© Volume 2, Issue 10, October 2026 | JATIR
e-ISSN: 2839-5786, p-ISSN: 3139-8464

Modular AI Workspaces: A Review of Large Language Models, Persistent Memory, Retrieval, Tool Use and Local Deployment, with the ASTRA AI Case Study

[1]Sayali Gaikwad.[2]Ajim Shaikh,[3]Nitin Kote,[4]Sanika Kamble
[1]Professor, Department of Information Technology Parvatibai Genba Moze College of Engineering, Wagholi, Pune, India
[2,3,4]Department of Information Technology Parvatibai Genba Moze College of Engineering, Wagholi, Pune, India

***Abstract*—Large language models (LLMs) are now general-purpose components for conversation, coding, research and document work, yet they are usually delivered through separate applications that fragment workflows, duplicate context and disconnect AI assistance from the artifacts users work on. This paper reviews the technologies behind an emerging class of systems termed AI workspaces, in which conversational AI shares persistent state, authentication and data with productivity modules. Five enabling areas are examined: persistent memory, retrieval-augmented generation, agentic tool use, multimodal AI, and efficient local inference. A reference architecture is synthesized, security and privacy threats are analyzed, and an evaluation framework for workspace-level systems is proposed. The ASTRA AI project, a twelve-module unified workspace with a Next.js and Spring Boot implementation and a local Qwen3 1.7B prototype, serves as a case study that separates planned design from implemented functionality. The review identifies open problems in memory scaling, permission-aware retrieval, small-model reliability, user isolation and workflow-level benchmarks.**

***Index Terms*—AI workspace, conversational memory, large language models, local LLM, persistent memory, retrieval-augmented generation, tool use, Spring Boot.**

I. INTRODUCTION

Large language models (LLMs) have progressed from research prototypes to general-purpose components for conversation, programming, research support and content generation [1], [2], [3]. Surveys now document their model families, training and adaptation methods, efficiency

techniques and application domains [3], [4]. Yet the value users obtain depends less on the raw model than on the software that surrounds it: interfaces, storage, retrieval, authentication, tool integration and orchestration. Reviews of industrial adoption observe that many deployments remain confined to isolated, well-bounded tasks rather than being embedded in end-to-end workflows [4].

In daily practice a user may rely on one product for conversational assistance, another for coding, another for research, and further applications for documents, calendars, tasks and team communication. Each keeps its own account, data store and context. The consequences are repeated transfer of information, loss of context between sessions, multiple subscriptions, weak collaboration and scattered data [5]. This motivates AI workspaces: platforms in which conversational AI and productivity modules share authentication, data and context rather than operating as separate destinations.

This paper reviews the technologies required to build such a workspace and discusses how they fit together. The ASTRA AI project [5], [6], a unified AI-first workspace combining AI chat, research, coding, documents, projects, tasks, calendar, collaboration and analytics, is used throughout as a concrete case study. The main contributions are:

1) a thematic review of five enabling technology areas: persistent memory, retrieval-augmented generation (RAG), agentic tool use, multimodal AI, and efficient local inference;
2) a reference architecture for modular AI workspaces synthesized from the literature;
3) a case study mapping the ASTRA AI design and prototype onto that architecture, separating planned design from implemented functionality;
4) an analysis of security and privacy risks, and an evaluation framework for workspace-level systems; and
5) a consolidated list of research gaps and future directions.

The rest of the paper is organized as follows. Section II describes the review method. Section III gives background on LLMs. Section IV defines the fragmentation problem. Section V reviews the enabling technologies. Section VI presents the reference architecture, and Section VII the ASTRA AI case study. Sections VIII and IX cover security and evaluation, Section X gives a comparative analysis, and Sections XI to XIII present research gaps, future work and conclusions.

II. REVIEW METHODOLOGY

A thematic review approach was adopted. Sources were grouped into nine categories: LLMs and their applications; LLM-based agents; memory mechanisms; RAG; tool use and orchestration; efficient inference; local and on-device deployment; security and privacy; and evaluation. Peer-reviewed IEEE publications and widely cited surveys were preferred. Because the field changes quickly, influential arXiv preprints are included where they provide detailed taxonomies or recent coverage, and are identified as such in the reference list. The aim is not to summaries individual papers but to identify relationships between research areas and determine what a complete AI

workspace requires. Statements about ASTRA AI are limited to what its project documentation states; no performance measurements are claimed for it.

III. BACKGROUND: LARGE LANGUAGE MODELS

A. Capabilities

The Transformer architecture [1] enabled parallel training on very large corpora and underpins modern LLMs. Models such as GPT-3 demonstrated in-context learning, where tasks are specified through prompts without weight updates [2]. Capabilities relevant to workspaces include question answering, summarization, code generation, information extraction, classification, reasoning and multi-turn dialogue. Open-weight families such as Qwen3 provide models from sub-billion to tens of billions of parameters, making local experimentation practical [7].

B. Limitations

A bare LLM has five limitations that motivate surrounding architecture. (1) Hallucination: plausible but unsupported output. (2) Knowledge limits: parametric knowledge becomes outdated and excludes private data. (3) Context limits: only a finite window can be processed, and models use long contexts unevenly, with information in the middle of a long prompt often used less reliably [8]. (4) Computational cost: large models demand substantial memory and compute. (5) Statelessness: a basic model keeps no application-level user memory between independent requests.

IV. FRAGMENTATION IN AI-ASSISTED WORK

The ASTRA AI documentation identifies three core aspects of the problem: too many disconnected tools; time and focus lost through constant platform switching; and the absence of a single source of truth for AI-assisted work [5], [6]. Legacy single-purpose AI tools are further described as having no memory across sessions, limited integration with code, documents, calendars and tasks, generic responses without project context, subscription and tool fatigue, limited collaboration, and fragmented data with inconsistent access models. These symptoms point to a design question: how can AI services obtain the right context from workspace modules while respecting user and team boundaries?

V. ENABLING TECHNOLOGIES

A. Persistent Memory

Memory carries information across turns and sessions. A systematic survey treats memory as essential for long-horizon and personalized agent behavior and examines how it is written, stored and read [9]; more recent work organizes mechanisms into context compression, retrieval-based stores, reflective memory, hierarchical context and learned memory management [10].

Representative systems include Generative Agents, which add a memory stream with retrieval and reflection [11], and MemGPT, which manages memory tiers by analogy with operating-system virtual memory [12]. Table I compares the main approaches.

TABLE I. Memory approaches for LLM systems

Approach	Strengths	Weaknesses
Context-window history	Simple; no extra components	Token cost and latency grow; long-context degradation [8]
Persisted conversation store	Survives restarts; supports history and titles	Limited by what is replayed to the model
Summarization	Controls prompt length	Information loss; compounding errors
Retrieval-based memory	Scales to long histories	Needs embeddings and an index; variable relevance
Structured / agentic memory	Long-term personalization	Complex; forgetting, contradiction and privacy are open issues

A complete memory system must decide what to store, how to represent it, when to retrieve it, how to handle conflicting memories and when to forget. Recent surveys identify contradiction handling, latency, privacy governance and selective management as continuing challenges [10], [9]. Many practical systems, including the ASTRA AI prototype, adopt a lightweight application-level scheme: conversations and messages are stored in a database, and a window of history together with user-profile fields is supplied to the model on each request. This is easy to implement and test, but it should be distinguished from autonomous long-term memory architectures.

B. Retrieval-Augmented Generation

RAG couples a parametric model with a non-parametric knowledge source so that relevant passages are retrieved and supplied to the generator [13]. It addresses hallucination, outdated knowledge and the inability of parametric models to access dynamic or private information [14]. Surveys distinguish naive RAG (retrieve and prepend), advanced RAG (query transformation, re-ranking, improved context construction) and modular RAG, in which retrieval, transformation and generation are replaceable modules [14]; the modular view suits modular workspaces. Self-RAG further trains models to decide when to retrieve and to critique retrieved evidence [15].

RAG does not eliminate hallucination. Failures include irrelevant or missed documents, poor chunking, incorrect ranking, insufficient context and conflicting sources, so retrieval quality should be evaluated separately from generation quality. In a workspace, the documents, projects and notes a user already owns form the natural knowledge base; retrieval must, however, enforce the same access controls as the underlying data. A system should be called RAG-enabled only once retrieval and grounding are implemented and evaluated.

C. Agents and Tool Use

Agentic approaches extend one-shot prompting with planning, reflection, tool use and multi-agent collaboration. A conventional interaction (user, LLM, response) becomes user, agent, LLM, then memory, tools and environment, then action and result. Surveys describe agents through profile, memory, planning and action modules [16], [17]. React interleaves reasoning traces with actions [18], and Tool former shows that models can learn when to call external APIs [19]. Production-oriented work stresses coordinated tool selection, retrieval, memory and execution [20], and agent surveys additionally examine cooperation paradigms, security and privacy [21].

In a workspace the request "analyses my project documents and create a task list" illustrates why these matters: it needs document access, retrieval, LLM reasoning, task creation through the task module and database persistence. A standalone LLM cannot complete it. Reliability matters most for small local models, whose instruction following and structured output are weaker; tool schemas should therefore be constrained, outputs validated and consequential actions confirmed by the user.

D. Multimodal AI

Multimodal models process text, images, audio, video and documents. For workspaces this enables pipelines such as document, vision model, extracted information, LLM, task; or screenshot, vision model, code analysis, code assistant. Multimodality broadens a chat assistant into a workspace assistant, at the price of greater compute demand and new attack surfaces such as instructions hidden in images or documents.

E. Efficient and Local Inference

Large model size, attention complexity and autoregressive decoding are the main sources of inference inefficiency; optimization methods are grouped into data-level, model-level and system-level techniques [22], or into quantization, pruning, distillation, efficient architectures, compilation and hardware-aware optimization [23]. On-device surveys focus on running models under tight memory and compute limits [24]. Representative techniques are post-training quantization [25], [26] and serving optimizations such as paged KV-cache management [27], while local model servers such as Ollama expose downloaded models through a simple API [28].

Local inference can reduce external data transmission, increase deployment control, remove dependence on remote API availability and allow offline use. It also brings hardware requirements, model-loading cost, memory limits, lower throughput on weak hardware and the need to secure local infrastructure. "Local" must therefore not be equated with "secure" or "private": logging, storage, network exposure and access control remain the application's responsibility. Because longer conversation history lengthens prompts, local deployment and memory design are coupled through latency and RAM use. A hybrid policy that routes simple tasks to a local model and complex ones to a larger model is also plausible.

VI. REFERENCE ARCHITECTURE FOR AI WORKSPACES

Synthesizing the reviewed literature, a modular AI workspace can be organized in five layers: a presentation layer (web or mobile client); an API layer with authentication, validation and routing; a service and orchestration layer containing memory, retrieval and tool components; a persistence layer (relational, document, vector and file stores); and an inference layer (hosted API or local model server). Fig. 1 shows the resulting structure. Separating the model from the rest of the system lets the model be replaced and allows new modules to reuse authentication and data services.

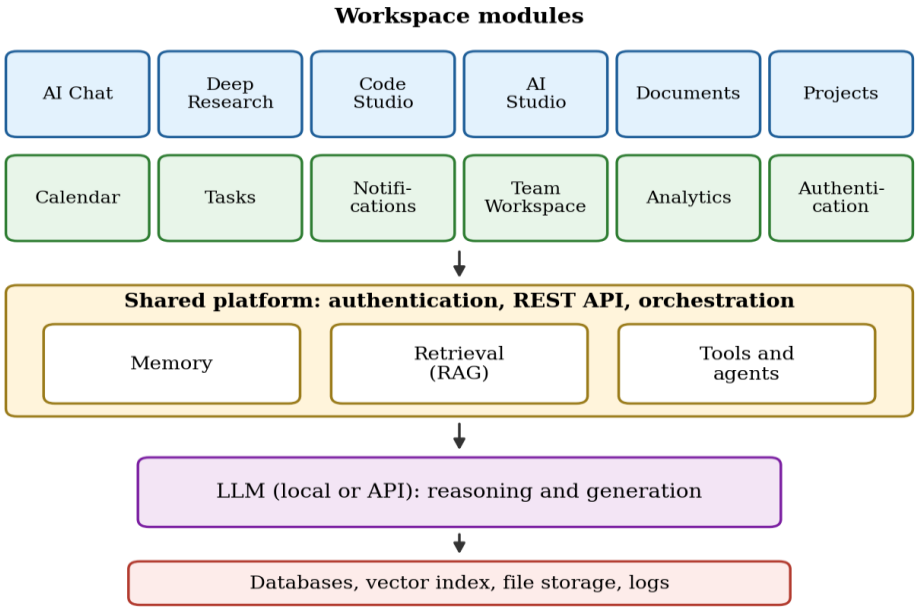

Fig. 1. Reference architecture of a modular AI workspace.

VII. CASE STUDY: ASTRA AI

A. Design Overview

ASTRA AI is proposed as a unified, SaaS-ready AI-first workspace for students, developers, startups, enterprises and research organizations [5], [6]. Its twelve modules are AI Chat, Deep Research, Code Studio, AI Studio, Documents, Projects, Calendar, Tasks, Notifications, Team Workspace, Analytics and Authentication. AI Chat provides conversational assistance; Deep Research supports research summarization; Code Studio provides AI-assisted coding; AI Studio supports prompt experimentation; Documents acts as a knowledge repository; Projects organize related tasks, documents and AI interactions; Team Workspace and Notifications provide collaboration and real-time updates; and Authentication forms the security boundary in front of AI services and the database.

The documented design specifies a Next.js frontend with Axios, a Java/Spring Boot REST layer, PostgreSQL or MongoDB for data, LLM APIs or inference services, JWT/OAuth 2.0

authentication, Redis and S3-compatible storage for caching and files, WebSocket-based real-time updates, Nginx as reverse proxy, and Docker with a GitHub-based CI/CD pipeline on a cloud platform with elastic scaling (Fig. 2).

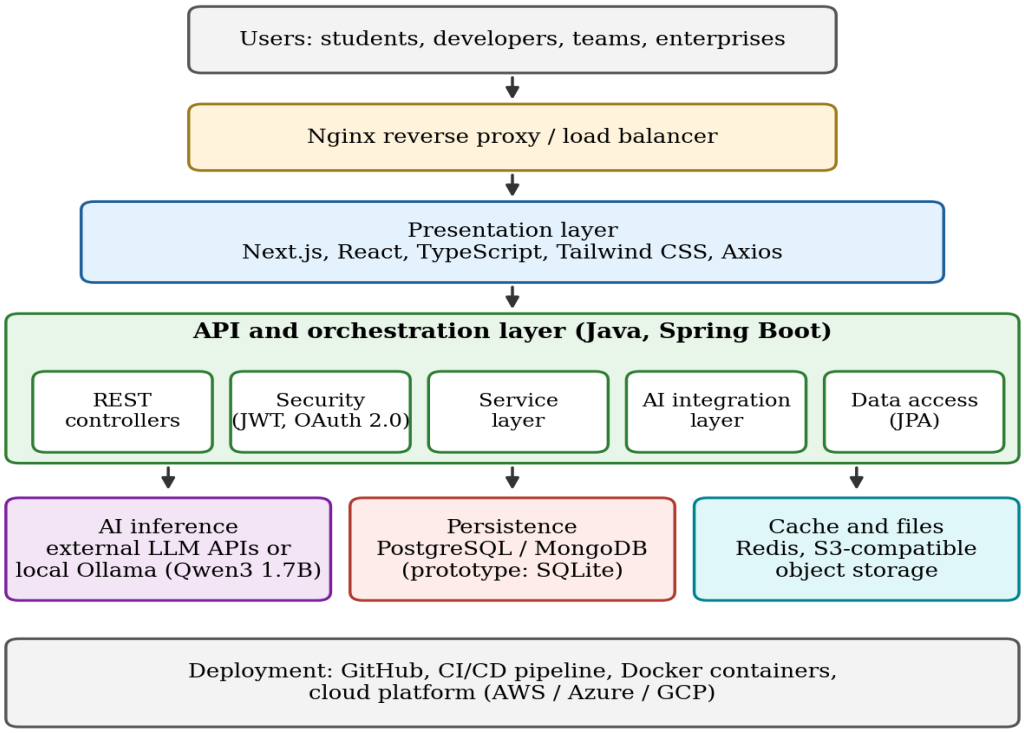

Fig. 2. ASTRA AI layered system architecture.

B. Request-Processing Workflow

The documented workflow has eleven steps: (1) the user logs in; (2) selects a module; (3) the Next.js frontend captures the prompt or task action; (4) Axios sends the request to the REST API; (5) the API authenticates and validates it; (6) it is routed to the appropriate AI service; (7) the service retrieves or stores contextual information such as chat history, project data and documents; (8) the result returns through the API; (9) the frontend renders it in the relevant module; (10) updates are synchronized through Team Workspace and Notifications if collaboration is enabled; and (11) the interaction is logged for context retention. Authentication is applied before any request reaches AI services or the database.

C. Data Model

The entity-relationship design contains User, Project, Task, Document, Chat, Calendar Event, Notification, Team, Team Member, AI Model and Audit Log entities, with mapping tables Project Document and Task Document. A user creates many projects; a project has many tasks, documents, chats and calendar events; users belong to teams; users receive notifications; and an AI model is used by many chats. The Audit_Log entity records actions per user and entity, supporting the auditability discussed in Section VIII.

D. Current Prototype

The present prototype implements the conversational core on a lighter stack: a Next.js frontend, a Java Spring Boot backend with Spring Data JPA, SQLite persistence, and a locally hosted Qwen3 1.7B model served through Ollama. Its data model links User, Conversation and ChatMessage. On each request the backend stores the user message, prepares conversation history, adds user context (username and display name), calls the model, stores the reply and returns it. A conversation title is derived from the first message. This realises a persisted conversation store with application-supplied user context, the second row of Table I. It is not yet a RAG system or an agentic system, and the remaining modules are planned design rather than validated functionality.

TABLE II. ASTRA AI: documented design vs. current prototype

Aspect	Documented design	Current prototype
Database	PostgreSQL / MongoDB, Redis, S3	SQLite
AI service	LLM APIs / inference services	Ollama with Qwen3 1.7B
Modules	Twelve modules	Conversational core
Memory	Chat history, project and document context	Persisted conversations plus user context
Deployment	Nginx, Docker, CI/CD, cloud	Local development

VIII. SECURITY AND PRIVACY

A workspace concentrates conversations, documents, source code, project data and identity, so a flaw has larger consequences than in a single-purpose chatbot. Table III lists the principal threats.

TABLE III. Principal threats to an AI workspace

Threat	Description
Prompt injection	Malicious instructions in user input or retrieved content alter model behavior; indirect injection through documents or web content is a concern for RAG and agents [29].
Unauthorised tool use	An agent executes an action without sufficient authorization or confirmation.
Cross-user leakage	Weak access control or manipulated identifiers expose one user's conversations or documents to another.
Sensitive data exposure	Private data appears in prompts, logs, databases or external APIs; models can also reproduce memorized training data [30].
Memory poisoning	False or malicious content written to stored memory later influences answers.
Model-level risk	Hallucination, bias and unreliable reasoning affect system behavior.

The OWASP list of LLM-application risks provides a useful checklist [31]. Defence should be layered: authentication, authorization, data-access control, input validation, tool-permission checks, the LLM, output validation, and audit logging. Four principles follow. First, enforce authorization in the backend on every conversation and artifact request. Second, scope memory and retrieval queries by user and team identifiers. Third, treat retrieved and user-supplied text as untrusted when building prompts. Fourth, require confirmation for state-changing actions. User isolation should be tested rather than assumed, for example by attempting to read one user's conversation with another user's credentials.

IX. EVALUATION FRAMEWORK

Benchmarks usually score isolated model abilities, whereas a workspace should be judged on whole workflows, such as question, retrieval, reasoning, tool execution, database update and user response. Table IV proposes evaluation dimensions. Latency should be reported over repeated runs (at least 30) with mean, median and standard deviation, by prompt and conversation length, together with hardware, model version and quantization level. No performance figures should be claimed for a prototype until such experiments have been completed.

TABLE IV. Proposed evaluation dimensions

Dimension	Example metrics	Notes
Latency	Time to first token; total time; tokens/s	Vary history length
Context retention	Correct recall of facts across turns (%)	Test position effects [8]
Persistence	Retrieval success after restart	Repeat many cases
User isolation	Rejection rate of unauthorized access	Include forged IDs
Resource use	CPU, RAM, VRAM, load time, energy	Local deployment [23]
Retrieval quality	Recall@k, faithfulness, citation accuracy	If RAG is used [14]
Task quality	Rubric or human rating	Compare larger baseline
Usability	Task completion time, satisfaction	Versus multi-tool use

X. COMPARATIVE ANALYSIS

Table V contrasts traditional AI applications with the emerging approaches reviewed here. The trend is from isolated language generation toward interconnected systems combining models, memory, retrieval, tools and applications. ASTRA AI does not introduce a new LLM architecture; its relevance lies in integrating several of these concepts within one modular workspace.

TABLE V. Traditional vs. emerging approaches

Area	Traditional	Emerging	Workspace role
LLM	Standalone generator	Reasoning engine	Core
Memory	Session-only	Persistent	Continuity

Area	Traditional	Emerging	Workspace role
Knowledge	Model parameters	RAG	Private data
Tools	Manual operation	Tool-using agents	Automation
Deployment	Cloud only	Local, cloud, edge	Flexibility
Interaction	Text	Multimodal	Input range
Productivity	Separate apps	Unified workspace	Workflow
Security	Application level	AI, tool and data	Safe action

XI. RESEARCH GAPS

1. Fragmented capabilities: memory, RAG, agents and efficient inference are usually studied separately, but a workspace needs them to operate together.
2. Memory and retrieval integration: the right division between personal memory and external knowledge retrieval remains open [10].
3. Local inference with persistent memory: the interaction of local models, long-term memory and latency needs empirical study [23].
4. Grounding on workspace artifacts: few studies examine permission-aware RAG over a user's own mixed documents, tasks and projects.
5. Small-model reliability: evidence is limited on how sub-2B models handle memory use, grounding and tool calling.
6. Workflow-level evaluation: there are no standard benchmarks for integrated workspaces or for end-to-end productivity effects.
7. Security of integrated systems: systematic testing of isolation, injection, memory poisoning and agent permissions in multi-module platforms is scarce [21].
8. Collaboration: shared team memory raises unresolved questions of ownership, consent and visibility.

XII. FUTURE DIRECTIONS

- Permission-aware RAG over documents, projects and notes, with source citation.
- Hybrid memory combining recent context, structured user profiles, rolling summaries and vector retrieval, with adaptive retrieval that decides when to retrieve.
- Multi-agent workspaces with specialized agents for coding, research, project management and scheduling, coordinated by an orchestration layer.
- Local-cloud hybrid inference routing tasks by complexity, privacy sensitivity and cost.
- Multimodal workspaces integrating documents, images, audio and video.
- Scalable deployment with server databases, caching, containers and concurrent inference serving.

- Dedicated AI security mechanisms for memory, RAG, tools and multi-user environments, and user studies measuring productivity effects.

XIII. CONCLUSION

Capable LLMs, including compact models that run locally, are now widely available, and the main engineering challenge has shifted to integrating them into coherent systems. Persistent memory, retrieval grounding, tool orchestration, efficient inference and robust security are complementary enabling technologies, each involving trade-offs among simplicity, capability, cost and risk. A modular AI workspace is best understood as an ecosystem of LLM, memory, retrieval, tools, data, authentication and interface rather than an LLM alone. ASTRA AI illustrates this direction through a unified twelve-module design and a working local conversational prototype. Realising the benefits of such workspaces will require permission-aware grounding, scalable memory, careful security engineering and, above all, rigorous empirical evaluation of complete workflows.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2017.
[2] T. B. Brown et al., "Language models are few-shot learners," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.
[3] W. X. Zhao et al., "A survey of large language models," arXiv:2303.18223, 2023.
[4] Y. Zhang et al., "A comprehensive overview and analysis of large language models: Trends and challenges," IEEE Access, vol. 13, pp. 95851-95875, 2025, doi: 10.1109/ACCESS.2025.3573955.
[5] ASTRA AI, "Modern AI workspace platform," project synopsis and technical documentation, Parvatibai Genba Moze College of Engineering, Wagholi, Pune, India, 2026.
[6] ASTRA AI, "Modern AI workspace platform," project presentation, 2026.
[7] A. Yang et al., "Qwen3 technical report," arXiv:2505.09388, 2025.
[8] N. F. Liu et al., "Lost in the middle: How language models use long contexts," Trans. Assoc. Comput. Linguist., 2024 (arXiv:2307.03172).
[9] Z. Zhang et al., "A survey on the memory mechanism of large language model based agents," arXiv:2404.13501, 2024.
[10] P. Du, "Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers," arXiv:2603.07670, 2026.
[11] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in Proc. ACM Symp. User Interface Softw. Technol. (UIST), 2023.
[12] C. Packer et al., "MemGPT: Towards LLMs as operating systems," arXiv:2310.08560, 2023.

[13] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2020.

[14] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," arXiv:2312.10997, 2023.

[15] A. Asai et al., "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," arXiv:2310.11511, 2023.

[16] L. Wang et al., "A survey on large language model based autonomous agents," arXiv:2308.11432, 2023.

[17] Z. Xi et al., "The rise and potential of large language model based agents: A survey," arXiv:2309.07864, 2023.

[18] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," arXiv:2210.03629, 2022.

[19] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," arXiv:2302.04761, 2023.

[20] "Tool and agent selection for large language model agents in production: A survey," in Proc. IEEE Conf. Artif. Intell. (CAI), 2026, doi: 10.1109/CAI68641.2026.11536377.

[21] "Large model-based agents: State-of-the-art, cooperation paradigms, security and privacy, and future trends," IEEE Commun. Surveys Tuts., vol. 28, pp. 1906-1949, 2025, doi: 10.1109/COMST.2025.3576176.

[22] Z. Zhou et al., "A survey on efficient inference for large language models," arXiv:2404.14294, 2024.

[23] "Make large language models efficient: A review," IEEE Access, vol. 13, pp. 154466-154490, 2025, doi: 10.1109/ACCESS.2025.3605110.

[24] "A survey on optimization techniques for on-device large language model inference," in Proc. Int. Conf. Ubiquitous Future Netw. (ICUFN), 2026, doi: 10.1109/ICUFN69619.2026.11628507.

[25] E. Frantar et al., "GPTQ: Accurate post-training quantization for generative pre-trained transformers," arXiv:2210.17323, 2022.

[26] J. Lin et al., "AWQ: Activation-aware weight quantization for LLM compression and acceleration," arXiv:2306.00978, 2023.

[27] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in Proc. ACM Symp. Oper. Syst. Princ. (SOSP), 2023.

[28] Ollama, "Ollama: Run large language models locally," software documentation. [Online]. Available: https://ollama.com

[29] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," arXiv:2302.12173, 2023.

[30] N. Carlini et al., "Extracting training data from large language models," in Proc. USENIX Security Symp., 2021.

[31] OWASP Foundation, "OWASP top 10 for large language model applications." [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/