Agents: An Open-source Framework for Autonomous Language Agents

Wangchunshu Zhou[1*] **Yuchen Eleanor Jiang**[1*] **Long Li**[1*] **Jialong Wu**[1*]
Tiannan Wang[1] **Shi Qiu**[1] **Jintian Zhang**[1] **Jing Chen**[1] **Ruipu Wu**[1] **Shuai Wang**[1]
Shiding Zhu[1] **Jiyu Chen**[1] **Wentao Zhang**[1] **Ningyu Zhang**[2] **Huajun Chen**[2]
Peng Cui[3] **Mrinmaya Sachan**[3]

[1]**AIWaves Inc.** [2]**Zhejiang University** [3]**ETH Zürich**

Abstract

Recent advances on large language models (LLMs) enable researchers and developers to build autonomous language agents that can automatically solve various tasks and interact with environments, humans, and other agents using natural language interfaces. We consider language agents as a promising direction towards artificial general intelligence and release AGENTS, an open-source library with the goal of opening up these advances to a wider non-specialist audience. AGENTS is carefully engineered to support important features including *planning*, *memory*, *tool usage*, *multi-agent communication*, and *fine-grained symbolic control*. AGENTS is user-friendly as it enables non-specialists to build, customize, test, tune, and deploy state-of-the-art autonomous language agents without much coding. The library is also research-friendly as its modularized design makes it easily extensible for researchers. AGENTS is available at `https://github.com/aiwaves-cn/agents`.

1 Introduction

> "An autonomous agent is a system situated within and a part of an environment that senses the environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."
>
> *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents [Franklin and Graesser, 1996]*

Large Language Models (LLMs) [Brown et al., 2020, Ouyang et al., 2022, OpenAI, 2023] such as ChatGPT make it possible to build autonomous agents that can automatically solve complicated tasks and interact with the environment, humans, or other agents by perceiving, reasoning, planning, and acting in the world [Weng, 2023]. Language agents are a promising step towards artificial general intelligence (AGI) and can help reduce human effort in certain roles such as customer service, consulting, programming, writing, teaching, etc. Some recent demos such as AutoGPT [Richards and et al., 2023] and BabyAGI [Nakajima, 2023] have demonstrated the potential of language agents and have gained massive interest from developers, researchers, as well as more non-technical audiences.

While intriguing, most of these demos or repositories are not friendly for *customizing*, *tuning*, and *deploying* new agents even for experienced developers or researchers. This limitation comes from the fact that these demos typically proof-of-concepts showcasing the possibility of language agents, instead of being larger frameworks that can be used to build and customize language agents over time. Moreover, most of these open-source repositories only cover a small portion of the core abilities of language agents including task decomposition [Nye et al., 2022], long-short term memory [Zhou

[*]Equal Contribution. Correspondence to: chunshu@aiwaves.cn

Preprint. Work in progress.

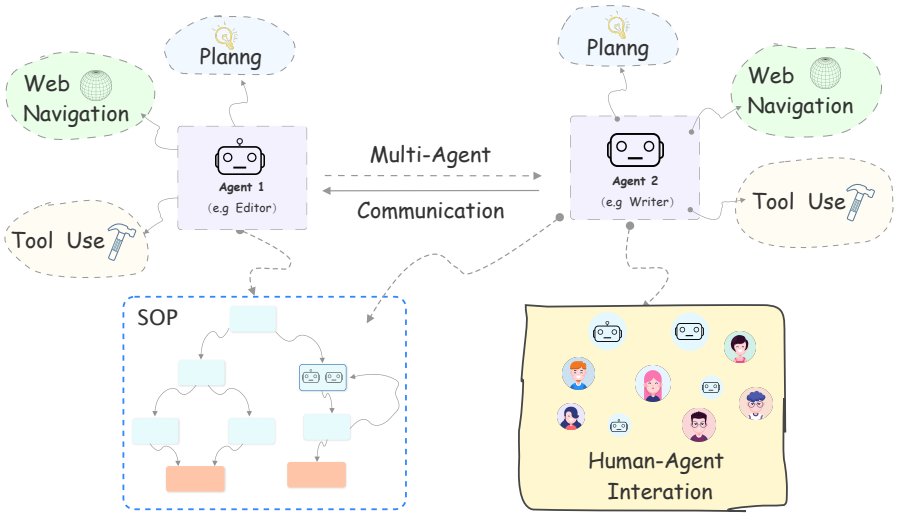

Figure 1: Illustration of the AGENTS framework.

et al., 2023], web navigation [Nakano et al., 2021], tool usage [Schick et al., 2023], and multi-agent communication [Foerster et al., 2016]. In addition, most (if not all) existing language agent frameworks solely depend on a short task description and rely completely on the abilities of LLMs to plan and act. This results in significant randomness and inconsistency across different runs, delivering an unsatisfactory user experience and making it hard to customize and tune language agents.

We believe the aforementioned limitations are important barriers for recent advances in language agents to reach a broader non-specialist audience and impact our society in a positive way. To this end, we release AGENTS, an open-source library and framework for language agents dedicated to supporting LLM-powered language agents. AGENTS's philosophy is to make customizing, tuning, and deploying language agents as simple as possible even for non-specialists while also remaining easily extensible for developers and researchers. In addition, the library also provides the following key features that make it a versatile framework for language agents:

Long-short term memory According to Franklin and Graesser [1996], a key difference between autonomous agents and computer programs (or machine learning models) is that machine learning models only need to respond to a single input/query, while autonomous agents need to interact with environments or other agents over time. Therefore, the ability to maintain long-short term memory is very important for autonomous agents. AGENTS integrates the memory components in [Zhou et al., 2023] and enables language agents to store and retrieve long-term memory with VectorDB and semantic search, and regularly update a short-term working memory with a scratchpad. Users can choose to equip an agent with long-term memory, short-term memory, or both of them by simply filling in a field in the config file.

Tool usage & Web navigation Another important feature for autonomous agents is the ability to use external tools and surf the internet. This is especially important for language agents because they rely on the language interface and thus need to use external tools to interact with environments beyond language communication and navigate the web to gather useful information. Following [Patil et al., 2023], AGENTS supports a few commonly used external APIs and provides an abstract class that enables developers to integrate other tools with ease. We also enable agents to navigate the internet and gather information by defining web search and web navigation as specialized APIs.

Multi-agent communication In addition to single-agent abilities, AGENTS also supports customizing multi-agent systems, which can be helpful for certain applications such as games [Park et al., 2023], social experiments [Li et al., 2023], software development [Qian et al., 2023], etc. One new feature for multi-agent communication in AGENTS is the "dynamic scheduling" feature. Instead of

scheduling the order for the agents to act with hard-coded rules, dynamic scheduling provides an option to define a controller agent that acts as a "moderator" and decides which agent to perform the next action considering their roles and the current history. Dynamic scheduling has the potential to make communication between multiple agents more natural and flexible. Developers can easily customize the controller by specifying its rule in the config file using natural language.

Human-agent interaction One limitation in existing agent frameworks is that while they enable agents, or multi-agents, to automatically solve tasks, it's not easy or even possible for human users to interact with the agents, especially in the multi-agent scenario. AGENTS seamlessly supports human-agent interaction in both single-agent and multi-agent scenarios, making it possible for one or more humans to communicate and interact with language agents.

Controllabilty Existing agent frameworks generally define and control the agents' behavior only using a system prompt and then let the agent plan and act on its own. In contrast, AGENTS provides a novel paradigm to build *controllable agents* via a *symbolic plan*, also referred to as *standard operating procedures* (SOPs). An SOP is a graph of multiple states that defines different situations an agent may encounter while accomplishing a task, and the transition rules between the states. Similar to SOPs in the real world, an SOP in AGENTS is a meticulously documented set of step-by-step instructions that outlines how a particular task or process should be performed by an agent or a group of agents. SOPs can be generated by an LLM and edited by the user when customizing and tuning the agent. After deployment, an agent will behave following specified instructions and guidelines for each state and dynamically adjust its current state according to its interaction with the environment, humans, or other agents. The introduction of the symbolic plan offers the opportunity to provide fine-grained control of an agent's behavior, making agents' behavior more stable/predictable and facilitating tuning/optimizing agents at the same time.

AGENTS is an ongoing effort maintained by researchers and engineers from AIWaves[2]. We look forward to support from community contributors on the project. The library and detailed documentation and tutorials are available on GitHub[3].

2 Related Work

2.1 Autonomous Language Agents

The concept of language agents has become very popular recently and a variety of language agents targeting different tasks have been proposed. For example, Generative Agents [Park et al., 2023] developed language agents to mimic human social behavior, WebAgent [Gur et al., 2023] demonstrated the possibility to build language agents that can complete the tasks on real websites following natural language instructions, Qian et al. [2023] and MetaGPT [Hong et al., 2023] experimented with software development in multi-agent communication settings, and Zhou et al. [2023] built language agents that act as interactive writing assistants.

In addition to language agents that target specific tasks, recent open-source projects such as Auto-GPT [Richards and et al., 2023], BabyAGI [Nakajima, 2023], and SuperAGI [SuperAGI, 2023] are aimed at the goal of building autonomous agents that do whatever users want and attracted massive interest from both developers and non-specialist audiences.

2.2 Language Agents Frameworks

More recently, a few open-source frameworks for language agents have been proposed. For example, Transformers Agents [Wolf et al., 2020] builds language agents that can automatically use tools to solve tasks described in natural language; LangChain [LangChain, 2022] supports end-to-end language agents that can automatically solve tasks specified in natural language; Camel [Li et al., 2023] and AgentVerse [Chen et al., 2023] are platforms tailored for building multi-agent systems; Gentopia [Xu et al., 2023] and XLang[4] are libraries for building tool-augmented agents. We illustrate the key features supported by these platforms and AGENTS in Table 1. We can see that AGENTS is the

[2] `https://www.aiwaves.org/`

[3] `https://github.com/aiwaves-cn/agents`

[4] `https://github.com/xlang-ai/xlang`

Table 1: Comparison of Language Agent Frameworks

Framework	Tool Usage	Long-short Term Memory	Multi-Agent	Human-Agent Interaction	Symbolic Control
Transformers Agents	✓	✗	✗	✗	✗
LangChain	✓	✓	✗	✗	✗
Auto-GPT	✓	✗	✗	✗	✗
Gentopia	✓	✗	✗	✗	✗
XLang	✓	✗	✗	✗	✗
Meta-GPT	✓	✗	✓	✗	✗
Camel	✓	✗	✓	✗	✗
AgentVerse	✓	✗	✓	✗	✗
AGENTS	✓	✓	✓	✓	✓

only framework that supports tool usage, long-short term memory, and multi-agent communication at the same time. AGENTS also offers human-agent interaction and controllability through symbolic plans (SOPs) for the first time.

3 Library Design

Code 1: Exemplar code for initializing and running a (multi) agent system with AGENTS

```
def main()
    # agents is a dict of one or multiple agents.
    agents = Agent.from_config("./config.json")
    sop = SOP.from_config("./config.json")
    environment = Environment.from_config("./config.json")
    run(agents,sop,environment)
```

AGENTS is designed following the philosophy in Franklin and Graesser [1996]: *"an **autonomous agent** is situated in an **environment**"*. Therefore, **agent** and **environment** are two major classes in the AGENTS framework. In addition to these two classes, we also include a class for symbolic plans, named **SOP** (short for Standard Operating Procedure), to make language agents more controllable. These main classes are all initialized from a config file which can be filled in plain text. In sum, a typical script for initializing and running a (multi) agent system with AGENTS is illustrated in Code 1. The config file not only defines these core objects but also factorizes complicated prompts into modularized prompt components. The factorization of prompts significantly reduces the expertise requirements and efforts for users to build (multi) agent systems. Using a single config file to define the agents, plan, and basic environment also facilitates the sharing of language agents (which will be discussed in the Agent Hub section). Each of these three core classes consist of standardized APIs that can be overwritten by experienced developers and researchers. We describe these classes in detail:

Code 2: Exemplar code for the running loop of a (multi) agent system in AGENTS

```
def run(agents,sop,environment):
    while not sop.finished:
        agent,state=sop.step(agents, environment)
        action=agent.step(state,environment)
        environment.update(agent,action)
        # optional, in case of dynamic planning
        # new_states = get_new_states(action)
        # sop.add_states(new_states)
```

3.1 Agent

The *Agent* class abstracts a language agent. Its UML is illustrated in Figure 1. We can see that an agent maintains its long-short term memory and has methods to observe the environment (`agent._observe(environment)`), act according to its current state (`agent._act()`) and update its memory (`agent._update_memory()`). All these methods are wrapped in the `agent.step()` method. This factorization enables developers to customize agents with new functionalities easily.

Unlike existing language agent frameworks that assume an agent must be based on an LLM, we include a "`_is_human`" property to an agent. If it is set to "`True`", the (`agent._act()`) will opt to provide observations and memory information to a human user and wait for the human user to input the action. This design allows flexible human-agent interaction in both single-agent and multi-agent systems by allowing human users to take the role of one or more language agents. It facilitates developers to build various interesting applications such as allowing human users to act as members of a team in debate and collaborate with (agent or human-based) teammates to beat another team, or act as CTO/engineers in a software company and collaborate with others for software development.

3.2 SOP

The SOP class contains a graph of the states of agents. Each state specifies a certain sub-task or sub-goal of all agents when accomplishing the task described by the SOP. States are abstracted into a `State` class. A `State` object contains modularized prompts for the agent to leverage an LLM and various tools or APIs that an agent can use in the state. We abstract everything an agent may use for action in a state into a "`Component`" class. The `Component` class consists of two subclasses corresponding to different parts of the prompt and tools or external APIs, named "`PromptComponent`" and "`ToolComponent`", respectively. `PromptComponent` includes modularized prompts that specify the task/goal, rules/constraints, (step-by-step) demonstrations for in-context learning, and the output format. `ToolComponent` supports more complex usage beyond modularized prompts, including external tools and APIs such as web search, knowledge bases, etc. The results of the tools are either included in the prompt or directly returned and processed afterward, according to the config file.

An SOP object also includes an LLM-based control function that decides the transition between different states and the next agent to act. The state transit function is named `sop._transit()` and the agent routing function is named `sop._route()`. Both of the functions are wrapped in an `sop.next()` function which is used in the main loop.

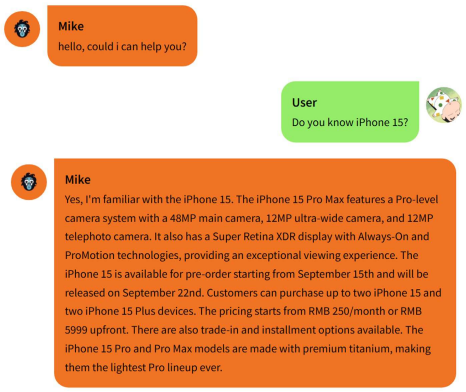

Figure 2: (a) Customer service agent

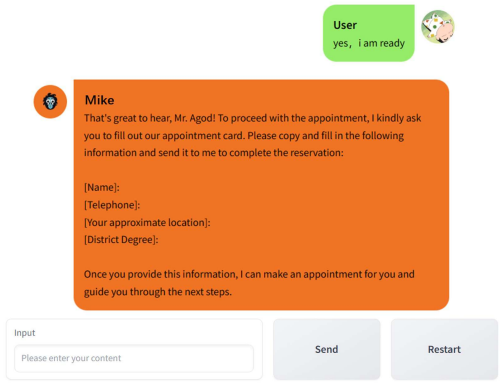

Figure 3: (b) Sales agent

3.3 Environment

The `Environment` class abstracts the environment in which the agents are situated. An `environment` consists of two main functions: `environment._observed()` and `environment.update()`. `environment._observed()` defines how the environment influences the agent's action (i.e., what information should be transferred to the agent upon observation, and `environment.update()` defines how the agent's action impacts the environment.

The execution logic of a (multi) agent system based on AGENTS is very intuitive. As illustrated in Code 2, in each iteration, the SOP first decides the state transition and selects the next agent to act based on the agents and the environment. The agent then takes an action based on its state and the environment. Then the environment updates itself based on the new action. Finally, if a workflow requires dynamically adjusting the plan based on the intermediate execution results, one can parse the output from an action, define a new state and add it into the current SOP.

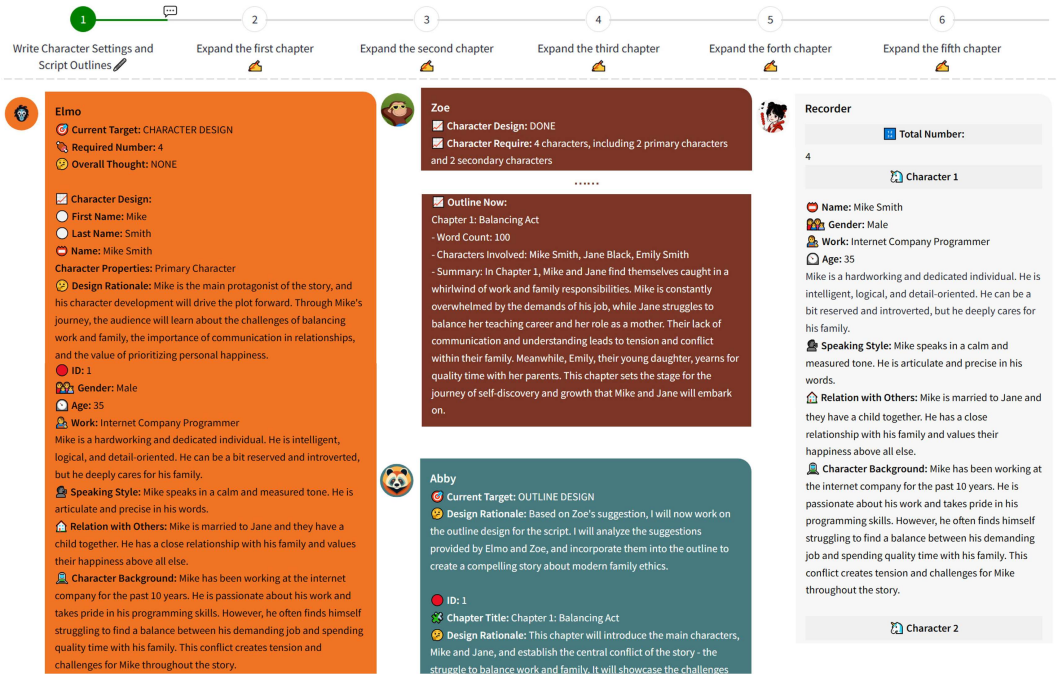

Figure 4: Multi-Agent System: Fiction Studio.

3.4 Implementation Details of Core Features

Long-short Term Memory : AGENTS implements long-short term memories for language agents following Zhou et al. [2023]. Specifically, long-term memories are action histories and are embedded by sentence-transformers [Reimers and Gurevych, 2019], stored in a VectorDB, and queried via semantic search. Short-term memories, or working memories, are in natural language form and updated by an LLM via a carefully tuned prompt.

Tool Usage & Web Navigation : AGENTS supports tool usage and web navigation via `ToolComponents`. For each external tool or API, developer can wrap the API call in the`ToolComponent.func()` method. For complicated tools of which the API call is context-dependent, AGENTS integrates the the "Function-calling" feature of OpenAI's GPT APIs to let LLMs decide how to use the tools. Web navigation is achieved by implementing web search as a specialized tool.

Multi-Agent Communication : Different from most existing frameworks for multi-agent systems that use pre-defined rules (e.g., let each agent act in a sequential order) to control the order for agents' action, AGENTS includes a controller function that dynamically decides which agent will perform the next action using an LLM by considering the previous actions, the environment, and the target of the current states. This makes multi-agent communication more flexible.

Human-Agent Interaction : AGENTS supports human-agent interaction in multi-agent systems by allowing human users to change the "`is_human`" field for a certain agent in the config file to "`True`". In this case, the user can play the role of the agent by himself/herself and input his/her own actions and interact with other language agents in the environment.

3.5 Deployment

Existing open-source frameworks for language agents focus on building proof-of-concept language agents that run either in the terminal or on Gradio [Abid et al., 2019]. In contrast, AGENTS supports

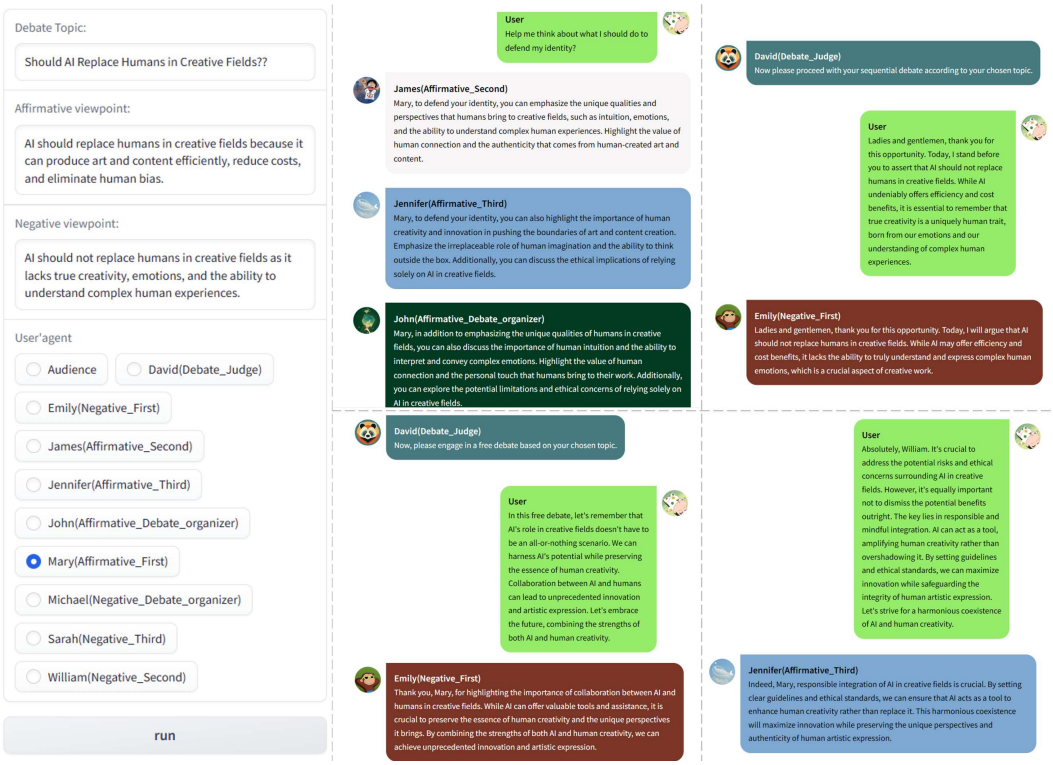

Figure 5: Human-Agent Interaction in a debate.

deploying language agents as APIs with FastAPI[5]. This greatly facilitates developers to integrate language agents in real-world applications.

3.6 The Agent Hub

AGENTS aims to not only facilitate the development, testing, and tuning of a language agents system but also makes the distribution and sharing of language agents easier. To this end, we introduce AGENT HUB, a platform that allows users to share their fine-tuned language agents as well as search/download useful language agents that others share on the platform. In this way, one can easily customize language agents by starting from community agents and slightly modifying them. This greatly reduces the effort of designing, testing, and tuning language agents from scratch.

4 Case Studies

We then present a few case studies on different language agents built with the library, including single-agent systems, multi-agent systems, and systems that require human-agent interaction. All demos are available at `http://www.aiwaves-agents.com/`.

4.1 Single-agent systems

We implement a few single-agent systems with AGENTS including a chit-chat bot, two customer service agents based on knowledge bases and web search engines, a shopping assistant agent, and a sales agent. The agents demonstrate different features of the library and the possibility of building language agents of different use cases using AGENTS. We present a screenshot of the customer service agent and the sales agent in Figure 2 and 3, respectively.

[5]https://fastapi.tiangolo.com/

4.2 Multi-agent systems

We also demonstrate how one can build a multi-agent system consisting of multiple agents interacting with each other in an environment. We select three scenarios including a fiction studio, a debate, and a software company. These scenarios include both cooperative and competitive scenarios, which are two main categories of multi-agent systems. All of the scenarios include multiple subtasks that are controlled through symbolic plans, i.e., SOPs. One can easily observe the language agents' behavior in each subtask and engineer the corresponding prompts to customize and improve the system. We present a system screenshot of the fiction studio system in Figure 4. We also showcase the human-agent interaction feature of the framework in a case study where a human user participate in a debate with language agents in Figure 5.

5 Conclusion

LLMs and language agents powered by them are playing increasingly important roles in both the NLP/AI community and our society in general. AGENTS, is a unified framework and open-source library for language agents. AGENTS aims to facilitate developers to build applications with language agents, researchers to conduct language agents research, and general non-technical audiences to build and customize personalized language agents.

References

Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International workshop on agent theories, architectures, and languages*, pages 21–35. Springer, 1996.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=TG8KACxEON`.

OpenAI. GPT-4 technical report, 2023.

Lilian Weng. Llm-powered autonomous agents. *lilianweng.github.io*, Jun 2023. URL `https://lilianweng.github.io/posts/2023-06-23-agent/`.

Toran Bruce Richards and et al. Auto-gpt: An autonomous gpt-4 experiment, 2023. URL `https://github.com/Significant-Gravitas/Auto-GPT`. [Software].

Yohei Nakajima. Babyagi, 2023. URL `https://github.com/yoheinakajima/babyagi`. [Software].

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2022. URL `https://openreview.net/forum?id=iedYJm92o0a`.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. Recurrentgpt: Interactive generation of (arbitrarily) long text, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021.

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. *CoRR*, abs/2302.04761, 2023.

Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, pages 2137–2145, 2016.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large scale language model society, 2023.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development, 2023.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis, 2023.

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.

SuperAGI. Superagi, 2023. URL `https://github.com/TransformerOptimus/SuperAGI`. [Software].

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://aclanthology.org/2020.emnlp-demos.6`.

LangChain. Langchain repository. `https://github.com/langchain-ai/langchain`, 2022.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents, 2023.

Binfeng Xu, Xukun Liu, Hua Shen, Zeyu Han, Yuhan Li, Murong Yue, Zhiyuan Peng, Yuchen Liu, Ziyu Yao, and Dongkuan Xu. Gentopia: A collaborative platform for tool-augmented llms, 2023.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*, 2019.