goal of generalist autopilots. Unlike copilot systems, which primarily rely on users to provide essential state information, autopilot systems complete tasks from start to finish independently. This requires the system to acquire the missing state information actively. Cognitive Kernel adopts a dynamic programming design where the central policy model (a fine-tuned LLM) could initiate an environment state perception task, essentially another agent task, as needed. The results demonstrate that Cognitive Kernel achieves better or comparable performance to other closed-source systems on core autopilot capabilities. Cognitive Kernel is fully dockerized, ensuring everyone can deploy it privately and securely. We open-source the system to encourage further research on LLM-driven autopilot systems[1 2].

1 Introduction

Large language models (LLMs) have revolutionized the landscape of AI applications (OpenAI, 2023; Anil et al., 2023; Zhao et al., 2023; Chang et al., 2024). Systems like ChatGPT[1] have significantly enhanced productivity in everyday tasks. However, these systems primarily function as "Copilots," where users are required to manage the majority of the work, such as planning the overall workflow, posing the right questions, or refining the model's output as needed (Xi et al., 2023; Wang et al., 2024b). To fully harness the capabilities of LLMs and reduce the burden of tedious, repetitive tasks, we shall shift from building "Copilot" systems to "Autopilot" systems that can independently complete tasks. For instance, while a Copilot system might assist in drafting a template for an invitation email, an Autopilot system should be capable of composing the entire email and sending it. At the same time, an autopilot system should be a general-purpose assistant capable of handling various user needs instead of being customized for a particular environment (e.g. an email agent).

To achieve this goal, one should give the agent system the capability of proactively acquiring the essential environment information when needed rather than passively waiting for humans to provide through prompts (Figure 1 (a)). The system should also adaptively interact with the suitable environment rather then being constrained to a certain task-specific environment (Figure 1 (b)). Following this design principle, we propose Cognitive Kernel, which has the freedom to interact with the real world to acquire missing information. As illustrated in Figure 1(c), Cognitive Kernel containts three conceptual components: the reasoning kernel, the perception kernel, and the memory kernel, corresponding to decision-making, state perception, and state storage, respectively. Analogous to a Turing Machine (Turing et al., 1936), the reasoning kernel functions as the transition mapping mechanism, the perception kernel serves as the tool for reading current state information, and the memory kernel acts as the tape, recording past state information. At each step, the reasoning kernel will generate the next action, which might involve activating the perception kernel to gather missing state information or engaging the memory kernel to store or retrieve critical historical information. Since such perception cannot be pre-defined, we treat the perception task as another fine-grained autopilot task in a dynamic programming paradigm.

This autopilot agent design poses significant challenges to the central policy model of the system, as it would need to handle general chats with users, environment perception planning and decision-making within diverse environments. Hence directly applying a fixed model leads to un-

[*]All authors contribute equally and are listed randomly.
[‡]Hongming Zhang is the project lead.
[1]Code: https://github.com/Tencent/CogKernel
[2]Demo: https://youtu.be/vZ4GEwIas-o

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations), pages 328–349

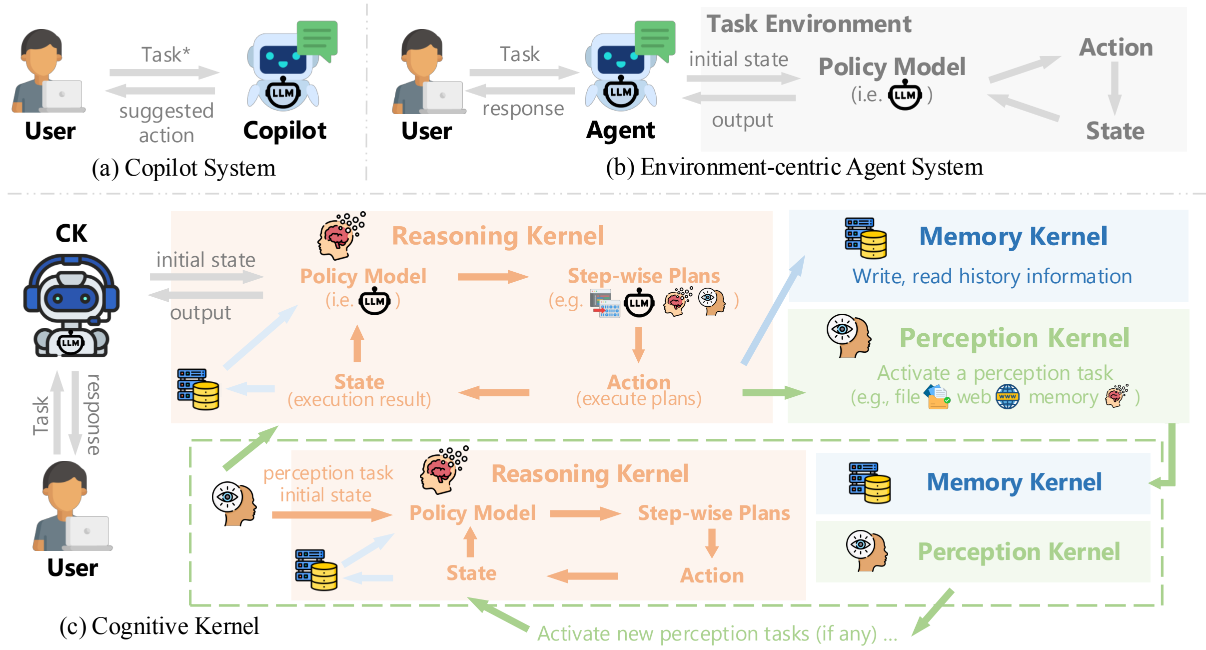

Figure 1: Comparison of conceptual frameworks: (a) Copilot system, (b) Environment-centric Agent system, and (c) Cognitive Kernel system, highlighting key structural differences. After receiving a task, Cognitive Kernel evaluates whether it has all essential state information to make a sound action. If not, it actively perceives missing state information from the environment, which can be a deeper-level self-contained autopilot task.

satisfactory performance. To address this issue, Cognitive Kernel employs a two-stage training process that enables the policy model to develop fundamental capabilities and then continuously learn from interaction feedback for further improvement after initial deployment.

We evaluate the system's performance across three essential capabilities for an "autopilot" system: real-time information management (e.g., gathering data and completing tasks on the open web), private information management (e.g., processing and understanding local files), and long-term memory management (e.g., personalizing the system based on user interactions). By comparing our system with leading closed-source systems (e.g., ChatGPT website, Kimi website) and open-source agent systems utilizing different foundation models, we have made the following observations: (1) No single system consistently delivers the best performance across all tasks; each system has its strengths as well as inherent behavioral biases; (2) Optimal performance is achieved through the deep integration of model and system design. Cognitive Kernel demonstrates significantly better results when paired with our specifically adapted model than an overall stronger model (GPT-4o).

To summarize, Cognitive Kernel makes the following contributions: (1) Proactive State Acquisition: Our system introduces an innovative approach to proactively acquiring state information, enabling the possibility of evolving from a "copilot" to an "autopilot" system. (2) A Dynamic Programming (DP) Framework: We propose an elegant and efficient DP-based agent design tailored for proactive state perception. (3) An easy-to-use dockerized system that everyone can deploy locally.

2 System Design

In this section, we introduce the motivations and design principles of the three kernels in our Cognitive Kernel system.[3] In Figure 2, we demonstrate an example usage of our Cognitive Kernel system.

2.1 Reasoning Kernel

The reasoning kernel generates and executes plans for the next action use the programming language (i.e., Python) as the medium (Li et al., 2024; Zhang et al., 2024). Basic Python operations, such as arithmetic (+, -, *, /) and logical comparisons (==, >, <) are treated as atomic actions, while Python functions serve as compound actions. This approach offers better flexibility in handling uncer-

[3]A conceptual framework of "Autopilot" systems can be found at Appendix B.

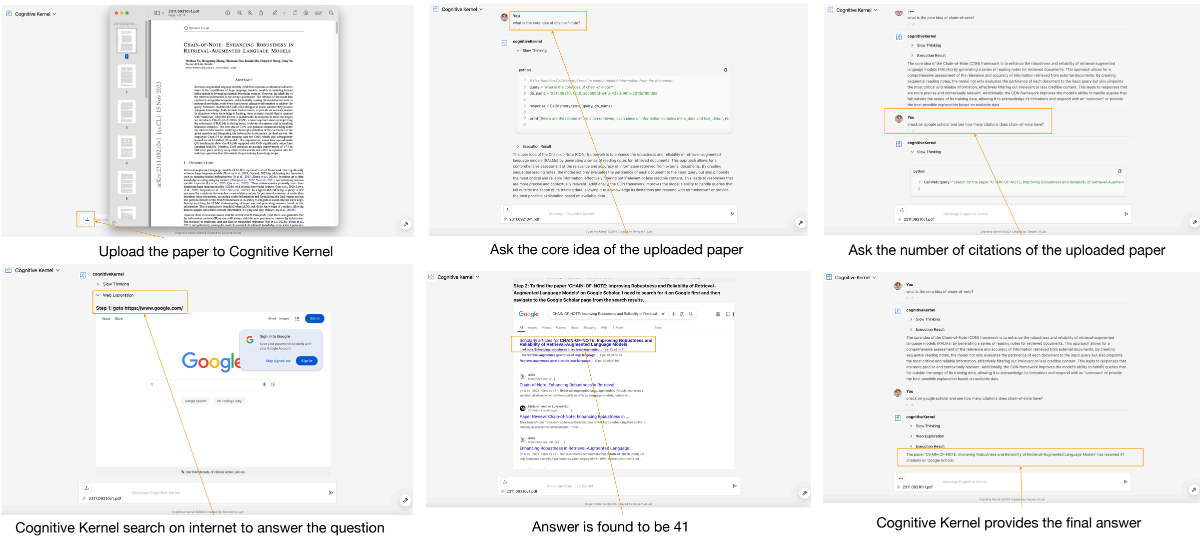

Figure 2: An example trajectory of Cognitive Kernel on completing user queries leveraging both private information management and real-time information management abilities.

tainty through constructs like "if/else" for designing alternative strategies or "for loop" for iterative attempts, which are often infeasible in natural language. Also, programming language provides a much higher level of parallelism than natural language, allowing multiple steps to be executed simultaneously. To enhance PL-centric planning, we further make two improvements to over the past code-centric agent systems. Firstly, instead of single-time execution, Cognitive Kernel implements a state caching mechanism to cache previous execution states and functions for future usage. Secondly, we implement an async parallel execution mechanism in Cognitive Kernel, enabling independent steps to run simultaneously and avoiding the slow actions blocking the whole execution process.

2.2 Perception Kernel

The perception kernel accesses the environment to perceive missing localized states. This subsection explains how Cognitive Kernel acquires temporally and spatially localized state information.

Temporally Localized State Perception. The world is constantly changing and the "autopilot" needs to access the up-to-date information. Like humans using the internet, Cognitive Kernel is equipped with open web access. When the reasoning kernel identifies the need for temporal state information, it sends a task-specific query to the perception kernel, which activates a web server to execute the task, as shown in Figure 1. At each step, the system observes the current web session as state information and relays it to the reasoning kernel for next-step actions. Specifically, we follow Zhou et al. (2024) and use the web page's accessibility tree as the observation to the agent due to its structural format and conciseness. We further optimized the raw accessibility tree to reduce redundancy and prune out irrelevant information. We define the following atomic actions such that Cognitive Kernel could operate the browser to perceive real-time information as humans do: (1) **Click**: click an element on the webpage; (2) **Type**: clear the text content in an element and fill it with new content; (3) **Scroll**: scroll up or down of the current viewport; (4) **Goback**: go back to the previously browsed page; (5) **Restart**: return to the homepage directly and restart the browsing process; (6) **Stop**: summarize the relevant information and sent back to the upper-level reasoning kernel. The execution loop runs until the task is completed or the maximum number of steps is reached.

Spatially Localized State Perception. Another perspective of the localized state information is the spatial one. Real-world tasks often involve private information only accessible to the local user. This section covers the two most popular localized state resources: *local files* and *long-tem history*.

Files such as documents and spreadsheets are essential for daily information transfer, making it crucial for an "autopilot" system to perceive and process local files effectively. Similar to humans, Cognitive Kernel uses basic operations like opening files (e.g., file.open()) and performing tasks such as keyword searches, counting specific terms, or ex-

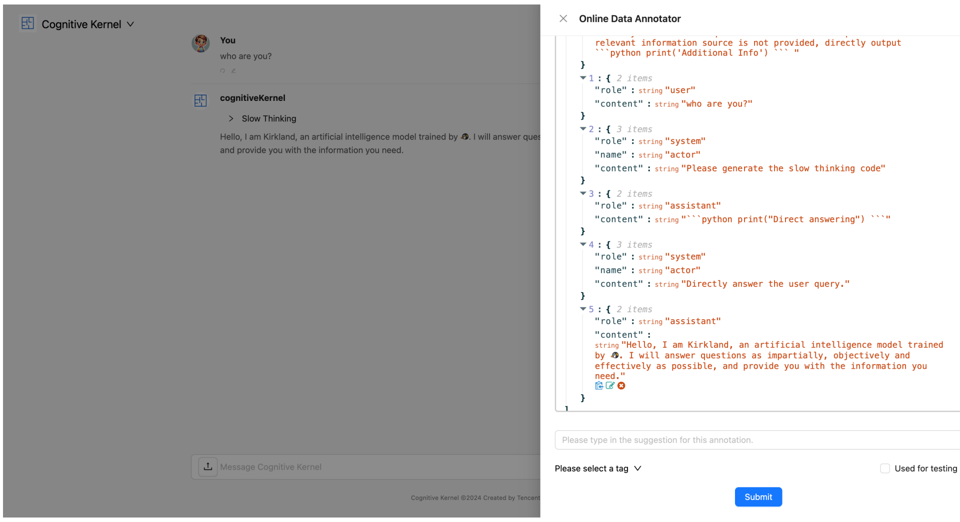

Figure 3: Cognitive Kernel 's frontend user interface when activating the feedback mode

tracting portions of input data (e.g., isolating rows with specific conditions in a spreadsheet) by generating code in the reasoning kernel. These capabilities enable Cognitive Kernel to handle straightforward tasks with precision and flexibility. In addition to these basic operations, Cognitive Kernel includes two advanced actions: (1) *Search**: Perform semantic-based retrieval to find relevant information within the file[4]. (2) *Read*: Employ a policy model to understand and summarize file content, extract key information, generate insights, or answer questions based on the document. After each action, the reasoning kernel determines if additional information is needed and activates appropriate operations for complex tasks.

Besides files, another important spatially localized state information is the long-term history between users and the "autopilot" systems. For example, a user might expect the system to remember their home location without needing to specify it in every conversation. However, appending all dialog histories to input is infeasible due to the limited context window size of langauge models. To solve this problem, we treat history as a special kind of spatially localized information that is available to the current user and "autopilot" system and formulate the storage and usage of such information as a perception task. Specifically, if Cognitive Kernel decides to store some information in the history or perceive history, it will activate the memory kernel to write into or load from the memory just like how modern computers operate the disk. More details of our memory design are in Section 2.3.

[4]This operation requires support of memory kernel. For implementation efficiency, when Cognitive Kernel receives the files, it will first add all contents into the memory kernel for later retrieval usage. More details are in Section 2.3.

2.3 Memory Kernel

The memory kernel is designed to provide a caching mechanism for the Cognitive Kernel, enabling it to save and retrieve past states. Its core component is a dense retrieval method that segments messages into fixed-length chunks and creates chunk indexes using representation models (Karpukhin et al., 2020; Ni et al., 2022; Izacard et al., 2022). To enable well-structured storage and fine-grained semantic retrieval, we employ a multi-granularity information management system as the memory kernel of Cognitive Kernel, including two major components: information processing/storage and information retrieval.

For Information processing and storage, we parse the user's messages into different granularities and create semantic index (i.e., embeddings) accordingly. These indexes encompass not only the original documents but also extracted propositions (Chen et al., 2024), key perspectives, and mentioned concepts. More details about obtaining fine-grained semantic index are in Appendix C.

For information processing and storage, user messages are parsed into various granularities, generating semantic indexes (i.e., embeddings). These indexes encompass original documents, extracted propositions (Chen et al., 2024), key perspectives, and mentioned concepts. For information retrieval, Cognitive Kernel employs a multi-granularity similarity matching system to identify the most relevant documents, propositions, and concepts. Retrieved documents are reranked based on their similarity scores (see Appendix C for more details).

3 System Implementation

3.1 Dockerized Design

We adopted a dockerized design to ensure efficient and secure deployment. As shown in Figure 1, the three conceptual kernels are deeply integrated. To optimize scheduling and execution, the system is organized into separate Docker containers, each tailored to its specific task. As shown in Figure 4, we implement Cognitive Kernel with five dockers:

Frontend Docker: We implement the frontend docker with React[2] and Nginx[3]. Figure 2, illustration of the frontend UI of Cognitive Kernelin various stages of execution. To collect feedback from users and thus continuously improve the system performance, we implement an online feedback module, where the user can see all details of the system execution and provide comments or suggestions accordingly. Users can activate the annotation mode, revealing an *Annotate* button next to each dialog turn. Upon clicking the button, the annotation interface will show up, as illustrated in Figure 3. This interface shows the full prompt of the current dialog session in the message format, including the system prompt turns. Users can edit the assistant's response directly or provide edit suggestions, with all changes saved to a persistent database.

Backend Docker: The backend docker is the center of Cognitive Kernel. At each step, the backend docker converts the input into a prompt, sends it to the inference docker to generate plans, and then executes them. To avoid repetitive plan generation and execution, the backend docker includes an execution state caching mechanism. Similar to a Jupyter Notebook[4], for any request, the backend docker will cache all intermediate execution results, including the variables and function definitions in the memory. For later steps, the backend docker will include that information as part of the state description and thus the model can avoid generating repetitive code, which can significantly increase the planning generation efficiency. Another optimization we did was the multi-processing synchronization. To increase the system efficiency, the backend docker adopts a multi-processing strategy to handle concurrent requests instead of processing the queue linearly. One thing worth mentioning is that since these concurrent requests might use shared cached variables, we will create a copy of the reused variables and provide a divergence of the caching branches to avoid potential conflicts.

Web Docker: We implemented the web server with Playwright[5]. Regarding the webpage observations, we found raw accessibility tree is sub-optimal because the tree could be lengthy and contain huge redundancy. To solve this problem, the web docker first parallelly localizes the visible elements from the current viewport and only constructs the tree for these elements. Then, we perform node deduplication heuristically so that only the nodes containing unique semantic information are kept. For actions that require arguments (target element to interact), e.g., click and type, previous works typically rely on the element's coordinates within the viewport to execute the action (Zhou et al., 2024). However, using the coordinates may lead to unexpected outcomes. For example, when dropdown menus or grid cells are expanded on certain websites, the coordinates of certain elements will overlap. To avoid such scenarios, Cognitive Kernel uses an element's role and name to pinpoint the target, ensuring precise execution regardless of the webpage layout.

Inference Docker: The inference docker receives prompts from the backend docker and calls the central language model to generate next-step plans. In Cognitive Kernel, we support both TGI[6] and vLLM (Kwon et al., 2023) as the inference server.

Database Docker: We use multiple database systems to fit the different needs of Cognitive Kernel. For permanent information that we want to store for a long time such as user feedback, we use the postgresql[7] due to its high reliability. For temporal content such as an uploaded file or cached execution results that are useful for the live sessions, we use sqlite[8] due to its lightness.

3.2 Policy Model Training

Since directly applying a closed-source model leads to unsatisfying performance, we trained our own model upon open-source language models (i.e., Llama3 (Dubey et al., 2024)). The training contains two stages. In stage one, we employ standard supervised fine-tuning. Specifically, we use a mixture of data including open-sourced instruction following data (Zhou et al., 2023; Luo et al., 2024), function calling data[9], agent trajectories data for various tasks (Zeng et al., 2024; Wang et al., 2024c; Yin et al., 2024; Zhou et al., 2024), and a small set of manually annotated data that fits our system design to train our model. This

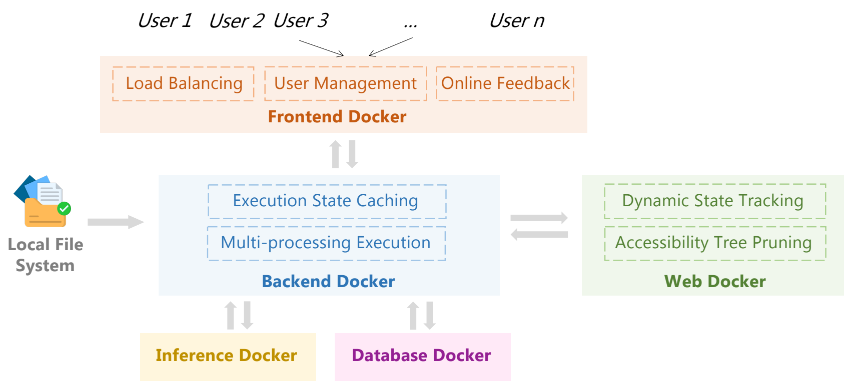

Figure 4: Engineering framework of Cognitive Kernel.

stage equips the model with the general problem-solving capability and the basic capability of invoking atomic actions. However, the output distribution remains relatively flat, leading to unstable performance. We conduct a second-stage training to overcome this challenge and enhance the model's generalization ability. Specifically, we deploy the first-stage model online and then collect the system's output trajectories given various inputs. Again, we used a mixture of data where the inputs are either mined from open-source datasets (Wang et al., 2024a; He et al., 2024; Dasigi et al., 2021; Trivedi et al., 2022) or submitted by internal users. Here, to ensure the quality of the collected data, we also collect judgments and feedback for the system trajectories. The judgments and feedback can come from a user (where the user can directly submit via Cognitive Kernel 's user interface) or the system itself (when the code produces the error and error message). More training details can be found in Appendix D.1.

4 Evaluation

4.1 Experimental Settings

We focus on evaluating Cognitive Kernel's ability on (1) gather real-time information and complete web-based tasks, (2) process user-uploaded files and answer questions, and (3) manage the interaction history with the user for better personalization.

Baseline Systems We mainly compare against the following general-purpose end-to-end AI systems in our experiments: **ChatGPT**[10] (4o), **Gemini**[11] (Pro-1.5), **Claude**[12] (opus), **Kimi**[13], and **Coze**[14], and we directly use their web interface for evaluation. Finally, we include a baseline using GPT-4o as Cognitive Kernel's backbone to assess the central policy model's impact.

Benchmarks For real-time information management, we evaluate systems on the WebCanvas benchmark (Pan et al., 2024), which includes 104 human-annotated tasks requiring interaction with live websites to complete specific instructions. For private information management, we assess systems using DocBench (Zou et al., 2024), which provides end-to-end evaluations with 229 real-world documents and 1,102 questions across five domains and four question types. For long-term memory management evaluation, we use LongMemEval (Wu et al., 2024a) containing of 500 human-assistant conversations. Each conversation consists of historical dialog sessions as well as a final test session. We provide additional benchmark details and evaluation in Appendix D.2 and D.3.

4.2 Overall Results

We present the overall results from our experiments in Table 1. We see that Cognitive Kernel can achieve the best results on real-time information management and long-term memory and comparable performance with state-of-the-art systems in the management of private information.

For real-time information management evaluation, the primary limitation of the baseline systems is their inability to directly interact with target websites, preventing them from completing tasks such as *adding items to cart* or *rating a movie on IMDB* (except Coze, which possesses web-browsing plug-ins). Furthermore, our observations reveal a consistent pattern of inherent behavioral biases across the systems. For example, Gemini often sticks to Google-related products/websites while ignoring the instruction, e.g., searching for music playlists on YouTube despite the instruction asking for Soundcloud. Also, Kimi-Chat nearly answers all of the questions in Chinese and it can only

Systems	Real-time Information (WebCanvas)	Private Information (DocBench)	Long-term Memory (LongMemEval)
GPT-4o (0806)	33.7	63.1	<u>59.3</u>
Gemini-Pro 1.5	31.7	55.4	-
Claude3-opus	-	67.6	-
Coze (GPT-4o)	<u>42.3</u>	28.6	58.1
Kimi-Chat	25.0	**70.9**	-
Cognitive Kernel (GPT-4o)	39.4	37.2	59.0
Cognitive Kernel	**49.0**	<u>68.2</u>	**85.9**

Table 1: The overall successful rates of different end-to-end systems: best in bold, second-best underlined.

access websites available in China, which leads to its low success rate. While an ideal system should not be impacted by such behavioral bias, we understand that the design decisions are also bounded by company and governmental policies.

In the private information management evaluation, all baseline systems can handle user-uploaded files, taking them as input to generate responses to user queries. Among the systems based on their own trained LLMs, such as GPT-4o, Gemini-Pro 1.5, and Claude3-opus, Coze stands out as a fully prompt-based system using GPT-4o as its backbone LLM. However, Coze and our Cognitive Kernel with GPT-4o performed poorly in handling user documents with a prompt-driven approach, as the system's instructions did not align well with the LLMs' training. Kimi-Chat outperformed all systems, particularly in managing long-context questions. GPT-4o ranks behind Kimi-Chat and Claude3-opus, often fabricating information not present in the document, especially when users ask unanswerable questions. Further analysis is provided in Appendix E.2.

For long-term memory management evaluation, we selected GPT and Coze as baseline methods, as they are the only two publicly available systems that support long-term memory. As shown in Table 1, GPT-4o slightly outperforms Coze, indicating that its long-term memory module design is superior. However, as discussed later in Appendix E.3, GPT-4o is susceptible to memory overwriting, which can result in catastrophic forgetting. In contrast, Cognitive Kernel achieves the best performance among all systems, with an overall accuracy of 85.9%. Additionally, we evaluated the scenario where the base LLM in the Cognitive Kernel is replaced with GPT-4o. The results showed a significant decrease in performance (from 85.9% to 59.0%), as GPT-4o is not specifically aligned with our system design. We provide more in-depth analysis of each core ability in Appendix E.

5 Applications

In this section, we showcase another application scenario of Cognitive Kernel to illustrate how users might interact with the system. In this example from Figure 10, we see that the user first asked the system to search for recent papers about web agents and download the first one it found. Cognitive Kernel again opened a web browser and searched web agent papers. Then it opened the first paper in the results and clicked the download button on the paper's arXiv page. After the paper was downloaded, the system returned the downloaded file's path. In the next turn, the user then asked how many times the keyword "HTML" is mentioned in the paper. Then Cognitive Kernel leveraged its private information management ability to open the paper and count the occurrences of "HTML." Then it returned the answer *5*, which we verified to be correct by opening the paper and manually searching the keyword.

6 Conclusion

In this paper, we presented Cognitive Kernel, a dockerized agent system towards the goal of general-purpose "Autopilots", which has three main components: reasoning kernel for decision-making, perception kernel for state perceiving, and memory kernel for state management. We further reorganized these components into several dockers for easy and safe deployment. We evaluate Cognitive Kernel's ability to handle real-time information, private information and long-term histories. The results show that Cognitive Kernel achieves stronger or similar performance compared to other SOTA closed-source systems. We release our system framework, model weights, and findings to the community, hoping to inspire future research on the direction of generalist "Autopilots" systems.

Broader Impact Statement

Since Cognitive Kernel directly interacts with the real world (through the web browser and host file system[5]), it might introduce additional risks for the user. For example, accessing confidential information on the host system or downloading malicious content from the internet. Thus it requires extra caution and a substantial amount of safe checks when deploying and running the Cognitive Kernel. In our experiments that involve the open web, we also closely monitor the Cognitive Kernel 's progress to make sure that the system is not causing any harm to the website hosts.

Also, we would like to note that Cognitive Kernel is a research project and not an official product from Tencent. Our intention is to explore the possibility of building "Autopilot" systems and we hope our work could inspire others and bring more positive social impact. Any misuse of the system that could be potentially harmful to others is strictly forbidden.

References

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45.

Arthur Charpentier, Romuald Elie, and Carl Remlinger. 2021. Reinforcement learning in economics and finance. *Computational Economics*, pages 1–38.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *CoRR*, abs/2310.05915.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2024. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 15159–15177. Association for Computational Linguistics.

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, Zheng Xiong, Yicheng Luo, Jianye Hao, Kun Shao, Haitham Bou-Ammar, and Jun Wang. 2023. Pangu-agent: A fine-tunable generalist agent with structured reasoning. *CoRR*, abs/2312.14878.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien

[5]By default, Cognitive Kernel's dockers are isolated such that they could not impact the host system. However, one could bind the host system's file path with docker's file path, e.g. for debugging purposes, which would make it possible for Cognitive Kernel to directly access the host file system.

Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Anthony Celi. 2019. Guidelines for reinforcement learning in healthcare. *Nature medicine*, 25(1):16–18.

Significant Gravitas. 2024. Autogpt.

Izzeddin Gur, Hiroki Furuta, Austin V. Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6864–6890. Association for Computational Linguistics.

Ronald A Howard. 1960. Dynamic programming and markov processes.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2024. Chain of code: Reasoning with a language model-augmented code emulator. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2023. BOLAA: benchmarking and orchestrating llm-augmented autonomous agents. *CoRR*, abs/2308.05960.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder:

Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. LASER: LLM agent with state-space exploration for web navigation. *CoRR*, abs/2309.08172.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y. Zhao, Yi Luan, Keith B. Hall, Ming-Wei Chang, and Yinfei Yang. 2022. Large dual encoders are generalizable retrievers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 9844–9855. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. 2024. Webcanvas: Benchmarking web agents in online environments. *CoRR*, abs/2406.12373.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *CoRR*, abs/2406.12373.

Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Kumar Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. In *Autonomous Vehicles and Machines 2017, Burlingame, CA, USA, January 29 - February 2, 2017*, pages 70–76. Society for Imaging Science and Technology.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *CoRR*, abs/2404.12253.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Alan Mathison Turing et al. 1936. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5.

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. 2024a. Openchat: Advancing open-source language models with mixed-quality data. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024b. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024c. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8:279–292.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024a. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024b. Stateflow: Enhancing LLM task-solving through state-driven workflows. *CoRR*, abs/2403.11322.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024c. Os-copilot: Towards generalist computer agents with self-improvement. *CoRR*, abs/2402.07456.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *CoRR*, abs/2406.04151.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *CoRR*, abs/2404.07972.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *CoRR*, abs/2405.15793.

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

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Raghavi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12380–12403. Association for Computational Linguistics.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. Assistantbench: Can web agents solve realistic and time-consuming tasks? *CoRR*, abs/2407.15711.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3053–3077. Association for Computational Linguistics.

Tianhua Zhang, Jiaxin Ge, Hongyin Luo, Yung-Sung Chuang, Mingye Gao, Yuan Gong, Yoon Kim, Xixin Wu, Helen Meng, and James Glass. 2024. Natural language embedded programs for hybrid language symbolic reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: less is more for alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Anni Zou, Wenhao Yu, Hongming Zhang, Kaixin Ma, Deng Cai, Zhuosheng Zhang, Hai Zhao, and Dong Yu. 2024. DOCBENCH: A benchmark for evaluating llm-based document reading systems. *CoRR*, abs/2407.10701.

Appendix

Notes

1. https://chatgpt.com/
2. https://react.dev/
3. https://nginx.org/en/
4. https://jupyter.org/
5. https://playwright.dev/
6. https://huggingface.co/docs/text-generation-inference/en/index
7. https://www.postgresql.org/
8. https://www.sqlite.org/
9. https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2
10. https://chatgpt.com/
11. https://gemini.google.com/
12. https://claude.ai/chat/
13. https://kimi.moonshot.cn/chat/
14. https://www.coze.com/

A Backgrounds

We first introduce previous efforts on rule-based automated systems, which motivates our formulation of an "autopilot" system. Then we cover more recent works in building and training task-specific agent systems.

A.1 From Rule-based Automated Systems to LLM-driven "Autopilots"

One of the automated system pioneers is the Turing machine (Turing et al., 1936), a mathematical model describing an abstract machine that manipulates symbols based on a table of rules and a foundation of modern computers. A Turing Machine contains two key concepts: states, the set of predefined variables, and transition functions, which are predefined rules describing how a system shall respond to a state. Although traditional Turing machines cannot directly serve as an "autopilot" system to solve real-world tasks due to the almost infinite space of possible states and transition rules with randomness among them, it motivates the two fundamental duties of an "autopilot" system: (1) perceiving and managing essential states; (2) make wise decisions based on the states.

LLMs implicitly learn the possible connection between states by compressing and modeling vast amounts of world information. They thus can serve as an approximation of the transition functions and partially solve the second duty (Wu et al., 2024b; Ma et al., 2023). However, how to efficiently and accurately perceive and manage the state remains unclear. Daily applications require both global state information (e.g., world knowledge) and localized state information that changes constantly (e.g., update-to-date knowledge or private information). LLMs can learn to model the global state information but cannot model the localized ones. Thus, how to manage the localized state information becomes the critical design choice for building an LLM-driven AI system. For a Copilot system, we leave the task of providing localized state information to users, which simplifies the task but also limits the capability of the system to solve tasks independently (Chen et al., 2021). In contrast, for an "autopilot" system, we expect it to monitor and acquire the localized state information by itself and thus it has the potential to solve a complete task without human involvement (Gravitas, 2024). Therefore, in Cognitive Kernel , we specifically designed the perception kernel and the memory kernel to perceive and memorize localized state information, and their orchestration is completely handled by the reasoning kernel, moving one step closer to an LLM-driven "Autopilot" system.

A.2 Recent Advancement in Model-based Agents

The concept of an agent, popular in the reinforcement learning (RL) community, has been foundational to artificial intelligence (Watkins and Dayan, 1992; Kaelbling et al., 1996). An agent interacts with an environment to learn how to achieve a specific goal by maximizing cumulative rewards (Arulkumaran et al., 2017; Sutton and Barto, 2018). Early work focused on the theoretical underpinnings of RL, such as Markov decision processes and dynamic programming (Howard, 1960). These methods provided the basis for the development of various RL algorithms. Recent advances like Q-learning (Watkins and Dayan, 1992) and policy gradients (Williams, 1992; Sutton et al., 1999) have been crucial, especially with the introduction of deep RL, which combines neural networks for approximating Q-values, as seen in Deep Q-Networks (Van Hasselt et al., 2016) achieving human-level performance in complex tasks. Key algorithms such as DDPG (Lillicrap et al., 2016) and PPO (Schulman et al., 2017) have significantly impacted robotics, healthcare, finance, gaming, and autonomous driving, showcasing RL's broad applicability (Gottesman et al., 2019; Yu et al., 2021; Charpentier et al., 2021; Silver et al., 2018; Sallab et al., 2017; Kiran et al., 2021).

However, previous assumptions in RL significantly differ from human learning processes, as the human mind is highly complex and capable of learning from a much wider variety of environments. With the recent developments of large language models (LLMs), the concept of agents has evolved beyond simple policy functions in restricted environments (Xi et al., 2023; Wang et al., 2024b). LLM-based agents can possess more comprehensive world knowledge, perform more informed actions, and also provide natural language interfaces for human interaction, making them more flexible and explainable (Yao et al., 2023; Liu et al., 2023; He et al., 2024; Gur et al., 2024; Yang et al., 2024). For example, ReAct (Yao et al., 2023) prompts LLMs to perform dynamic reasoning to create, maintain, and adjust high-level action plans (reason to act), while also interacting with external environments (e.g., Wikipedia) to incorporate additional information into reasoning (act to reason), thereby achieving superior performance in benchmarks. However, prompting-based agent frameworks still perform poorly in many real-world agent scenarios.

To make LLM-agents task experts, a series of works have focused on collecting expert trajectories from diverse environments and tasks, and training LLM-based agents through behavioral cloning (Zeng et al., 2024; Chen et al., 2023). However, obtaining these expert trajectories is often costly and lacks sufficient exploration. Another line of work involves training LLM-based agents based on environmental feedback using RL methods to align LLMs with agent task objectives (Christianos et al., 2023). Additionally, some approaches utilize self-improvement, where the model explores the environment to obtain high-reward trajectories and fine-tunes itself based on these trajectories (Putta et al., 2024). Although these training strategies have shown promising performance in reasoning, coding, and web tasks, these trained agents remain constrained to their specific tasks, struggle to generalize to general-purpose usage, and are restricted to pre-defined environments.

Existing efforts on LLM-driven agents often follow an environment-centric design, where they partially solve this problem by designing an environment for each task to manage and feed localized state information (Liu et al., 2024; Xi et al., 2024). Moving one step further and aiming to create a general-purpose "autopilot" system that has the potential to solve more general tasks, Cognitive Kernel switchs from environment-centric to model-centric and asks the system to actively perceive the localized state information with tools an ordinary person could use.

B A Conceptual Framework of "Autopilot" Systems

Motivated by previous automated systems (Turing et al., 1936), an autopilot system AS should excel at managing the current state and making wise actions accordingly. Thus, we first formulate the conceptual autopilot framework as a 6-tuple $AS = \langle \mathcal{S}, s_n, \mathcal{A}, a_n, T, \mathcal{M} \rangle$, where $\mathcal{S}$ is the set of all possible states, $s_n \in \mathcal{S}$ is the state[6] at timestamp n, $\mathcal{A}$ is the set of possible actions, $a_n \in \mathcal{A}$ is the action at timestamp n, T is the transition matrix, which determines a_n based on s_n, $\mathcal{M}$ is the memory component that records s_0 to s_{n-1}. Note that in real applications, $\mathcal{S}$ and $\mathcal{A}$ are arbitrarily large to be enumerated by modern machines. As the size of T is $|\mathcal{S}| \cdot |\mathcal{A}|$, T is also too large to be enumerated.

To address these limitations and create a practical autopilot system, we further decouple the states into global and localized ones, where the global state information is the world knowledge shared by most humans and localized state information is temporally or spatially unique to the current task. Specifically, we can decouple $\mathcal{S}$ as:

$$\mathcal{S} = \mathcal{S}^g \cup \mathcal{S}^l, \tag{1}$$

where $\mathcal{S}^g$ and $\mathcal{S}^l$ represent the set of global and localized state information, respectively. Similarly, for any state s_n at timestamp n, we can also decouple it as:

$$s_n = s_n^g \cup s_n^l. \tag{2}$$

Based on the assumption that large language models such as GPT-4 have compressed the world's knowledge through the pre-training, we can use an LLM as the policy function F to simulate $\mathcal{S}^g$, s_n^g, and T. Thus, we can reformulate AS as $\langle \mathcal{S}^l, s_n^l \mathcal{A}, a_n, F, \mathcal{M} \rangle$, where F is the LLM-based policy function that predicts a_n conditional on s_n. Since LLMs are essentially probabilistic models,

[6]Each state is a set of variables and their values. In this paper, we omit the concept of variables for simplicity.

this formulation no longer guarantees the execution correctness, and thus a_n can only be viewed as the most likely action based on the trajectories that F has seen during the training phase.

A remaining challenge is where to get the localized state information in a real system. As discussed in Section 1, a key difference between autopilot systems and copilot systems is that the state information might not be provided and the system must perceive the state information actively. To better represent this, we add one more variable $s_n^o \subseteq s_n^l$ to denote the observed localized states at each step n. Empirically, s_n^l is the optimal local state that one can not easily get, and s_n^o is the localized state information the autopilot system truly has and can rely on. At each step n, F will first determine whether s_n^o is close enough to s_n^l, if it is not close enough, F will initiate a perception task to perceive more localized states. Since each perception task might be an autopilot task requiring further state perception and planning, we denote it as P^k, where k is the depth of the perception tasks. Thus, we can get the final formal formulation of AS as:

$$\begin{aligned} AS &= \langle \mathcal{S}^l, s_n^l, s_n^o, \mathcal{A}, a_n, F, \mathcal{M}, P^0 \rangle, \\ P^k &= \langle \mathcal{S}^l, s_n^l, s_n^o, \mathcal{A}, a_n, F, \mathcal{M}, P^{k+1} \rangle. \end{aligned} \tag{3}$$

In Section 2, we will cover the implementation details of Cognitive Kernel and explain how it fulfills the above formulation.

C Design of the Memory Kernel

The overall framework of the memory kernel is illustrated in Figure 5, which includes two major components: information processing/storage and information retrieval.

The information processing/storage component is illustrated in the right part of Figure 5 (the blue rectangle). For any given information, we first convert it into plain text and treat it as a regular document. For example, for dialogue history with the timestamp information, we could create a sentence in the format of "C@@T," where C and T represent the content and timestamp, respectively. The documents are parsed into different granularities:

- Documents. An example of document is denoted as doc_emb in Figure 5;
- Propositions (Chen et al., 2024). The document d in the bottom right corner of Figure 5 can be broken down into propositions of p_1 = "The Yellow River is in China," p_2 = "The length of Yellow River is 5,464 km," ... We denote embedding representations of all propositions as (prop_1_emb, ...).
- Key concept and perspective. For example, the concept and perspective for p_1 is "Yellow River" and "country", and the concept and perspective for p_2 is "Yellow River" and "length". To facilitate the semantic matching, we concatenate the concept and perspective as a phase and then compute the embedding representation (c_p_1_emb, ...).
- Mentioned Concepts. For example, the mentioned concepts for p_1 are "Yellow River" and "China", and the mentioned concepts for p_2 are "Yellow River".

Information retrieval. The information retrieval part is shown in the left-side of Figure 5 Given an input query q, we first compute its embedding Query_emb and extract key concepts and perspectives with the same models we used for the processing step. Then Cognitive Kernelconducts the multi-granularity matching with the following granularities:

- Document-level soft matching finds the most relevant documents based on the similarity between Query_emb and all doc_emb;
- Proposition-level soft matching finds the most relevant propositions based on the similarity between Query_emb and {prop_1_emb, ...} and then retrieves the corresponding documents;
- Concept-level soft matching finds the most relevant concept+proposition combinations based on the similarity between Query_emb and {c_p_1_emb, ...} and then finds the corresponding proposition and documents;
- Concept-level Hard Matching finds all documents that share mentioned concepts with query q.

Finally, Cognitive Kernelreranks the retrieved documents based on their maximum similarity according to the above four matching methods. For example, if the retrieved result of document-level soft matching is [(doc A, 0.8), (doc B, 0.7)] and the retrieved result of proposition-level soft matching

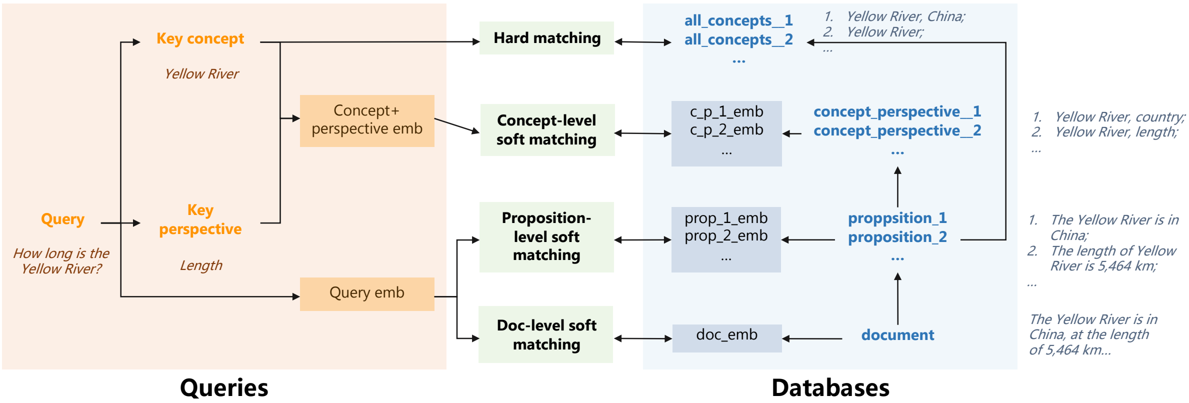

Figure 5: The overall framework of the multi-granularity information management system.

is [(doc B, 0.9), (doc C, 0.6)], then the merged result of the two matching methods is [(doc B, 0.9), (doc A, 0.8), (doc C, 0.6)].

D Experiments (Additional Details)

D.1 Trainng Details

Pre-trained Model We train our own policy models based on Llama3 series (Dubey et al., 2024)

Hyper-parameters The hyper-parameters for learning the 70B and 8B policy models are listed in Table 2.

Training Data Table 3 displays the training data statistics for the proxy model, primarily divided into conventional instruction-following data and agent data.

D.2 Benchmarks

For real-time information management evaluation, we conduct experiments on the recently released WebCanvas benchmark (Pan et al., 2024). WebCanvas test set contains 104 human-annotated tasks that require interacting with real-world live websites to complete, and each of the tasks specifics a target website for interaction. We simply provide each task instruction to each system and it is expected to strictly follow the instructions to find the target webpage and complete the task.

For private information management evaluation, we conducted extensive assessments using DocBench (Zou et al., 2024). DocBench provides an end-to-end evaluation: starting with a raw file input along with user questions and evaluating the system based on the quality of the answers generated. This benchmark includes 229 real-world documents and 1,102 questions, spanning five distinct domains: Academia, Finance, Government, Law, and News. Furthermore, it encompasses four major types of questions: text-only, multi-modal, meta-data, and unanswerable questions. We simply upload the files to each system and ask the questions.

For long-term memory management evaluation, we use two benchmarks: LLM-generated test cases and human-written test cases. Specifically, each test case consists of a session of messages between the user and the assistant acting as dialog history, followed by a final question about the details of previous messages. The goal is to assess if the assistant can accurately retrieve the ground truth message(s) from the dialog history and correctly answer the query. The benchmarks has four categories of questions: Single Message (1-M), where answering the final question relies on a single ground truth message; Multiple Messages (Mul-M), where answering the final question requires combining information from two or more ground truth messages; Knowledge Update (Update), where the user initially provides some information and later updates or corrects it. Temporal Reasoning (Temp), where the questions require inferring the chronological order of two events mentioned in conversation history. For the LLM-generated test cases, we use the LongMemEval (Wu et al., 2024a), which prompts the LLM (i.e., GPT-4o) to first create users with particular attributes, preferences, and experiences, then we have an LLM role-plays the user while another LLM role-plays the assistant to create the dialog history. For the manually-created test cases, we write both the dialog history and the final question and we only cover the first three categories. Finally, an LLM is prompted to generate the question and the answer. The generated conversations, questions, and answers are manually checked by

	Learning Rate	Sequence Length	Warmup Step	Global Batch Size	Epoch
70B	1e−5	8192	0	64	3
8B	5e−5	8192	50	64	3

Table 2: Hyper-parameters for the 70B and 8B policy models.

	# Samples
Instruction-following	33,882
Agent	49,996
Total	83,878

Table 3: Training data statistics for the policy model.

humans to ensure quality. During the evaluation process, we manually enter the dialog history (user turns) to the tested systems, then start a new chat session and ask the final question.

D.3 Metrics

We mainly focus on the end-task success rate in our evaluation. However, the definition of task success and evaluation methods are different for each of the target scenarios. Here we provide the detailed definitions.

For WebCanvas, (Pan et al., 2024) proposed step-wise scores that use human-annotated key nodes along the gold trajectory to evaluate the system's web browsing performance. Upon further inspection of the annotated key nodes, we found that this metric can significantly underestimate the system performance. Since there exist many possible trajectories that lead to task completion, simply matching the key nodes from one trajectory can overlook many other valid paths. Thus we opt for manual evaluation and we only focus on the overall task success rate. Here, since many tasks cannot be truly "completed," we consider a task to be successful if 1) the required information is gathered from the target website and 2) all the necessary actions are performed with regard to the correct elements. For example, the system cannot truly buy a gift card (without valid account information), we consider it to be successful if it has added the correct gift card to the cart on the target website and filled in the fake user information in the instruction to the correct cells on the checkout page. We closely monitor the system's web interaction sessions during our experiments to ensure that no harm is done to website hosts. For systems that do not provide the intermediate trajectories, we consider a task to be successful if the system provides the links to the correct websites that satisfy all requirements in their responses.

For DocBench, we follow (Zou et al., 2024) and adopt GPT-4o to automatically evaluate the correctness of the generated answers based on the reference. As reported by (Zou et al., 2024), relying on string matching or number extraction to evaluate the accuracy of generated response can be imprecise, since different LLMs and systems exhibit substantial variations in the organization and style of their outputs, potentially leading to biases in traditional metrics. We instruct GPT-4o to assign a score of 0 (incorrect) or 1 (correct), thus using Accuracy to measure system performance.

For long-term memory evaluation, we ask Cognitive Kernel as well as baseline systems to generate the answer for each question, and manually judge the correctness of the answers.

E In-depth Analysis

E.1 Real-time information management

We further conduct experiments using a specialized web agent system specifically designed for web interaction. In particular, we rerun the WebCanvas agent (Pan et al., 2024) using the GPT-4o API as the backbone and then manually evaluate its trajectories. We further categorized the failure cases for WebCanvas agent and Cognitive Kernel to better understand the limitations of existing systems. Specifically, we define the following categories: **Missing Detail**: the system almost succeeded but missed a detail in the instruction. **Reasonable Attempt**: the system makes reasonable actions on the webpage (similar to a human when navigating an unseen website), but runs out of max steps before completing the task due to unfamiliarity. **Completely Failed**: the system's action trajectory does not make much sense. **Blocked**: the system's web browsing is blocked by bot detectors or Captcha.

From Figure 6, we see that the WebCanvas Agent system coupled with GPT-4o achieves a much higher success rate than baselines shown in Table 1, showing that the model capability is not the only factor that affects end-task performance. Also, in addition to the highest success rate, Cognitive Ker-

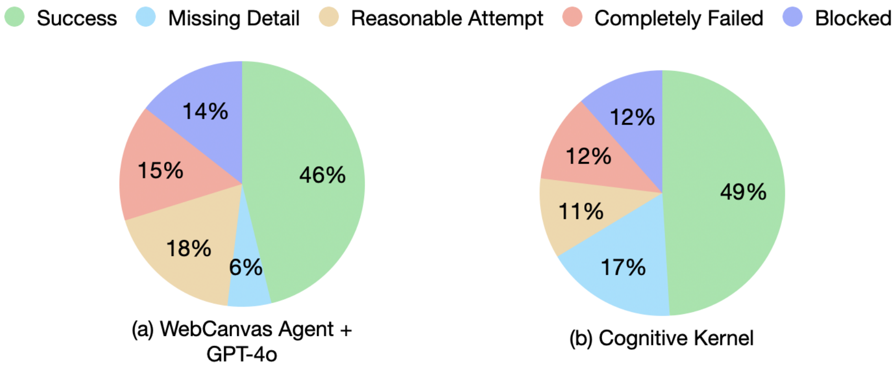

Figure 6: Overall task completion results on the WebCanvas test set.

nel also has a much larger portion of cases where the system almost completed the task, suggesting the overall superiority of the system. We showcase an example of **Missing Detail** in Figure 8 and an example of **Reasonable Attempt** in Figure 9 in the appendix.

E.2 Private information management

Figure 7 presents a radar plot illustrating the accuracy of various end-to-end LLM-based systems and open-source LLMs using a *parse-then-read pipelines* across different question types in the DocBench. The questions in DocBench are categorized into four major types: Text-only, Multimodal (including tables and figures), Meta-data, and Unanswerable questions.

The left subfigure compares these systems with the end-to-end systems presented in Table 1. Diving into this detailed comparison, we can observe the specific capabilities of these systems in handling different document-based questions. As shown, Kimi-Chat and Claude3-opus perform well across all these categories, demonstrating balanced performance on different types of questions. Notably, GPT-4 underperforms in the unanswerable category, suggesting potential overfitting in GPT-4's optimized file systems, likely due to training on datasets that only include answerable questions with provided golden answers. On the other hand, Gemini-Pro 1.5 struggles with figure-based questions in documents, with its performance in the multimodal category primarily driven by table-based questions. Coze (GPT-4o) performs poorly in handling user documents due to misalignment between system instructions and the LLM's instruction-following capabilities. Other systems show relatively balanced performance, with gaps mainly attributable to the backbone LLMs and their system designs. As shown in the right subfigure, we also compare the results with recent state-of-the-art open-source LLMs using *parse-then-read pipelines*. To enable LLMs to process documents as input, we use the `fitz` package to extract text, tables, and images from PDF files, then feed the questions, along with the extracted information, into the LLM to obtain the final answer. It is evident that simply using these LLMs does not lead to strong performance on document-based tasks. Compared to Llama-3 70B, Cognitive Kernel demonstrates a 22% relatively improved performance, highlighting the importance of system design in handling diverse types of user questions.

E.3 Long-term memory management

The detailed evaluation results of Cognitive Kernel, along with baseline systems on long-term memory, are presented in Table 4. While GPT-4o is generally more powerful than GPT-4o-mini, it performs significantly worse in long-term memory evaluation. Upon closely examining the intermediate content within the long-term memory module, we discovered that GPT-4o is more prone to modifying or overwriting existing memories when receiving new input from the user, even when the new content is only semantically similar to the old memory rather than actual update. As a result, the system may lose the ability to answer the final question accurately. Additionally, Coze does not perform as well as GPT in this setting. The choice of the underlying base model also has a significant impact on Coze's performance, with Coze + GPT-3.5-turbo performing substantially worse than Coze + GPT-4o. In our system, we found that the Cognitive Kernel's performance is suboptimal when using GPT-4o as the base LLM. This is likely because GPT-4o is not fully aligned with our system prompt

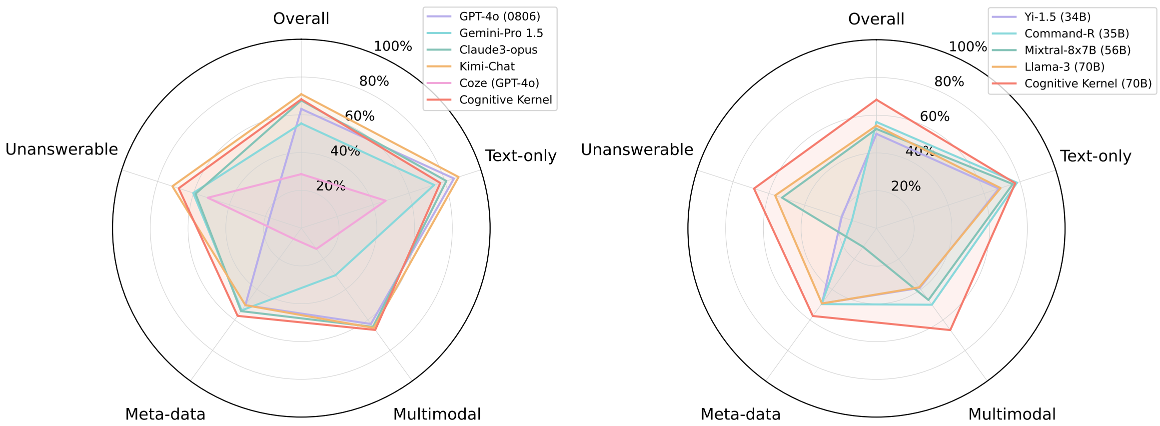

Figure 7: Performance comparison of various end-to-end systems (left) and open-source LLMs across different question types in the DocBench.

Systems	Human-written test cases				LLM-generated test cases					Avg
	1-M	Mul-M	Update	All	1-M	Mul-M	Update	Temp	All	
GPT-4o-mini	70.0	70.0	80.0	73.3	100.0	71.4	68.0	66.7	75.9	74.6
GPT-4o	60.0	50.0	70.0	60.0	76.0	40.5	84.0	45.8	58.6	59.3
Coze (GPT-3.5-turbo)	60.0	50.0	60.0	56.7	48.0	9.5	36.0	8.3	23.3	40.0
Coze (GPT-4o)	80.0	80.0	90.0	83.3	68.0	14.3	24.0	37.5	32.8	58.1
Cognitive Kernel (GPT-4o)	70.0	50.0	50.0	56.7	68.0	73.8	40.0	54.2	61.2	59.0
Cognitive Kernel	100.0	90.0	90.0	93.3	100.0	81.0	84.0	45.8	78.4	85.9

Table 4: The results of long-term memory management.

even though the prompt is natural and accurate for humans. After switching to the adapted LLM, the performance improved from 59.0% to 85.9%. This observation shows that there is no perfect model and a continuously evolving AI system is crucial in real applications.

E.4 Error Cases

In Figure 8, we illustrate an example of **Missing Details** from the WebCanvas test set. Given the query, the system first searched for the director of Smile and found it to be Parker Finn. Then it tried to search for other movies directed by Parker Finn, and it found the answer from the IMDB website. However, the initial query requires the answer from the TVGuide website, thus the task is not considered as successful.

In Figure 9, we show an example of **Reasonable Attempt**. The task requires checking the iPhone repair status, and it would be considered successful if it can reach the Apple My Support page [7] (upon signing in one can check the status there). In this case, Cognitive Kernel examined different pages on the Apple website that are relevant to iPhone repair but could not find the page with explicit information for the status check. After reaching the maximum number of steps it is forced to stop. We believe such errors are reasonable and could be potentially solved by allowing the system to explore the website before executing any tasks and update its memory accordingly.

F Discussions and Limitations

Despite that Cognitive Kernel achieves promising performance on several realist tasks, there is still a huge gap between Cognitive Kernel and a generalist "autopilot" system. In this section, we discuss the limitations of our current system, which is also our future working directions.

F.1 Multi-modal Perception Ability

For a generalist "Autopilot" system, it's critical to have multi-modal perception ability because the real world is multi-modal and the rich information that helps decision-making is often embedded in other modalities beyond text. However, current Cognitive Kernel employs an LLM as the central policy model, thus it cannot handle multi-modal inputs such as images or audio. Intuitively, all three

[7] https://support.apple.com/my-support

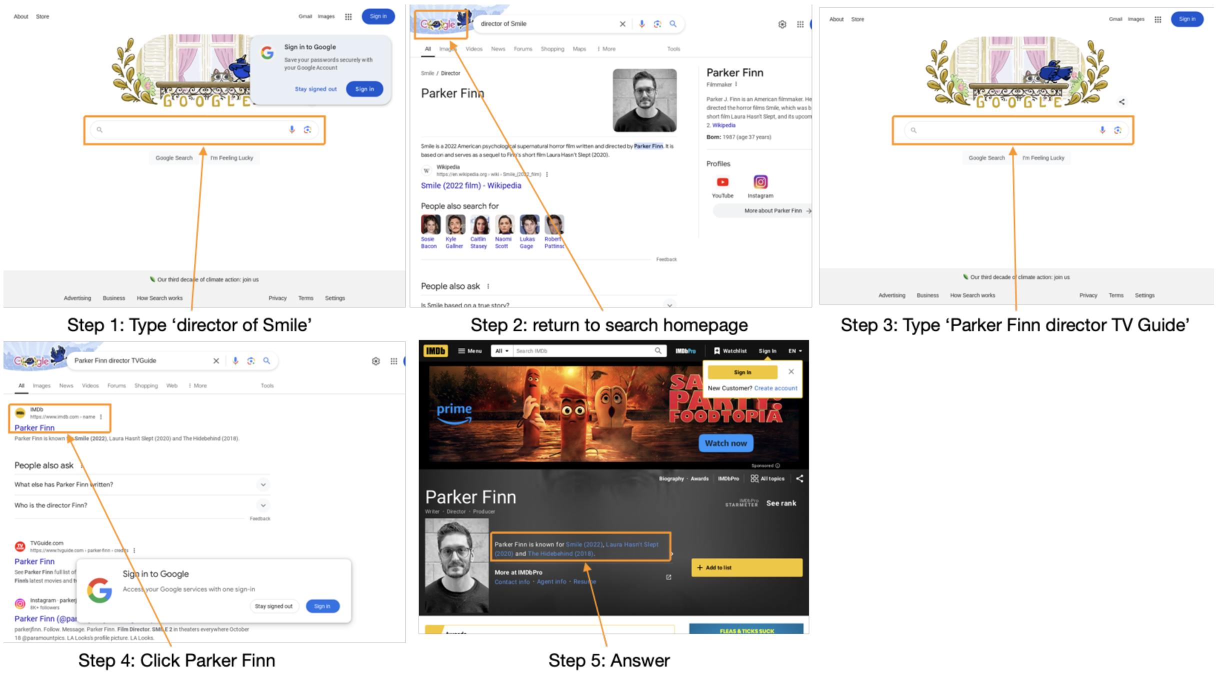

Figure 8: An example trajectory of **Missing Details** from Cognitive Kernel. The task instruction is "*Find more films from the director of Smile on tvguide*"

use cases we experimented with could greatly benefit from other modalities and provide a better user experience. For example, for private information management, the system can better understand local files by reading both the text and images embedded in the files. For real-time information management, the system can better understand the websites by observing the visual layout and visual elements not captured in the accessibility tree, therefore navigating the web more effectively. For long-term memory management, the user can also send images to the system or speak to it directly without typing any text, and the system can read and write memories of different modalities to better serve different scenarios. Therefore it is a promising direction to equip Cognitive Kernel with multi-modal perception ability and we leave this exploration for future work.

F.2 Self-improvement Through Search and Feedback

Even though we give Cognitive Kernel freedom to be a generalist autopilot system, it tends to mimic the training trajectories. Thus, an important research question is how to make it capable of continuously evolving to overcome the limitation of the limited training trajectories such that it could generalize to unseen tasks. Similar to previous efforts on reinforcement learning, a promising way towards this goal is by letting the system search for different task-solving strategies in unseen situations, automatically collect feedback signals for different trajectories, and orchestrate the learning and system updates autonomously. The most critical component in this pipeline is the feedback signal. Currently, Cognitive Kernel still relies on users or external models to provide the feedback, which limits its scalability. Ideally, the system should also acquire the ability to be a critic, such that it can serve as a value function to estimate the quality of its own task-solving trajectories. With such ability, the system could adopt algorithms such as Monte-Carlo Tree Search (MCTS) (Tian et al., 2024) to explore the real-world environments, collect the self-feedback, and continue improving itself with this signal. We believe this direction is a necessary path toward a true "Autopilot" system and warrant further investigation.

F.3 Robust System-level Support

Since the purpose of an "Autopilot" system is to complete various user tasks in the real world, it requires much more than a powerful policy model. In other words, the system needs a robust and scalable infrastructure to support the central policy model to function as the "brain" an intelligent agent, much like the relationship between the human body and the brain. With this in mind, we adopted a dock-

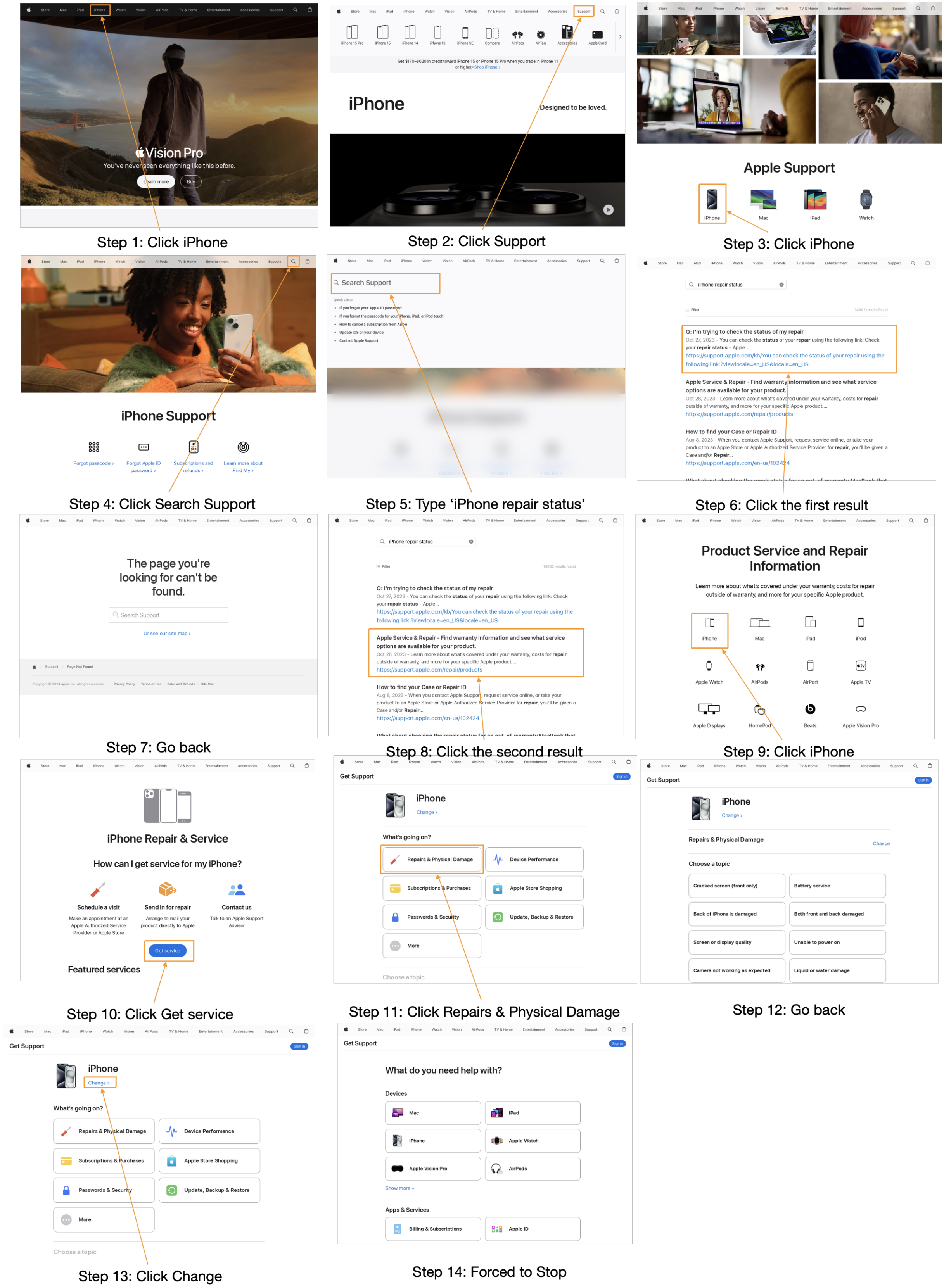

Figure 9: An example trajectory of **Reasonable Attempt** from Cognitive Kernel. The task instruction is "*Check the status of your iPhone repair on apple.*"

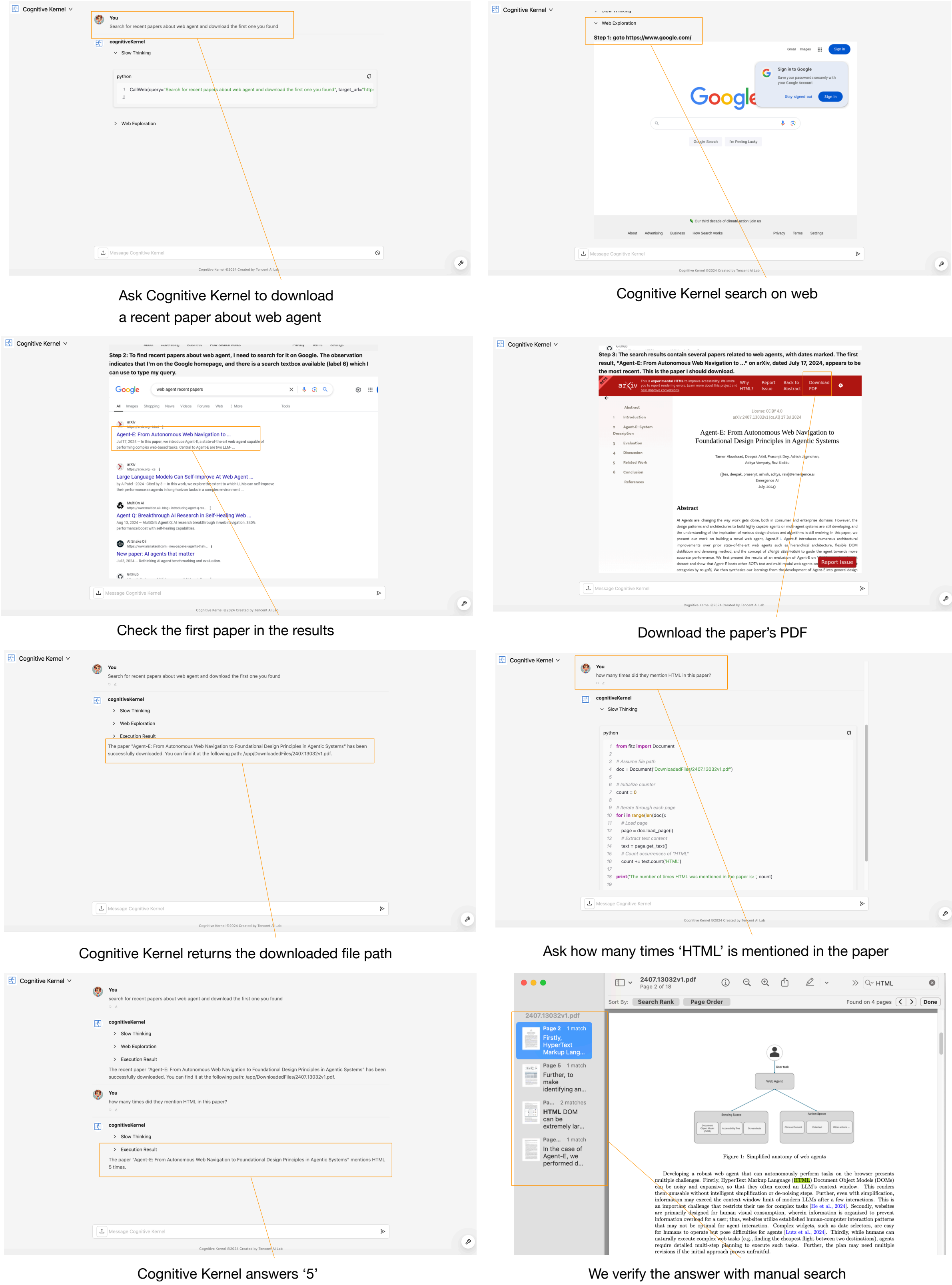

Figure 10: An example trajectory of Cognitive Kernel on completing user queries leveraging both real-time information management and private information management abilities.

erized design and equipped Cognitive Kernel with various functionalities such as file processing and web interaction, as described in Section 2. However, we recognize that what we currently have is still quite limited and there is ample room for improvement in terms of both scope and robustness. For example, our system only supports user-uploaded file processing for private information management. To further expand the scope of applications, we could allow the system to directly access the local file systems and control other software on the operating system level (Xie et al., 2024; Wu et al., 2024c). In this way, we can enable the system to manage different kinds of private information for the user.

From the perspective of robustness, we also noticed that our system could not cover all edge cases in the environment. For example, when Cognitive Kernel leverages the browser to complete web-based tasks. In certain cases, despite the policy model predicting the correct action, the action could not be executed in the web server due to certain technical issues (e.g., elements in iFrames could not be directly clicked as other buttons). Concurrent works on web agents have also observed that such technical issues contribute to a large portion of failure cases (Yoran et al., 2024). We believe that improving the system's robustness is equally important as improving the capabilities of the policy model. Only the organic combination of these two can truly unleash the power of large language models and build "Autopilot" systems.