Note: The experimental section is currently in progress and will be updated in the subsequent versions.

Open-Creator: Bridging Code Interpreter and Skill Library

Junmin Gong*
gongjunmin@timedomain.ai
Timedomain

Sen Wang †
sayo@timedomain.ai
Timedomain

Wenxiao Zhao †
sean.z@timedomain.ai
Timedomain

Jing Guo‡
joe.g@timedomain.ai
Timedomain

ABSTRACT

AI agents enhanced with tools, particularly code interpreters, hold promising prospects for broad and deep applications. However, existing systems often require reinterpreting problems and generating new codes for similar tasks, rather than reusing previously written and validated functions, leading to inefficiencies in token usage and lack of generalization. In response to these challenges, we introduce *open-creator*, a novel AI agents framework bridging code interpreters and skill libraries. Open-creator is designed to standardize various inputs (including, but not limited to, dialogic problem-solving experiences, code files, and API documentation) into a uniform skill object format. This framework supports local saving, searching, and cloud uploading by users. Adopting a modular strategy, open-creator allows developers and researchers to create, share, and reuse skills without concerns over compatibility or version control. Furthermore, it offers flexibility in modifying, assembling, and disassembling created skills, which is crucial as skills may need updates over time and with changing environments. This mechanism allows AI agents to continually optimize skills based on feedback and new data. Our approach paves the way for more efficient and adaptable AI agent functionalities, contributing to the field's ongoing development. The open-creator code is publicly accessible at: https://github.com/timedomain-tech/open-creator.

1 Introduction

AI agents engage in complex reasoning by integrating planning [1, 2, 3, 4], decision-making, and the utilization of appropriate tools or APIs [5, 6]. However, these tools are typically predetermined and designed by humans, and the number of available tools is often limited due to the constraints on the input context length of Large Language Models (LLMs). To enhance the versatility of AI agents, a viable approach is to amalgamate the code-generation capabilities of LLMs with code execution functionalities. This integration allows for the flexible writing and execution of code to address specific user needs, embodying the role of Code Interpreters [7].

Given that LLMs occasionally generate erroneous codes—leading to low robustness and inability to meet user requirements—recent research has focused on enabling LLMs to auto-correct codes through environmental feedback [8, 9, 10, 11]. Additionally, there is emphasis on developing sophisticated projects through rational task decomposition and persistent memory. This focus has given rise to a plethora of AI agent frameworks, including MetaGPT [12], ChatDev [13], GPT-enginger [14], GPT-term [15], and codeplan [16]. These studies explore collaborative mechanisms among different roles, introduction of improved environments, enhanced feedback from agents, optimized task decomposition, and various engineering tricks, collectively contributing to the flourishing ecosystem of AI agents in the fields of Computer Science and Software Engineering. A comprehensive literature review in this area has been conducted by Wang et al [17].

	Learning	Outcome	Generalization	Interpretability
ANN	Parameter Optimization	Weights of NN	Mapping Function	Low
LLM	Instruction Fine-tuning	Instruction-based LLM	In-context Learning Prompt	Medium
Agents	Tool Creation	Skill Library	RAG	High

Table 1: Comparison of Different Learning Approaches

2 Related Work

Among the various contributions, two works, namely CREATOR [18] and VOYAGER [10], merit particular attention.

2.1 CREATOR

The CREATOR framework emerges as a noteworthy initiative addressing the intricate challenges faced by LLMs in tool utilization for problem-solving. Unlike conventional approaches, CREATOR uniquely positions LLMs not merely as users but as creators of tools. This pivotal transition enhances the models' capabilities in solving problems with heightened precision and flexibility, offering a fresh perspective on leveraging the tool-creating prowess of LLMs. The framework comprises four stages:

- **Creation:** LLMs are given explicit instructions and examples, guiding them to generate tools specific for given problems with focus on essential features.
- **Decision:** Post-tool creation, LLMs decide on tool application based on documentation and context analysis.
- **Execution:** This involves integrating the crafted tool code with the decision-making code, executing it through a code interpreter, and capturing the results or error messages.
- **Correction:** In case of execution failure, LLMs make necessary adjustments to the tool or decision, based on error tracking information.

The innovative approach of CREATOR brings to the forefront several advantages. Firstly, it alleviates the cognitive load on LLMs by distinctly separating the processes of abstract thinking (required for tool creation) and concrete thinking (necessary for decision-making). Secondly, the approach facilitates automated corrections through the strategic use of code testing and sensitivity analysis. Lastly, compared to LLM cache benefits [19], the validated tools crafted through CREATOR inherently possess superior generalization capabilities, making them adept at tackling a variety of new problems efficiently.

We consider the above as a new novel learning mechanism distinct from both the supervised learning in Artificial Neural Networks (ANNs) [20] and the few-shot in-context learning observed in LLMs [21]. Table 1 provides a concise summary of this proposition:

- For ANNs, the traditional approach involves learning weights through gradient descent algorithms to approximate an input-output mapping function. The generalization capability of ANNs primarily relies on the quality of the supervised data and the efficacy of the mapping function itself.
- In the case of LLMs, extensive pre-training on large corpora coupled with fine-tuning through specific instructions allows these models to significantly improve performance on downstream tasks, even with only a few examples for in-context learning.
- Regarding Agents, the learning process transitions into a procedure where the interaction experiences with users, which are aimed at problem-solving, are converted into persistent skills. Generalization, in this context, involves utilizing Retrieval-Augmented Generation (RAG) [22, 23, 24] to semantically search and select the most appropriate tools for the current context, as well as handling parameter passing effectively.

This *create and reuse* mechanism propels AI agents forward, capturing skills developed through LLMs. Unlike traditional deep learning, which often acts as a 'black box', this approach ensures skills are both reliable and interpretable. These skills are systematically stored in a library. When faced with new challenges, the system uses RAG techniques to select the best tools from this library. If no existing skill suffices, the system iteratively crafts new ones, ensuring continuous enhancement and adaptability in tackling intricate problems.

LLMs have a natural propensity to provide detailed steps and explanations. However, solving intricate problems under context constraints necessitates a more abstract skill set. Traditionally, users had to manually assemble operations specific to different domains, a process that was not only labor-intensive but also resulted in skill sets that were difficult to reuse, hindering the construction of complex, layered projects. The introduction of a framework for automated skill library creation substantially reduces the cost associated with manual tool design, facilitating the rapid and scalable creation and preservation of tools that can meet nuanced and customized requirements.

Furthermore, skills stored in the library can be chain-combined and refactored to develop new skills. The cost associated with creating new skills is minimized since the process can be automated: a task decomposition agent can simulate user demands, a code interpreter can craft the problem-solving process, and the resultant solution can be abstracted into a new skill.

2.2 VOYAGER

VOYAGER, on the other hand, provides empirical validation of the CREATOR concept within the context of the Minecraft game. It introduces a lifelong learning agent based on LLMs, designed for open-ended exploration tasks in Minecraft. This agent incorporates three innovative components:

- *Automatic Curriculum*: It utilizes GPT-4 to continuously generate new exploration tasks and challenges, fostering the acquisition of more complex skills by the agent. The curriculum dynamically adjusts the difficulty of tasks based on the agent's current status and progress.
- *Skill Library*: Whenever GPT-4 generates and verifies executable code that successfully completes a new task, this code is added to the skill library as a new skill. Each skill in the library is represented by code, which can be reused, interpreted, and combined to form more complex skills.
- *Iterative Prompting*: This mechanism refines the code generated by GPT-4 through execution, environmental feedback, error acquisition, and self-validation of task success. The iteration continues until the task is successfully completed, after which the refined skill is added to the skill library.

Experimental results demonstrate that VOYAGER can autonomously acquire various skills without human intervention, outperforming other LLM-based methods. It excels in obtaining unique items, unlocking milestones in the crucial technology tree more quickly, and traversing longer distances. Importantly, the skills learned by VOYAGER can be generalized to new Minecraft worlds, facilitating the completion of novel tasks.

Despite the innovative approaches of CREATOR and VOYAGER, each has its limitations. CREATOR falls short by not incorporating the concept of a skill library and does not discuss how created skills are stored and reused through a mechanism like RAG. On the other hand, VOYAGER, while effective, is specialized for the Minecraft environment, which might pose challenges when generalizing to other complex environments or tasks.

To advance the concepts of CREATOR and the skill library further, our open-creator introduces the following three key additions:

- *Consistency in Skill Schema:* A dedicated "skill_library" ensures consistency in user experience when facing challenges. Access to carefully curated and refined knowledge not only offers reliable solutions but also guarantees uniform results. This consistency is crucial when reflecting on and replicating successful processes by others, as inconsistent or unpredictable experiences can be frustrating. While CREATOR introduced methodologies, the absence of open-source code makes it challenging for others to replicate their success. The dedicated skill library in open-creator provides a standardized repository of skills that eliminates the usual inconsistencies associated with problem-solving, offering a robust and consistent user experience.
- *Skill Library Hub:* A significant downside of not having a cohesive skill library is the missed opportunity to leverage the collective wisdom of the global community. Innovative developers and users around the world continually discover optimized solutions to challenges. Without a centralized platform to archive and share these insights, there's a risk of continually reinventing the wheel. The skill library serves as a repository where community members can contribute, refine, and validate a diverse array of solutions, thereby amplifying the potential of shared knowledge and facilitating the creation of a robust knowledge base.
- *Skill Refactoring:* Open-creator allows for more flexibility in modifying, assembling, and disassembling created skills. Skills may need updates over time and with changing environments. The introduced mechanism allows AI agents to continually optimize skills based on feedback and new data, supporting the evolution of skills to meet emerging needs and challenges.

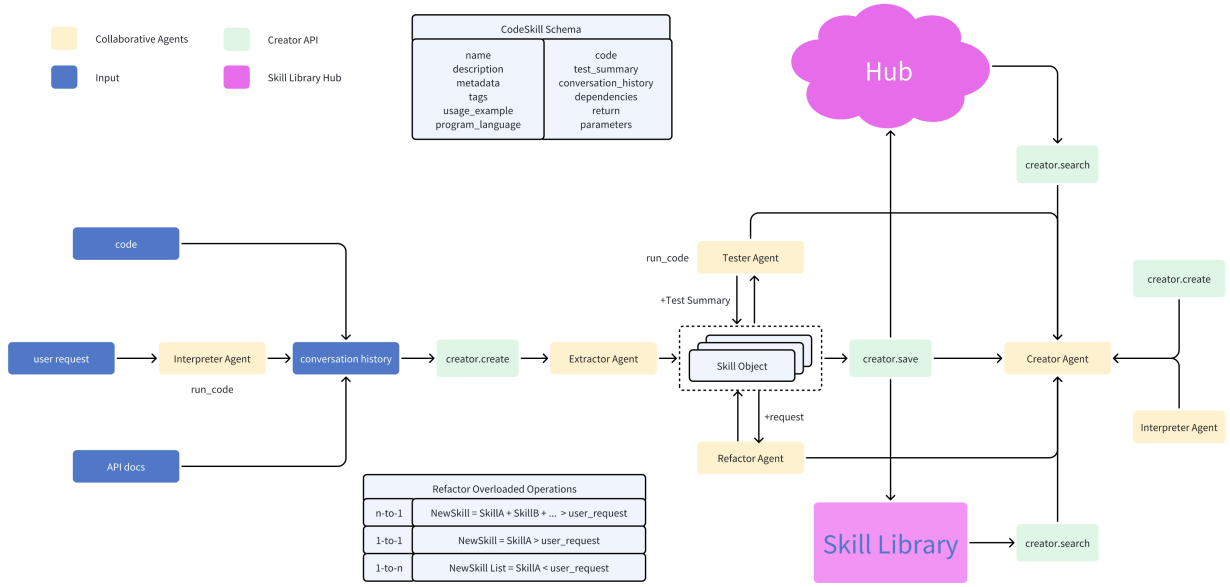

Figure 1: **The Overview of Open-Creator Framework**

3 Method

Open-Creator is an integrated package, incorporating all functionalities of CREATOR agents along with additional features such as saving to local or remote skill libraries and performing RAG searches from the skill library. Open-Creator is composed of three main components: Creator API, Collaborative Agents, and Skill Library Hub.

3.1 Creator API

The Creator API is a pivotal component of Open-Creator, serving as an essential interface for both developers and researchers focused on AI agents and the AI agents themselves. Designed with an emphasis on simplicity and user-friendliness, the API primarily offers three critical functionalities:

1. *creator.create*: This function stands out for its versatility, enabling the generation of unified skill objects from a wide range of sources. Users can derive skill objects from dialogues between users and agents that have code interpreter tools, representing the problem-solving processes. Also, they can craft these objects directly from sources such as code files, API documentation, specific problem requirements, or even by utilizing existing skills.

2. *creator.save*: Once the skills are formulated, they require a reliable storage solution. This function offers the flexibility for users to save their skill objects in diverse formats. Be it locally or on cloud platforms like the Hugging Face Hub, users have the freedom to choose their preferred storage method.

3. *creator.search*: The retrieval process is streamlined with this function. It begins by transforming the structured skills into vectors. Following this, a semantic search mechanism is employed to ensure users can retrieve the top k skills, ideally suited for tackling new problems.

3.2 Collaborative Agents

The Collaborative Agents encompasses five primary components:

1. **Extractor Agent:** Responsible for converting existing problem-solving experiences (typically dialogues with a code interpreter), textual content, and documents into a unified skill object. The skill object encapsulates skill name, description, use cases, input-output parameters, associated dependencies, and code language. The code within the historical records is modularized and reorganized.
2. **Interpreter Agent:** It leverages the open-source project, 'open-interpreter' [7], for prompt templates and code execution settings. The agent generates dialogue histories in the absence of known problem-solving procedures. Depending on execution results and user feedback, it preliminarily verifies the accuracy of results. The prompt templates of 'open-interpreter' utilize thought chains and the rewoo framework. Initial approaches to user queries involve incremental planning and task decomposition, followed by execution and retrospective outlining of the next steps. The ReWOO [2] framework delineates the language model's inference process from external tool invocations, significantly reducing redundant prompts and computational requirements.
3. **Tester Agent:** A variant of the interpreter agent, its primary role differs as it generates test cases and reports for stored skill objects. This evaluates their robustness and generalization performance, subsequently providing feedback to the interpreter for iterations.
4. **Refactor Agent:** This agent facilitates modifications based on user demands. A technique involving operator overloading elegantly represents skill amalgamation, fine-tuning, and modularization of complex skills. Instead of repetitively restructuring extensive skill inputs, a mathematical operation-based approach simplifies the interface. Skill objects can be accumulated, and the resultant skill objects are appended with natural language using symbols > or <. For instance, "skillA + skillB > user_request" represents the merging of two skills as per user demands. "SkillA < user_request" illustrates the modularization of a complex skill based on user requirements. For skill fine-tuning, "skillA > user_request" suffices.
5. **Creator Agent:** This agent orchestrates the usage of the Creator API interfaces and coordinates the operations of the above four agents in response to user queries and intents. It uses the search interface to retrieve relevant skills for problem-solving. If the retrieved skills are inadequate, it employs the create interface to devise a new solution, followed by the save interface to persist the new skill. It inherently supports direct operations on API interfaces and dispatches responses across various agents. The agent also employs overloaded operators for skill iterative updates and refactoring.

The Agents are developed on the langchain [25] platform and are optimized with LLM cache. They are progressively designed to support diverse open-source or proprietary API-based LLMs. Figure 1 aptly depicts their interrelationships.

3.3 Skill Library Hub

The Skill Library focuses on the persistent storage of skills. It employs a directory structure where each skill is stored in its named subfolder. Additionally, the advantages of the Hugging Face Hub community are harnessed to allow users to upload their private skill libraries to the cloud.

After users craft a skill, they have the option to save it on the cloud by providing a Hugging Face 'repo_id'. If the user hasn't established a repository, one is automatically forked from our pre-defined template. Following the fork, the user's skill is uploaded. To access skills from others' libraries, users need only supply the public repo_id and the designated skill directory name for downloading locally. After downloading, the search index is auto-updated to include the new skill. With the community's growth around the skill library, we've also introduced cloud-based searching, making it easier to tap into collective community insights.

4 Acknowledgements

We extend our heartfelt appreciation to Killian Lucas, the author of open-interpreter. We are also immensely grateful to the Discord community members, including warjiang, leonidas, AJ Ram, Papillon, minjunes, localman, jofus, Satake, oliveR, piotr, Grindelwald, Nico, MyRealNameIsTim, Pablo Vazquez, and jbexta, for their extensive discussions on the skill library and invaluable feedback. Their collective wisdom greatly enriched our work.

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[2] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models, 2023.

[3] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[4] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

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

[6] Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A benchmark for tool-augmented llms, 2023.

[7] KillianLucas. Open interpreter, 2023.

[8] Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting. *arXiv preprint arXiv: Arxiv-2303.14100*, 2023.

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

[10] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.

[11] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2023.

[12] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for multi-agent collaborative framework, 2023.

[13] Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development, 2023.

[14] Anton Osika et al. Gpt engineer, 2023.

[15] 101dotxyz. Gpteam: Collaborative ai agents, 2023.

[16] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D C, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. Codeplan: Repository-level coding using llms and planning, 2023.

[17] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents, 2023.

[18] Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Disentangling abstract and concrete reasonings of large language models through tool creation, 2023.

[19] zilliztech. Gptcache : A library for creating semantic cache for llm queries, 2023.

[20] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015.

[21] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey on in-context learning, 2023.

[22] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[23] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation, 2022.

[24] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey, 2023.

[25] langchain team. Hwchase17/langchain: building applications with llms through composability, 2023.