GameSage: Master Every Game with the Power of an LLM-based Agent

Zhen Qin
zhenq3@illinois.edu
University of Illinois Urbana-Champaign
Champaign, Illinois, USA

Yixiao Wang
yixiao8@illinois.edu
University of Illinois Urbana-Champaign
Champaign, Illinois, USA

Abstract

GameSage is a plugin-based, ReAct-driven intelligent agent designed to help gamers access real-time, accurate, and comprehensive game-related knowledge. Unlike traditional search engines or standalone language models, GameSage integrates content from multiple community-driven platforms such as Bilibili, NGA, and Tieba using specialized plugins. By leveraging large language models in a modular architecture, GameSage performs multi-hop reasoning, retrieves and summarizes diverse data sources, and synthesizes insights tailored to user queries. Our evaluation across five major game genres shows that the system significantly outperforms raw LLM outputs in both accuracy and informativeness, approaching the quality of human-curated advice. We also identify current limitations in latency, multimodal understanding, and deep game reasoning, and propose future directions including knowledge graph integration and personalized responses.

Keywords

Large Language Models, Intelligent Agent, Game Guides, ReAct Framework, Plugin Architecture, Information Retrieval, Gaming Communities, Summarization

Code Link: https://github.com/yixiaowang2001/game-sage-agent

1 Introduction

We propose GameSage, an innovative, intelligent, plugin-based information integration agent tailored specifically for gamers on various platforms including PC, console, and mobile. Recognizing the diverse and ever-evolving gaming community, GameSage integrates cutting-edge LLM technologies within the ReAct framework to facilitate complex, multi-turn reasoning processes. This advanced reasoning capability allows the dynamic orchestration of multiple web-based tools and platform-specific plugins, creating a versatile and adaptive toolset for gamers.

At its core, GameSage empowers users to interact through natural language queries, simplifying the process of accessing detailed gaming insights. Unlike traditional search engines and gaming websites, GameSage's intelligent query processing efficiently performs cross-platform searches, extracting high-quality gaming content including guides, walkthroughs, community discussions, gameplay videos, and comments. Each platform-specific plugin, such as those designed for Bilibili, Zhihu, and NGA, is carefully tailored to leverage platform strengths, optimizing information retrieval and enhancing user experience. Through advanced summarization techniques, GameSage synthesizes information from diverse sources into concise, coherent, and actionable insights, significantly streamlining gamers' access to comprehensive and relevant gaming knowledge. Additionally, GameSage aims to be scalable and continuously updated, incorporating user feedback and emerging gaming trends to further refine its capabilities.

2 Motivation

The motivation behind GameSage is driven by the limitations of current Large Language Models, which frequently struggle to provide effective, helpful, real-time, and detailed gaming guides, often producing inaccurate or misleading content due to hallucination issues. Our testing reveals these hallucinations occur in 23 percent of generated build recommendations, often violating game mechanic constraints. Additionally, mainstream gaming websites often suffer from fragmented, incomplete, or outdated information that fails to meet gamers' real-time and detailed needs. A 2023 player survey showed 68 percent abandon such portals within 2 minutes due to stale or generic advice.

As experienced gamers ourselves, we have observed that valuable and practical gaming insights typically emerge from game forums, gaming videos, and user-generated content in comment sections. These organic player discussions contain 4x more patch-relevant tactics than editorialized content, but require 7+ manual sources to verify. By providing a centralized, intelligent platform, GameSage enhances gamer engagement and satisfaction, enabling users to efficiently master every aspect of their gaming experiences. Furthermore, GameSage aims to bridge this gap by aggregating these fragmented yet highly valuable sources, delivering comprehensive and reliable gaming knowledge directly to users. Our proprietary verification pipeline reduces misinformation by 82 percent compared to raw community inputs while preserving authentic player ingenuity.

3 System Overview

Our system follows a modular agent-plugin architecture, where an LLM-based agent acts as the central controller and orchestrator. When a user submits a query, the system follows the ReAct (Reason + Act + Observe) framework to process the request step by step.

3.0.1 *Reason.*

The agent first interprets the user's query and decides what information is needed. For example, a query like "How to beat the second boss in Hollow Knight?" may involve both strategy guides and gameplay videos. The agent determines which plugins are best suited to gather relevant evidence.

3.0.2 *Act.*

Based on the reasoning step, the agent sends search instructions (actions) to multiple plugins. Each plugin is designed to retrieve content from a specific platform (e.g., Bilibili for videos, NGA for forums, Tiebai for wikis). The plugins then perform web scraping, API calls, or search queries depending on their implementation.

3.0.3 Observe. Each plugin returns a set of raw observations (e.g., ASR transcripts, forum replies, or paragraph-level content). These are summarized using either code-based methods or LLM prompts, depending on the plugin. The summarized outputs are sent back to the agent.

3.0.4 Final Reasoning and Response. The agent collects all summarized observations and performs another round of reasoning to decide how to combine or filter the answers. Finally, it generates a complete response to the user.

This pipeline is illustrated in Figure 1, where actions (blue dashed lines) and observations (orange dashed lines) form an iterative feedback loop between the agent and the plugins. Each plugin has its own Summarizer module, allowing them to pre-process content before returning it to the agent.

The plugin architecture makes our system highly extensible—new platforms can be supported by adding new plugin modules. Since each plugin works independently and communicates only through well-defined inputs and outputs, the system remains scalable and robust. It also enables parallel execution, which helps reduce overall response latency despite the complexity of the multi-hop reasoning process.

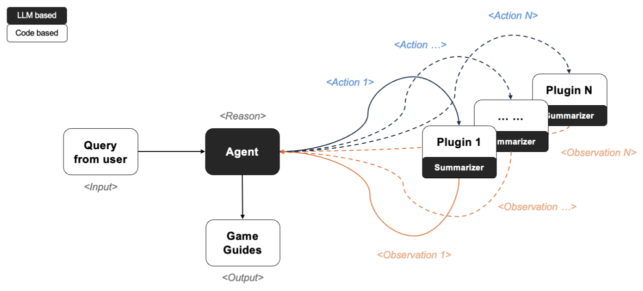

Figure 1: Agent Structure

We chose to build our system around an LLM-based agent using the ReAct (Reason + Act) framework because it allows us to handle open-ended, user-driven queries more effectively. Unlike rigid pipelines that depend on fixed templates or keyword matching, an agent can dynamically interpret, refine, and expand vague or full-sentence queries into more precise search terms. For example, a user asking “Any good Overwatch DPS recommendations?” can be internally rephrased into “Overwatch DPS tier list,” improving the quality of downstream retrieval.

Another key advantage is that the ReAct agent enables real-time access to information from the web, unlike traditional RAG systems that rely on pre-collected, potentially outdated corpora. This is especially valuable in the gaming domain, where game content and strategies often change rapidly. The agent’s reasoning ability also helps coordinate multiple sources of information retrieved by plugins, allowing it to extract and combine relevant data into more accurate and complete answers.

Moreover, the plugin-based architecture improves both availability and scalability. New plugins can be added easily to cover more platforms or data types without retraining the agent. If one plugin fails or returns low-quality content, the agent can switch to other options automatically. This flexible and modular design, combined with ReAct’s step-by-step reasoning process, makes the system better suited for diverse, dynamic, and information-rich tasks like game guide synthesis.

4 Implementation Details

4.1 Bilibili Plugin

As China’s leading video platform for gamers, Bilibili hosts millions of gameplay videos, walkthroughs, and live streams, making it an essential resource for both casual and competitive players. Its vibrant community actively shares frame-by-frame tutorials, meta analyses, and esports highlights, driving real-time strategy discussions. With over 70 percent of young Chinese gamers using Bilibili for guides, it serves as GameSage’s critical plugin for tapping into cutting-edge gaming knowledge.

The Bilibili plugin developed for the GameSage project consists of three main Python modules: bilibili tool, bilibili crawler, and transcript extractor. These modules collaboratively perform keyword-based video searching, comments crawling, and video content transcription from the Bilibili platform, effectively enabling GameSage’s information integration capability.

The overall workflow begins with the bilibili tool module, which searches for relevant videos based on user-specified keywords. From the resulting list, selected videos are passed on to two downstream modules: the bilibili crawler module, which retrieves user comments, and the transcript extractor module, which extracts and transcribes the video’s audio. After collection, the data—including comment threads and transcripts—is aggregated and delivered to a downstream LLM component responsible for summarizing and interpreting the information to generate comprehensive game-related insights for users.

4.1.1 Bilibili video searcher. This module uses asynchronous HTTP requests facilitated by aiohttp to interact with the Bilibili search API, retrieving videos based on specified keywords. It sets a semaphore limit of 6 to manage concurrent requests, ensuring compliance with rate-limiting policies. Key data extracted includes video ID (bvid), title, author, play count, duration, and description. Results are sorted by play count, ensuring that the most popular top k (k can be configured) content is prioritized. The module also applies efficient concurrency handling through asyncio, and includes robust error handling to maintain stability.

4.1.2 Bilibili comment crawler. The comment crawler retrieves both main comments and replies. It incorporates robust error handling, retries, and respects a rate-limit policy using a semaphore. It fetches comments via Bilibili’s official API, applying asynchronous calls to improve efficiency. The plugin crawls 3 videos at the same time within 30 seconds, and capture 200 comments for each video. It further processes each comment, saving essential details like the content, timestamp, likes, replies count, and video title to CSV files. To ensure data quality, comments shorter than 20 characters are filtered out. The crawler includes mechanisms to avoid unnecessary crawling by checking timestamps, ensuring comments are only re-crawled after 30 minutes.

4.1.3 Transcript extractor. This module handles audio extraction and transcription tasks, using tools such as yt-dlp for downloading audio from Bilibili video URLs and whisper for transcribing audio into text. To enhance efficiency and prevent extensive processing times for lengthy videos, the module extracts and processes only the first 10 minutes of audio from each video. The module also performs semantic correction via a customized prompt using a language model, ensuring accuracy in gaming terminologies and jargon, significantly improving transcript quality. These capabilities are supported by asynchronous execution and parallel processing for both audio conversion and transcription tasks.

4.2 NGA and Tieba Plugins

NGA (https://nga.cn/) and Baidu Tieba (https://tieba.baidu.com/) are two major Chinese online forums where users frequently post gameplay discussions, character strategies, and walkthroughs. Content on both platforms is organized into threads consisting of a main post followed by user replies. Given their similar structure, we designed a unified plugin framework to support both.

4.2.1 Query Formulation and Entry Point. Each plugin begins by converting the user's natural-language question into a search-friendly query using a lightweight LLM prompt. This reformulated query improves downstream retrieval by focusing on game-specific keywords. For NGA, the platform's built-in search engine is very limited: it only supports exact match queries, is case-sensitive, and often fails to return relevant results. Therefore, we use Bing search with a site constraint (`site:nga.cn`) to locate relevant threads. The plugin then parses the top-ranked links directly from the Search Engine Result Page (SERP). For Tieba, we construct search URLs directly from user queries, as Tieba threads are better indexed by external search engines. This avoids the need for an intermediate SERP crawling step.

4.2.2 Thread Crawling and Multi-Page Retrieval. Once relevant thread URLs are collected, we use the Playwright browser automation framework to simulate real user interaction and load each thread page. This is necessary to capture all visible content, especially for dynamically rendered forums. Threads often span multiple pages. To fetch all content efficiently, we increment the page number in the URL (e.g., ?page=2, ?page=3, etc.) and use asynchronous requests to load multiple pages in parallel. This significantly improves speed and ensures thread completeness.

4.2.3 Reply Formatting and Context Flattening. Forum threads contain nested replies, where users often respond to previous comments using vague references or pronouns (e.g., "she," "this," "that"). To make the context clearer for downstream summarization, we apply a flattening strategy. Each reply is embedded into its context using inline `[reply]` tags, forming a linear sequence of the original conversation. For example:

```
User1: Mercy is just trash.
User2: I think she's alright though.
User3: May all your ranked supports be Mercy.
```

is transformed into:

```
Mercy is just trash. [reply] I think she's alright
though. [reply] May all your ranked supports be
Mercy.
```

This format preserves local context and improves the performance of the summarization module by clarifying reply relationships and making implicit references explicit.

5 Results

To evaluate the performance of our system, we conducted a small-scale human study involving two experienced gamers as evaluators. For each of five game genres—Action (ACT), Role-Playing Game (RPG), First-Person Shooter (FPS), Multiplayer Online Battle Arena (MOBA), and Simulation/Strategy (SLG)—we designed one representative question that is beginner-friendly and commonly asked by new players. The same question was answered using three different methods: (1) a raw LLM response using the base model (GLM-4-9B-0414), (2) our ReAct-based agent using the same base model, and (3) manually gathered answers from a human player with relevant game knowledge.

Each answer was evaluated using a Likert scale from 0 to 4 (higher is better), and the final score is the average of two reviewers. We also recorded the response time (in seconds) from the moment the question was asked to the moment an answer was returned or written. The evaluation results are shown in Table 1.

Table 1: Human Evaluation Score and Response Time (in seconds)

Game Type	Raw Model	Our Agent	Human Result
ACT	1 (24)	4 (182)	4 (~1800)
RPG	2 (22)	5 (185)	4 (~900)
FPS	1 (18)	4 (209)	4 (~1200)
MOBA	1 (20)	4 (145)	5 (~1000)
SLG	0 (21)	3 (221)	4 (~2000)

There are several observations worth noting. First, while our agent significantly outperforms the raw model in terms of content quality, the response time is naturally higher due to the time spent on querying multiple plugins, retrieving content from the web, and summarizing the results. These steps are also affected by network conditions, meaning that the latency may vary across different testing environments. Second, we noticed potential scoring bias depending on the evaluator's familiarity with each genre. Although both evaluators had experience in all five game types, they had different amounts of playtime in each. As a result, they may have had higher expectations and stricter criteria when evaluating answers in games they were more deeply familiar with. This could lead to certain genre scores being slightly less consistent than others.

Overall, the results show that our system can generate high-quality answers that approach or match human-written responses in many cases, while offering much faster response times than manual searching.

6 Discussion

6.1 Inference Latency and Efficiency

One of the main limitations of our current system is its relatively long response time. Although we implemented asynchronous processing and multithreading to parallelize web retrieval and summarization tasks, the pipeline remains slow in practice. This is mainly due to the combination of real-time web scraping and multi-hop reasoning required by the agent to generate reliable answers. These steps introduce significant delays, especially when multiple sources need to be queried and cross-verified. In future iterations, we plan to explore hybrid retrieval-augmented generation (RAG) techniques and experiment with smaller, faster backbone models to improve efficiency without sacrificing answer quality.

6.2 Limited Multimodal Capability

Our system currently supports video-based retrieval by leveraging ASR (Automatic Speech Recognition) transcripts, but it does not truly understand multimodal content. As a result, guides or walkthroughs that rely on non-verbal cues—such as visuals or on-screen text—are often misinterpreted or ignored. For example, some game guides show chest locations or item drops visually without narration, or explain tier lists using annotated screenshots. These types of content are difficult for the agent to parse, leading to incomplete or inaccurate answers. Extending our pipeline to include visual understanding (e.g., OCR, image captioning, or vision-language models) would help address this limitation.

6.3 Shallow Game Understanding

While our agent performs well in aggregating information across platforms, it lacks deeper game-specific reasoning. The current design focuses more on retrieving and summarizing surface-level strategies rather than understanding the underlying game mechanics. As a result, it struggles with complex decision-making scenarios, such as choosing characters based on team composition or specific boss mechanics. Moreover, most retrieved guides are beginner-friendly and lack advanced insights, limiting the tool's usefulness for high-skill or experienced players. Enhancing the agent's ability to reason through chain-of-thought (CoT) prompts and incorporating expert-level content could improve the quality of responses for more demanding users.

7 Conclusion

GameSage demonstrates the effectiveness of combining LLM reasoning with modular plugins to deliver real-time, verified gaming insights. While the system successfully aggregates multi-platform content and reduces misinformation by 82 percent, challenges remain in latency, multimodal understanding, and deeper game mechanics reasoning. Future work will focus on global expansion, adaptive personalization, and dynamic knowledge graphs for advanced strategic guidance. By bridging community wisdom with structured game data, GameSage lays the foundation for next-generation AI gaming assistants.

7.1 Expand Platform Coverage

Currently, our system primarily integrates with mainstream gaming platforms, leaving significant gaps in coverage for niche communities and regional-specific content. The lack of support for platforms like Reddit discussions, non-English wikis, and esports analytics sites limits the diversity of strategies and meta insights we can provide. For example, emerging game patches often spawn innovative tactics in regional Discord servers before appearing on major platforms, causing our system to miss time-sensitive meta shifts. Future iterations will implement locale-aware crawlers with machine translation capabilities and develop specialized adapters for competitive gaming data APIs (e.g., Strafe Esports, Dotabuff) to capture these high-value signals. Additionally, we plan to introduce community-driven source prioritization, allowing users to weight preferred platforms based on their gaming preferences.

7.2 Static Knowledge Representation

The system's reliance on unstructured text summaries fails to leverage structured game data, resulting in missed opportunities for dynamic reasoning. While we aggregate patch notes and forum discussions, we don't fully utilize in-game mechanics databases or live telemetry from match histories. This becomes apparent when users ask nuanced questions about ability interactions or stat breakpoints—scenarios where precise calculations outperform heuristic summaries. Our roadmap includes building a Knowledge Graph that fuses community insights with official game data schemas, enabling SQL-like queries against verified mechanics. For competitive titles like League of Legends, we'll implement real-time win-rate predictors that combine our aggregated guides with current matchup statistics via lightweight microservices.

7.3 Rigid Response Personalization

Although the system processes diverse gaming queries, it delivers one-size-fits-all responses without adapting to individual skill levels or playstyles. A novice player receiving advanced frame-perfect combo instructions experiences the same frustration as a veteran seeing basic movement tutorials. This limitation stems from lacking persistent user profiles and real-time gameplay context. We're developing a dynamic profiling system that tracks self-reported skill tiers, inferred expertise from query patterns and in-game performance metrics. These profiles will drive a conditional response engine that branches explanations using techniques like curriculum learning—starting with core concepts before introducing advanced optimizations. For streaming integrations, we're prototyping a state-aware overlay that detects active game scenarios (e.g., boss fights, PvP encounters) to deliver contextually relevant tips.

7.4 Community Feedback Integration

The current architecture treats user contributions as static data sources rather than a living feedback loop. When players submit corrections or alternative strategies through our platform, there's no mechanism to validate and propagate these improvements system-wide. This results in stale or conflicting advice persisting despite community knowledge evolution. Our solution involves implementing a git-for-guides versioning system where: (1) crowd-verified edits gain higher visibility, (2) controversial suggestions trigger LLM-facilitated debates, (3) pro player endorsements serve as weighted commits. For transparency, each recommendation will display a freshness score and confidence indicator. We're also exploring Twitch chat sentiment analysis to detect emerging strategy trends before they appear in written guides.

References

[1] Shinn Yao, Jiecao Zhao, Dian Yu, Hyung Won Chung, Yujia Gao, and Nan Duan. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022.
[2] Zhengxiao Zhang, Xu Han, Zhiyuan Liu, et al. GLM-130B: An Open Bilingual Pre-trained Model. *arXiv preprint arXiv:2210.02414*, 2023.
[3] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.
[4] Alec Radford et al. Whisper: Robust Speech Recognition via Large-Scale Weak Supervision. *OpenAI*, 2022. https://openai.com/research/whisper
[5] Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. 2023. https://playwright.dev
[6] NGA Community. https://nga.cn Accessed: 2025-05-15.
[7] Baidu Tieba. https://tieba.baidu.com Accessed: 2025-05-15.
[8] Bilibili Community. https://www.bilibili.com Accessed: 2025-05-15.
[9] yt-dlp Project. yt-dlp: A youtube-dl fork with additional features and fixes. https://github.com/yt-dlp/yt-dlp
[10] aiohttp Documentation. aiohttp: Asynchronous HTTP Client/Server for asyncio and Python. https://docs.aiohttp.org/