

Context Engineering 2.0: The Context of Context Engineering

Qishuo Hua^{1,3} Lyumanshan Ye^{1,3} Dayuan Fu^{2,3} Yang Xiao^{2,3} Xiaojie Cai^{1,3}
 Yunze Wu^{1,2,3} Jifan Lin^{1,3} Junfei Wang³ Pengfei Liu^{1,2,3,†}

¹SJTU ²SII ³GAIR

 Github  SII Context

“A person is the sum of their contexts.”

— Authors

Abstract

Karl Marx once wrote that “the human essence is the ensemble of social relations” (Marx, 1845), suggesting that individuals are not isolated entities, but are fundamentally shaped by their interactions with other entities — within which contexts play a constitutive and essential role. With the advent of computers and artificial intelligence, these contexts are no longer limited to purely human-human interactions: human-machine interactions are included as well. Then a central question emerges: **How can machines better understand our situations and purposes?** To address this challenge, researchers have recently “developed” the concept of *context engineering*. Although it is often regarded as a recent innovation of the agent era, in fact, we argue that related practices can be traced back to over **20 years ago**. Since the early 1990s, it has evolved through distinct historical phases, each shaped by its intelligence level of machines: from early human-computer interaction (HCI) frameworks built around primitive computers, to today’s human-agent interaction (HAI) paradigms driven by intelligent agents, and potentially to human-level or even superhuman intelligence in the future. In this paper, we discuss the context of context engineering, provide a systematic definition, outline our perspective on its historical and conceptual landscape, and examine key design considerations for its practice. By addressing these questions, we aim to offer a conceptual foundation for context engineering and sketch its promising future. This paper serves as a stepping stone for a broader community effort toward systematic context engineering in AI systems.

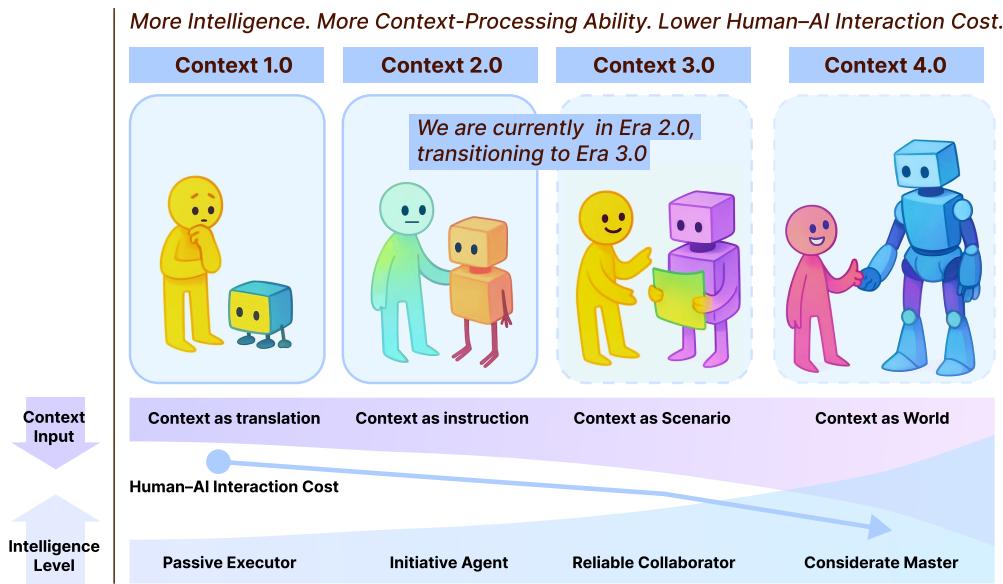


Figure 1: The Overview of context engineering 1.0 to context engineering 4.0, illustrating that more intelligence leads to greater context-processing ability and lower human-AI interaction cost.

^{0†} Corresponding author.

Contents

1	Introduction	3
2	Theoretical Framework	4
2.1	Formal Definition	4
2.2	Stage Characterization	6
3	Historical Evolution	6
3.1	Over 20 Years Ago: Era 1.0	7
3.1.1	Technological Landscape	7
3.1.2	Theoretical Foundations	7
3.1.3	Core Practices	7
3.2	20 Years Later: Era 2.0	8
4	Context Collection and Storage	9
4.1	Typical Strategies in Era 1.0 and 2.0	9
4.2	Human-Level Context Ecosystem	10
5	Context Management	10
5.1	Textual Context Processing	10
5.2	Multi-Modal Context Processing	11
5.3	Context Organization	12
5.3.1	Layered Architecture of Memory	12
5.3.2	Context Isolation	12
5.4	Context Abstraction	13
6	Context Usage	14
6.1	Intra-System Context Sharing	14
6.2	Cross-System Context Sharing	15
6.3	Context Selection for Understanding	15
6.4	Proactive User Need Inference	17
6.5	Lifelong Context Preservation and Update	17
6.6	Emerging Engineering Practices	18
7	Applications	19
7.1	CLI	19
7.2	Deep Research	19
7.3	Brain-Computer Interfaces	20
8	Challenges and Future Directions	20
9	Conclusion	21

1 Introduction

In recent years, the rapid rise of large language models (LLM) and intelligent agents has drawn increasing attention to how context influences model behavior. Studies have shown that the content placed within the context window can significantly affect model performance (Liu et al., 2021). At the same time, there is growing demand for systems capable of multi-step reasoning and operating over long time horizons (Yao et al., 2023). These trends make one question central: how can we enable machines to better understand and act upon human intent through effective context mechanisms, especially in long-horizon tasks?

To address this challenge, researchers have recently focused on *context engineering*: the practice of designing, organizing, and managing contextual information so that machines can act in ways that align with human intentions (Mei et al., 2025). Recent years have witnessed extensive implementations of context engineering in LLMs and agents, including prompt engineering (Liu et al., 2021; Reynolds and McDonell, 2021; Wei et al., 2022), retrieval-augmented generation (RAG) (Lewis et al., 2020; Izacard and Grave, 2022), tool calling (Yao et al., 2022; Schick et al., 2023), and long-term memory mechanisms (Wu et al., 2022; Dai et al., 2019). These techniques expand a machine’s capacity to assimilate high-entropy context and have materially influenced the design of interactive systems.

Despite these advances, the field is often misunderstood. Context engineering is commonly perceived as a recent development, and “context” is often narrowly defined as dialogue history, system prompts, or agent-centric environmental input. In fact, context can be more broadly defined, and context engineering has been practiced **for more than 20 years**. Early research in ubiquitous computing, context-aware systems, and human–computer interaction established foundational principles and methods that remain relevant today (Reeves, 2012; Baldauf et al., 2007a; Preece et al., 1994). Recognizing this history is essential to understand both the current state of the field and its future potential.

We argue that the development of context engineering should be viewed through a broader historical perspective rather than being confined to recent technical practices. By tracing its evolution over the past two decades, we can gain a deeper understanding of its underlying principles and recognize how different approaches to dealing with context have shaped the progress of intelligent systems. This view enables AI research to build upon its historical trajectory, establishing a robust and future-oriented foundation.

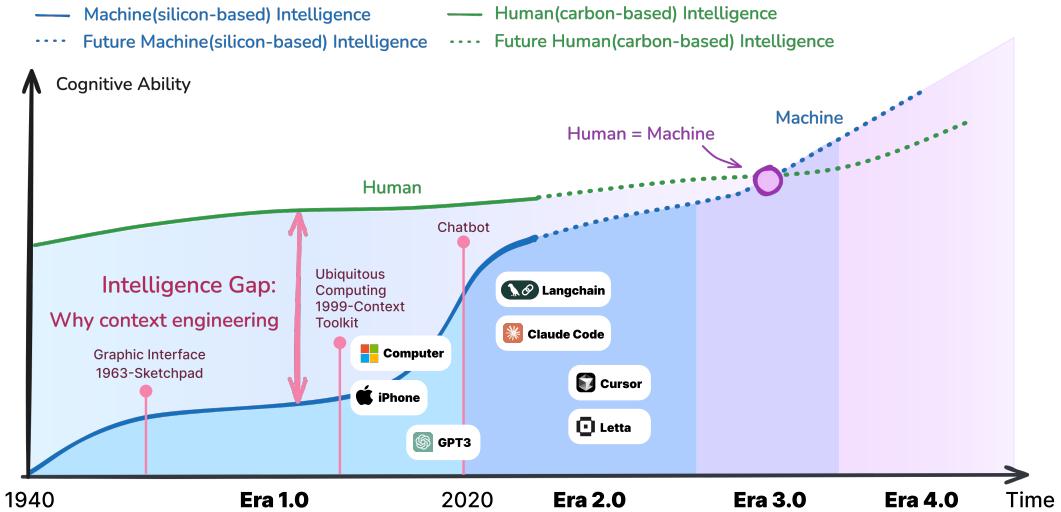


Figure 2: Trajectories of carbon-based and silicon-based cognitive abilities over time. The gap illustrates the motivation for context engineering.

From this broader perspective, context engineering can be viewed as a process of *entropy reduction*. Machines, unlike humans, cannot effectively “fill in the gaps” during communication. When people interact, they rely on the listener’s ability to actively reduce information entropy — the capacity to infer missing context through shared knowledge, emotional cues, and situational awareness (Kapteijns and Hintz, 2021). Machines, at least at present, lack this ability. As a result, we must “preprocess” contexts for them, compressing the original information into forms they can understand. This represents the core “effort” in context engineering: the effort you need to invest in transforming high-entropy contexts and intentions into low-entropy representations that machines can understand. As illustrated in Figure 2, context engineering has always existed to bridge the cognitive gap between human (carbon-based) and machine (silicon-based) intelligence. While carbon-based intelligence develops relatively

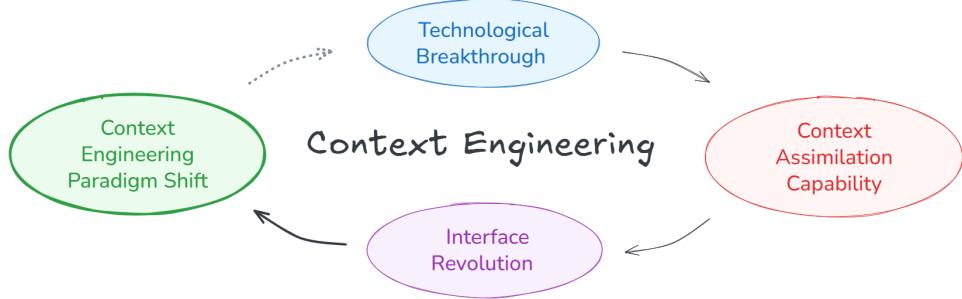


Figure 3: Evolutionary process in context engineering

slowly, silicon-based intelligence iterates at a much faster pace. Therefore, the key driver behind paradigm shifts lies in the rapid advancement of machine intelligence. The more intelligent machines are, the more natural context engineering becomes, and the lower the cost of human–machine interaction.

Each qualitative leap in machine intelligence triggers a fundamental revolution in human-machine interfaces. As illustrated in Figure 3, **technological breakthroughs lead to a surge in context assimilation capability, which drives interface revolutions and ultimately results in paradigm shifts in context engineering**. These shifts are not gradual improvements, but a series of paradigm changes that fundamentally reshape how humans and machines communicate. Based on this recurring pattern, we can conceptualize the evolution of context engineering as a progression through **four distinct stages**: Context Engineering 1.0 — primitive computing with structured, low-entropy inputs (Dey, 2001a); 2.0 — intelligent agents capable of interpreting natural language and handling ambiguity (Jennings et al., 1998); 3.0 — human-level intelligence, enabling nuanced communication and seamless collaboration (Morris et al., 2023); and 4.0 — superhuman intelligence, where machines can proactively construct context and reveal needs that humans have not explicitly articulated. Each stage entails qualitatively different design trade-offs and highlights the shifting roles of humans and machines.

The core contributions of this paper are as follows. First, We situate context engineering within a broader historical perspective, tracing its origins before modern intelligent agents. Second, we present a systematic and broadly defined theoretical framework, including the essence of context engineering from an entropy reduction perspective and a four-stage evolutionary model reflecting its technical progress. Finally, we propose general design considerations by comparing typical practices, providing guidance for future intelligent system development.

In the remainder of this paper, we will first present our definition of context engineering and outline the theoretical framework that grounds our analysis. We will then trace the historical evolution of the field, with particular attention to the distinctive characteristics of the 1.0 and 2.0 eras. Building on this foundation, we discuss design considerations structured around three core dimensions: context collection, context management, and context usage. We analyze current practices, identify unique challenges, and explore emerging approaches that may shape future advancements.

2 Theoretical Framework

To better understand context, we begin by formalizing the fundamental constructs that define context in computational systems. Building on Dey’s foundational definition of context (Dey, 2001b), we express these constructs mathematically, providing a precise representation of context. Based on this formalization, we define context engineering and examine how it has evolved across different stages of machine intelligence.

2.1 Formal Definition

Definition 1 (Entity and Characterization). *Let \mathcal{E} be the space of all entities (users, applications, objects, environments, etc.) and \mathcal{F} be the space of all possible characterization information. Denote by $\mathcal{P}(\mathcal{F})$ the power set of \mathcal{F} . For any entity $e \in \mathcal{E}$, define the situational characterization function:*

$$\text{Char} : \mathcal{E} \rightarrow \mathcal{P}(\mathcal{F}) \quad (1)$$

where $\text{Char}(e)$ returns the set of information characterizing entity e .

For example, when a user types “Search related documentation for me” in the Gemini CLI¹, the set of relevant entities can include the user, the Gemini CLI application, the terminal environment, external tools, memory modules, and backend model services. Here, $\text{Char}(e)$ can describe the user (e.g., the input prompt), the application

¹<https://github.com/google-gemini/gemini-cli>

(e.g., system instructions or configuration), the environment (e.g., current working directory), external tools (e.g., available plugins or search tools), short-term or long-term memory modules (e.g., session history or stored knowledge), and the model service (e.g., supported capabilities or response format).

Definition 2 (Interaction). *An interaction refers to any observable engagement between a user and an application, encompassing both explicit actions (e.g., clicks, commands) and implicit behaviors (e.g., attention patterns, environmental adjustments) that may influence or be influenced by the computational system.*

In the Gemini CLI example, the explicit interaction is the user's command input, while implicit aspects can include terminal state, previously retrieved context, usage of memory modules, or tool invocation status.

Definition 3 (Context). *For a given user-application interaction, Context is defined as:*

$$C = \bigcup_{e \in \mathcal{E}_{\text{rel}}} \text{Char}(e) \quad (2)$$

where $\mathcal{E}_{\text{rel}} \subseteq \mathcal{E}$ is the set of entities considered relevant to the interaction.

This captures context as “any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application” (Dey, 2001b). In the same example, \mathcal{E}_{rel} can include the user, the Gemini CLI application, the terminal environment, external tools, memory modules, and backend model services. The general context C is then the aggregation of these characterizations.

Definition 4 (Context engineering). *Context engineering is the systematic process of designing and optimizing context collection, storage, management, and usage to enhance machine understanding and their task performance. Formally, it can be defined as:*

$$\text{CE} : (C, \mathcal{T}) \rightarrow f_{\text{context}} \quad (3)$$

where C denotes the raw contextual information as defined in Equation (2), \mathcal{T} represents the target task or application domain, and f_{context} is the resulting context processing function that transforms and optimizes context representations for improved task performance. f_{context} encompasses a flexible composition of context processing operations:

$$f_{\text{context}}(C) = \mathcal{F}(\phi_1, \phi_2, \dots, \phi_n)(C) \quad (4)$$

where \mathcal{F} represents a composition function that combines various context engineering operations ϕ_i .

The composition \mathcal{F} can involve parallel processing, iterative refinement, conditional branching, or any combination of operations tailored to the specific application requirements. In practice, the operation set $\{\phi_i\}$ in context engineering 2.0 may include: (1) collecting relevant contextual information through sensors or other channels, (2) storing and managing it efficiently, (3) representing it in a consistent and interoperable format, (4) handling multimodal inputs from text, audio, or vision, (5) integrating and reusing past context (“self-baking”), (6) selecting the most relevant contextual elements, (7) sharing context across agents or systems, and (8) adapting context dynamically based on feedback or learned patterns.

The Scope of Context Engineering This definition deliberately avoids limiting context engineering to any specific technology or era. Whether the receiving entity is a 1990s primitive computer with a graphical interface or a 2025 agent, the fundamental challenge remains the same: how to make contexts and intentions accurately understood. The specific techniques and formats used in context engineering evolve with technology, but the core challenge of bridging the gap between human intentions and machine understanding remains constant.

Why This Definition Matters? This broader definition serves several important purposes. First, it connects current prompt engineering practices with the rich history of human-computer interface design, allowing us to learn from decades of accumulated knowledge. Second, it provides a theoretical framework that can explain why certain context designs work across different technologies and eras. Third, it offers a foundation for predicting how context engineering will evolve as machine understanding capabilities continue to advance. By understanding context engineering as a fundamental aspect of human-machine communication rather than a narrow technical practice, we can better appreciate both its historical development and its future trajectory. This perspective reveals context engineering not as a recent invention, but as an evolving discipline that will continue to adapt as the nature of human-machine interaction itself transforms.

2.2 Stage Characterization

To better understand context engineering, we must expand our view beyond the current moment and recognize it as a fundamental aspect of human-machine communication that has been continuously evolving for decades; this ongoing evolution can be characterized through four stages, each aligned with major advances in machine intelligence. Specifically, machine intelligence can be considered to progress through the following stages: (i) Primitive Computation, (ii) Agent-Centric Intelligence, (iii) Human-Level Intelligence, and (iv) Superhuman Intelligence.

Era 1.0: Primitive Computation (1990s–2020) In this stage, machines had only a very limited ability to interpret contexts. They could process structured inputs and recognize simple environmental cues, but lacked a deeper understanding of meaning or intent. Human–machine interactions relied on rigid, predefined formats, such as selecting from menus or using simple sensor data as input. Although this era went beyond binary-level commands, all contexts still had to be explicitly prepared and translated into formats the machine could directly process (Shneiderman, 1987).

Era 2.0: Agent-Centric Intelligence (2020–Present) The emergence of LLM, exemplified by the release of GPT-3 in 2020 (Floridi and Chiratti, 2020; Brown et al., 2020), marks a turning point for context engineering and agent-centric intelligence. Machines in this stage exhibit moderate intelligence, characterized by the ability to comprehend natural language inputs and infer some implicit intentions. Human–machine collaboration becomes increasingly viable, as users are able to express their needs conversationally and systems can interpret much of the underlying meaning. The context is no longer limited to explicitly defined signals; it can encompass ambiguity and incomplete information. Agents actively reason over contextual gaps, using advanced language understanding and in-context learning to provide more adaptive and responsive interactions (Bommasani et al., 2021).

Era 3.0: Human-Level Intelligence (Future) With anticipated breakthroughs, intelligent systems are expected to approach human-level reasoning and understanding (Goertzel and Pennachin, 2021). In this stage, context engineering transcends the current pattern, enabling agents to sense contexts and assimilate high-entropy information like humans. The scope of interpretable context expands significantly, such as social cues, emotional states, and richer environmental dynamics. Such advances enable truly natural human–machine collaboration, with AI acting as knowledgeable and effective peers.

Era 4.0: Superhuman Intelligence (Speculative) As intelligent systems surpass human capabilities, they begin to possess a “god’s eye view”, understanding human intentions more deeply than humans themselves. At this stage, the traditional subject–object relationship is inverted: instead of machines passively adapting to human-defined contexts, they actively construct new contexts for humans, uncover hidden needs, and guide human thinking. Signs of this transformation are already emerging, for example, in Go, professional players are learning novel, superhuman strategies from AI. In this way, machines become sources of insight and inspiration, fundamentally redefining the nature of human-machine collaboration (Silver et al., 2016).

3 Historical Evolution

Aspect	Context Engineering 1.0	Context Engineering 2.0
Time Period	1990s–2020	2020–Present
Technical Background	Ubiquitous computing, context-aware systems, HCI	Large language models, agents, prompt engineering
Typical Systems	Context Toolkit, Cooltown, ContextPhone	ChatGPT, LangChain, AutoGPT, Letta
Context Modalities	Location, identity, activity, time, environment, device state	Token sequences, retrieved documents, tool APIs, user history
Core Mechanisms	Sensor fusion, rule triggers	Prompting, RAG, CoT, memory agents
Context Tolerance	Relatively Low	Relatively High
Human-likeness	Relatively Low	Relatively High

Table 1: Comparison between context engineering 1.0 and 2.0 across representative dimensions

To better understand how context engineering has evolved, we compare its core characteristics across two major eras. The following sections examine the foundations of context engineering in its early development (Era 1.0) and the significant advancements that define the current landscape (Era 2.0).

3.1 Over 20 Years Ago: Era 1.0

We define context engineering as the practice of enabling effective understanding and communication between humans and machines. In the context engineering 1.0 era, humans are responsible for providing context, while primitive computers are the recipients of information. This era primarily spans the 1990s to 2020. In context engineering 1.0, humans struggle to adapt to machines: designers served as “*intention translators*”, converting complex human intentions into structured, machine-readable formats. This stands in contrast to the modern era of AI with semantic understanding capabilities, where machines can autonomously interpret unstructured, high-entropy information.

3.1.1 Technological Landscape

To understand context engineering 1.0, we must first consider the technological landscape of the 1980s and 1990s. This period marked a transition from command-line interfaces (CLI) to graphical user interfaces (GUI), making computers more accessible to the general public. However, this shift did not eliminate the cognitive load on users. Instead, it redefined how humans interacted with machines, requiring precise adaptations to the limitations of the technologies of the time. In 1991, Mark Weiser introduced the concept of *Ubiquitous Computing* (Weiser, 1991), envisioning a seamless integration of computing into everyday environments. This vision suggested that devices could provide services without requiring active user input. Building on this idea, *Context-Aware Computing* emerged as a framework to explore whether systems could sense user states, environments, and tasks to adapt their behavior dynamically (Schilit and Theimer, 1994; Abowd et al., 1999a). This led to foundational questions: *What exactly is “context”? How can it be defined, processed, and made usable by machines?*

Despite these conceptual advances, the technological limitations of the time were significant. Machines could execute only pre-defined program logic and lacked the ability to understand natural language semantics, reason about problems, or handle errors effectively. This created a significant gap between human thinking and machine processing capabilities. To bridge this gap, **context engineering** provided strategies for making human intentions actionable by machines. Designers had to decompose complex goals into simple, structured components that machines could process. Clear interaction pathways needed to be designed to ensure machines could follow pre-determined logic. Additionally, feedback mechanisms were implemented to allow adjustments based on user input or environmental changes. These strategies were essential to overcoming the limitations of machines and enabling them to act on human intent effectively.

3.1.2 Theoretical Foundations

The early 2000s saw the emergence of robust theoretical frameworks for context engineering. Among them, Anind K. Dey’s 2001 definition of *context* stood out as a cornerstone (Dey, 2001a):

“*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves.*”

This definition emphasized the multidimensional nature of contexts. Dey’s work laid the foundation for systematically capturing and utilizing context to improve system adaptability and user experience. The theoretical work of this era was broad and deep (Baldauf et al., 2007b), emphasizing a holistic understanding of context. This contrasts with today’s narrower focus, which often isolates specific aspects of context, such as chat history. The narrowing of this focus represents a step backward. Revisiting the foundational theories of context engineering 1.0 can help address this regression and inspire more comprehensive approaches to context-aware systems.

3.1.3 Core Practices

In the era of context engineering 1.0, a key innovation was the shift from traditional input devices, such as keyboards and mice, to a distributed, sensor-centric paradigm (Schilit and Theimer, 1994; Abowd and Mynatt, 2000). This transition reflected an emerging need to continuously capture richer contextual signals from both users and their surrounding environments. Building on this vision, Anind Dey introduced a general framework for context-aware systems that provided the conceptual and architectural foundation for this generation (Dey, 2001a). The framework was instantiated through the *Context Toolkit*, which defined a modular and reusable framework to support the acquisition, interpretation, and delivery of context (Salber et al., 1999). It was organized around five core abstractions: *Context Widgets*, *Interpreters*, *Aggregators*, *Services*, and *Discoverers*. Widgets encapsulated sensors and exposed standardized interfaces; Interpreters derived higher-level meaning from raw contextual data; Aggregators integrated multiple context sources; Services offered application-level access to contextual functionality; and Discoverers enabled dynamic registration and discovery of context components (Salber et al., 1999). This architectural separation of concerns established a practical foundation for scalable and adaptive context-aware systems, marking a decisive step in the formalization of early context engineering practices.

In sum, context engineering 1.0 laid the foundational thinking and architectural paradigms for building responsive systems. It brought together ideas from ubiquitous computing and HCI, established key abstractions through Anind Dey's formalization (Salber et al., 1999), and enabled practical implementations through modular, sensor-driven designs. Although limited in expressivity and scalability, this phase provided the essential groundwork for the more advanced designs that followed in context engineering 2.0 era.

3.2 20 Years Later: Era 2.0

The evolution from context engineering 1.0 to 2.0 represents a significant leap in machine intelligence, moving from primitive computing to intelligent agents (Ye et al., 2025a). Compared to era 1.0, which was dominated by rule-based reasoning and structured sensor input, agents introduce significant advancements throughout the context pipeline. These improvements span from how context is acquired to how raw signals are tolerated, interpreted, and ultimately used for intelligent behavior. In particular, the following shifts characterize the evolution:

Acquisition of Context: Advanced Sensors In Era 2.0, sensors remain central to context acquisition. Advances in sensing technology, such as smartphones, wearables, and ambient devices, have greatly expanded the diversity and coverage of available contexts. Context engineering 2.0 emphasizes not only a broader diversity of sensors, but also the ability to extract diverse contextual signals from each individual sensor.

Category	Device/Collector	Collected Modalities	Example Input
Personal Computing	Smartphone	Text, Image, Audio, Location, Touch	Messages, Photos, Voice
	Computer (Laptop/PC)	Text, Image, Keystroke, Cursor	Mouse movement, Typing
	Smartwatch	Heart rate, Motion, Audio	Pulse, Steps
Immersive Technology	Smart glasses/AR headset	Video, Gaze, Voice, Scene context	Eye tracking, Ambient video
	VR/AR controller	Motion, Haptic feedback	Gesture, Button press
	Smart speaker	Audio, Voice command	Conversations, Voice tone
Physiological Sensing	Brain-computer interface	Neural signals, Emotion	EEG, Arousal, Cognitive load
	Skin sensors/wearables	Temperature, Galvanic response	Stress, Emotion, Touch pressure
	Eye tracker	Gaze, Blink, Pupil dilation	Fixation patterns, Attention shift
Environmental Systems	Car system	Location, Gaze, Driving behavior	Driving style, Eye direction
	Home IoT devices	Environment, Sound, Motion	Temperature, Appliance use
	Online behavior tracking	Text, Clickstream, Scroll	Search intent, Interest patterns

Table 2: Representative Multimodal Context Collectors by Category

Tolerance for Raw Context: From Structured Inputs to Human-Native Signals Before reaching human-level intelligence, a system's intelligence is principally governed by its degree of human-likeness, best measured by its tolerance for raw context, or in other words, the ability to consume and process high-entropy information input. In the 1.0 era of context-aware systems, input was limited to simple, structured signals like GPS coordinates, time of day, or predefined user states (Baldauf et al., 2007b; Dey, 2001a). These features were easy to process, but required developers to define in advance what counts as meaningful context. In contrast, modern systems in the 2.0 era can interpret context from signals that resemble natural human expression, such as free-form text, images, or video (Radford et al., 2021; Alayrac et al., 2022a). This shift is not merely about adding new input types; it reflects a deeper improvement in the system's ability to understand messy, ambiguous, and incomplete data. Enabled by advances in foundation models and multimodal perception, these systems can now handle inputs that were previously considered too raw or unstructured. As a result, context can be ingested directly in its native form, without the need for heavy pre-processing. This marks a fundamental step toward human-level flexibility in context interpretation.

Understanding and Utilization of Context: From Passive Sensing to Active Understanding and Collaboration Context-aware systems in the 1.0 era typically operated under simple condition-action rules (Schilit and Theimer, 1994; Abowd and Mynatt, 2000). In other words, they sensed predefined signals and triggered fixed responses. For example, “if location is office, then silence the phone.” These systems respond according to where you are, but not to what you are doing. In contrast, 2.0 systems aim to actively interpret what the user is doing and collaborate to achieve shared goals. For example, when you are writing a research paper, the system can analyze your previous paragraphs and current writing intentions to suggest a suitable next section. It does not just sense your environment; it integrates into your workflow. That is what we call context-cooperative; we develop from **context-aware** to **context-cooperative** systems.

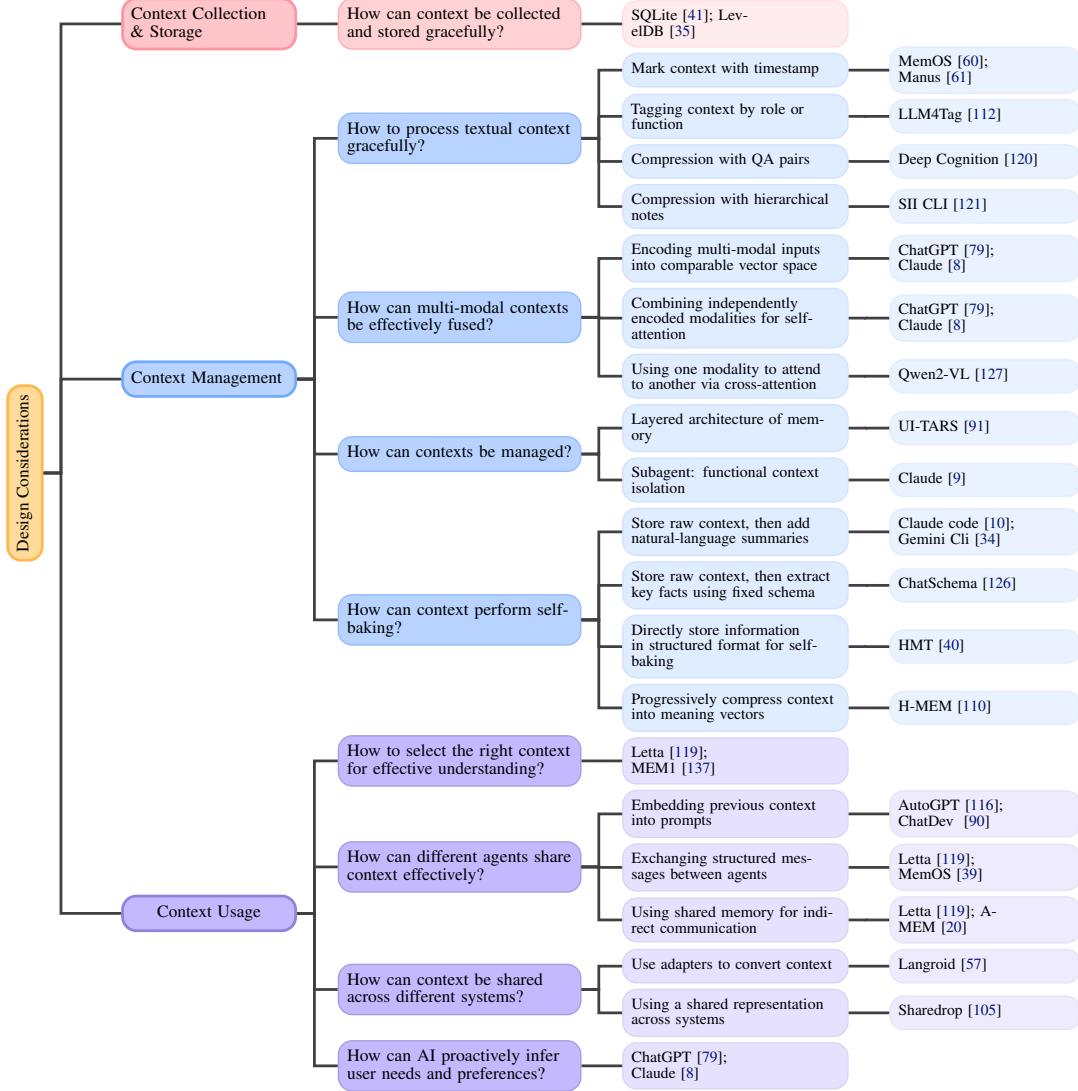


Figure 4: Design considerations of context engineering across different eras, note that the examples listed only cover a subset

4 Context Collection and Storage

As computing environments become more complex, there is a growing need for enhanced context collection capabilities. In response, advances in sensing technologies and AI now make it possible to collect context from richer sources (or sensors). Storage has also diversified across local devices, network servers, cloud platforms, etc., each with different trade-offs in latency, capacity, scalability, and security. Two fundamental design principles guide this process. The Minimal Sufficiency Principle states that systems should collect and store only the information necessary to support a task: The context value lies in sufficiency, not volume. The Semantic Continuity Principle emphasizes that the purpose of context is to maintain continuity of meaning, rather than merely continuity of data. Together, these principles shape how context should be collected and preserved in intelligent and reliable systems.

4.1 Typical Strategies in Era 1.0 and 2.0

In the early stages, context was collected primarily on a single device, such as a desktop computer or an early smartphone, using limited sensors (GPS, clock, keyboard/mouse events) or application logs that recorded usage patterns and user interactions (Schilit and Theimer, 1994; Dey, 2001a; Abowd et al., 1999b). Storage practices in this period were largely local. Context data was typically recorded as log files or structured documents within the local file system or stored in simple local databases. Temporary states, such as recent user input or window activities, were often kept in memory caches or temporary folders and discarded when the system was shut down. Although some systems attempted to upload context to centralized servers, these efforts were constrained by high latency and unstable network connectivity (Satyanarayanan, 2001). In general, storage strategies in this era prioritized

standalone availability on a single device rather than cross-device synchronization or secure data protection.

With technological progress, context collection became distributed across multiple endpoints, including smartphones, wearables, home sensors, cloud services, and third-party APIs (Lane et al., 2010; Miluzzo et al., 2008). Agents integrated multimodal signals into continuous context streams (Baltrušaitis et al., 2019). Storage practices of this period commonly adopted a layered architecture, with storage strategies determined by the intended usage of the data. For example, short-lived or frequently accessed data may be cached in fast-access memory or at edge nodes to minimize latency. Data requiring medium-term retention, such as activity records or user preferences, can be stored in local embedded databases (e.g. SQLite (Hipp, 2000), LevelDB (Ghemawat and Dean, 2014), RocksDB (Facebook, 2013)) or, where security is paramount, within OS-backed secure storage or hardware security modules (Kostiainen et al., 2012). For long-term persistence, scalability, and cross-device synchronization, cloud storage platforms and remote server databases may be used.

In the case of code agents, many tasks may run over long periods of time and often span multiple sessions, making it impractical to rely solely on the context window, which is both short-term and limited in capacity. To address this, systems periodically store the task state and progress in long-term memory so that an agent can resume work after interruption by restoring the relevant context. Such long-term memory can be maintained in local databases or secure storage, supported by cloud or remote services for cross-device synchronization, and, in some cases, even embedded into model parameters to provide more stable continuity and long-term adaptability. For example, Claude Code demonstrates a practical approach by maintaining structured notes, where key information is periodically written out of the context window into external memory and retrieved when needed. This strategy provides a lightweight but persistent form of memory, allowing the agent to track progress and avoid loss of information in complex tasks. More generally, this mechanism has been shown to support long-horizon activities, such as managing strategies over thousands of steps in a Pokémon game and then resuming seamlessly after resets (Team, 2025a). In this way, structured external memory extends the agent’s planning horizon far beyond the limits of a short and transient context window.

4.2 Human-Level Context Ecosystem

In the 3.0 era, AI systems achieve contextual awareness that rivals human perception. They may seamlessly collect tactile information (e.g. texture, pressure, temperature), recreating the sensory experience of human touch. Through smell and taste, they interpret environmental conditions, detecting smoke as a danger signal or assessing food freshness. In addition, they capture intent and emotion from vocal tone, pauses, eye contact, and even silence, understanding the subtle social contexts that define human interaction. Systems with human-level intelligence unify the context into a long-term *personal digital memory*. Storage is no longer just about preserving data, but serves as a dynamic cognitive infrastructure: capable of autonomously organizing, refining context to support continuous reasoning and interaction across scenarios and over time, as well as achieving human-like “forgetting” and “recalling” capabilities. Data flows securely between local and cloud environments, ensuring that users retain absolute control over sensitive information while still benefiting from global knowledge and computational resources, enabling natural human–machine symbiosis.

5 Context Management

5.1 Textual Context Processing

Effective context engineering isn’t just about collecting raw context, it is also about how we process it. A well-designed processing approach forms the foundation for everything that follows: interpretation, compression, and retrieval. It allows systems to focus on what matters, learn from past experiences, and build sustained long-term understanding. This section examines the fundamental question: How do we process raw textual context to get the best results? Beyond simply storing raw context, such as multi-turn dialogues, we examine several commonly used designs, along with their respective trade-offs.

Mark Context with Timestamp A common design is to attach a timestamp to each piece of information, preserving the order in which it was generated. This method is popular in chatbots and user activity monitoring due to its simplicity and low maintenance cost. However, this approach suffers from several limitations. Although timestamps preserve temporal order, they provide no semantic structure, making it difficult to capture long-range dependencies or retrieve relevant information efficiently. As interactions accumulate, the sequence grows linearly, leading to scalability issues in both storage and reasoning (Li et al., 2025; Perdigão, 2025).

Tagging Context by Functional and Semantic Attributes This approach organizes contextual information by explicitly tagging each entry with a functional role, such as “goal”, “decision”, and “action”, to make each entry easier to interpret. Recent systems enable tagging from multiple dimensions, including priority levels, source information, etc., thereby supporting more efficient retrieval and context management (Westhäuser et al., 2025;

Tang et al., 2025). Although this helps clarify the meaning of each piece of information, it can be slightly rigid and may limit more flexible reasoning or creative synthesis.

Compression with QA Pairs This method reformulates context into distinct question–answer pairs to improve retrieval efficiency, particularly in applications such as search engines or FAQ-based systems (LlamaIndex, 2024). However, it disrupts the original flow of ideas, making it less suitable for tasks that require a comprehensive understanding of the context, such as summarizing or reasoning (Hwang et al., 2024; Dai et al., 2025).

Compression with Hierarchical Notes This method organizes information in a tree-like structure, where broad concepts branch into increasingly specific sub-points. Although this structure helps to present ideas clearly, it primarily reflects how information is grouped rather than how ideas are logically connected. Relationships such as cause and effect or evidence and conclusion are often not represented. Furthermore, this design does not capture how understanding evolves over time, for example, when new insights emerge or existing ideas are revised (Team, 2025f; Mao et al., 2025; Liskavets et al., 2024; Che et al., 2024).

5.2 Multi-Modal Context Processing

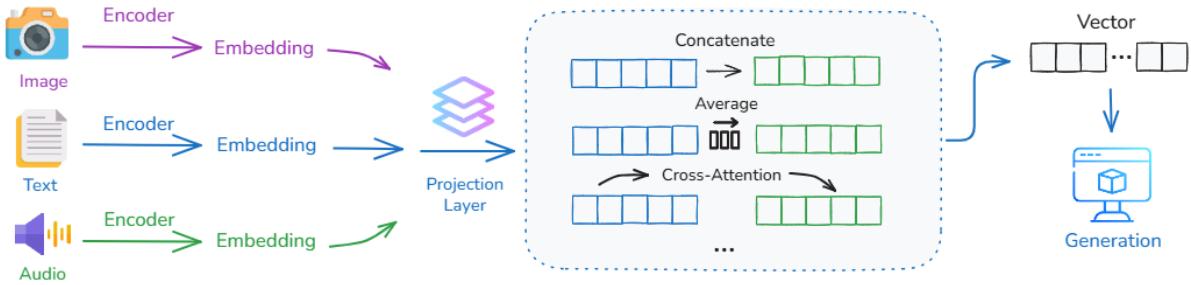


Figure 5: An example workflow for processing multimodal context with hybrid strategies

Context in LLM-based systems is becoming increasingly multimodal, including text, images, audio, video, code, sensor data, and even environmental states. As machines interact with real or simulated environments, they must be able to integrate information across modalities into a coherent and unified representation rather than processing each modality in isolation. A core challenge lies in the heterogeneity of these modalities: they differ in structure, information density, and temporal dynamics. For instance, text is discrete and sequential; images are high-dimensional and spatial; audio is continuous and unfolds over time. How can these be jointly encoded, compared, and reasoned over? In era 2.0, we identify several common strategies:

Mapping Multimodal Inputs into a Comparable Vector Space This method transforms inputs from different modalities, such as text, images, and video, into a shared vector space, so that their meanings can be directly compared. Each modality is first processed by its own encoder. Since these vectors initially live in separate representation spaces with different statistical properties, each is then passed through a learned projection layer that maps it into a shared embedding space of fixed dimensionality. In this space, semantically related content from different modalities is positioned close together, while unrelated content is pushed apart (Peng et al., 2023; Jaegle et al., 2021; Lv et al., 2024).

Combining Different Modalities for Self-Attention After being projected into a shared embedding space, modality-specific tokens are jointly processed by a single Transformer architecture. In this unified self-attention mechanism, text and visual tokens attend to each other at every layer, allowing fine-grained cross-modal alignment and reasoning. This approach, adopted by modern multimodal LLMs, enables the model to capture detailed correspondences, such as which phrase refers to which region of an image, rather than relying on the shallow concatenation of independent embeddings (OpenAI, 2024; Anthropic, 2025a).

Using One Modality to Attend to Another via Cross-Attention This method uses cross-attention layers to allow one modality (such as text) to directly focus on specific parts of another modality (such as images). Concretely, features of one modality are used as queries, while features of the other modality are treated as keys and values in the attention mechanism (Vaswani et al., 2017). This setup enables the model to retrieve relevant information across modalities in a targeted and flexible way. Cross-attention mechanisms can be flexibly implemented either as dedicated modules before the main Transformer architecture or embedded within the Transformer blocks themselves, depending on the overall system design (Yasunaga et al., 2023; Alayrac et al., 2022b). However,

traditional designs typically require specifying which modalities interact, whereas the human brain can flexibly integrate information across sensory and memory channels without relying on such fixed mappings.

5.3 Context Organization

5.3.1 Layered Architecture of Memory

Managing information effectively across different time scales is a fundamental challenge in AI systems. As Andrej Karpathy puts it, LLMs can be viewed through an operating system analogy: the model acts like a CPU, while its context window resembles RAM—fast but capacity-limited working memory. Just as an Operating System decides what data to load into RAM, context engineering determines what information should enter the window for effective reasoning (langchain Research, 2025). Like what we did in system, AI architectures benefit from separating memory into distinct layers based on temporal relevance and importance. This hierarchical approach allows systems to maintain quick access to recently relevant information while preserving valuable knowledge in more stable, long-term storage. For example, LeadResearcher systems store research plans in persistent memory when handling ultra-long contexts (>200k tokens), preventing key information from being lost due to context window limits (Team, 2025b). The core insight is that different types of information require different retention strategies. Recent contexts need fast retrieval, but may become irrelevant quickly, while important patterns and learned knowledge should persist across sessions. By organizing the memory hierarchically, systems can optimize both responsiveness and storage efficiency.

The framework we present here focuses on a two-layer model for clarity, but the principles extend naturally to more complex architectures. Real implementations often include additional intermediate layers, such as working memory for active processing, episodic buffers for recent events, or specialized caches for different domains, each with its own temporal characteristics and selection criteria (Qin et al., 2025; Stackademic, 2025).

Definition 5.1 (Short-term Memory). *Short-term memory is defined as the subset of context with high temporal relevance, selected by a processing function:*

$$M_s = f_{\text{short}}(c \in C : w_{\text{temporal}}(c) > \theta_s) \quad (5)$$

where $w_{\text{temporal}}(c)$ is the temporal relevance weight function of context element c , θ_s is the temporal relevance threshold for short-term memory, and f_{short} is a processing function that may involve human judgment, heuristic filtering, or system-level operations.

Definition 5.2 (Long-term Memory). *Long-term memory is defined as the processed and abstracted subset of context with high importance:*

$$M_l = f_{\text{long}}(c \in C : w_{\text{importance}}(c) > \theta_l \wedge w_{\text{temporal}}(c) \leq \theta_s) \quad (6)$$

where $w_{\text{importance}}(c)$ is the importance weight of context element c , θ_l is the importance threshold for long-term memory, and f_{long} is a composite function that may combine selection, abstraction, and compression to generate stable representations.

Definition 5.3 (Memory Transfer). *The transfer from short-term to long-term memory is defined as:*

$$f_{\text{transfer}} : M_s \rightarrow M_l \quad (7)$$

This transfer function represents the consolidation process where frequently accessed or highly important information in short-term memory is processed to become part of long-term memory. The transfer is governed by factors such as repetition frequency, emotional significance, and relevance to existing knowledge structures.

5.3.2 Context Isolation

Subagent A subagent provides an alternative way around context limitations while also reducing the risk of context pollution, representing an emerging strategy for effective context management through functional context isolation (Team, 2025a). The Claude Code subagent system illustrates this principle: Each subagent is a specialized AI assistant with its own isolated context window, custom system prompt, and restricted tool permissions. When a task matches the expertise of a subagent, the main system can delegate it to that unit, which then operates independently without contaminating the primary context of conversation. This principle can be applied to context selection by implementing separation along functional dimensions (e.g., analysis, execution, validation) or hierarchical layers (e.g., planning, implementation, review), where each isolated unit receives only the minimum permissions required for its specific responsibilities, improving both system reliability and interpretability (Anthropic, 2025b). Unlike static retrieval approaches such as RAG, the LeadResearcher first makes a plan. If the available information is insufficient, it can issue further searches, adjust keywords, or create new sub-tasks. The subagents may work in parallel, while the LeadResearcher summarizes interim results and decides on the next steps. This feedback loop helps the system converge on high-quality answers (Team, 2025b).

Lightweight References Context isolation often relies on storing large information externally and exposing only lightweight references in the model’s window. In the sandbox approach, as used in HuggingFace’s CodeAgent, bulky output is stored in a separate sandbox and retrieved only when needed. In this way, the model interacts only with concise references, while the sandbox holds the full data and provides it on demand. A similar principle applies to schema-based state objects, where heavy elements such as files or logs remain in external storage and only selected fields are surfaced. Both approaches reduce token overhead while retaining access to the complete context when required (Martin, 2025).

5.4 Context Abstraction

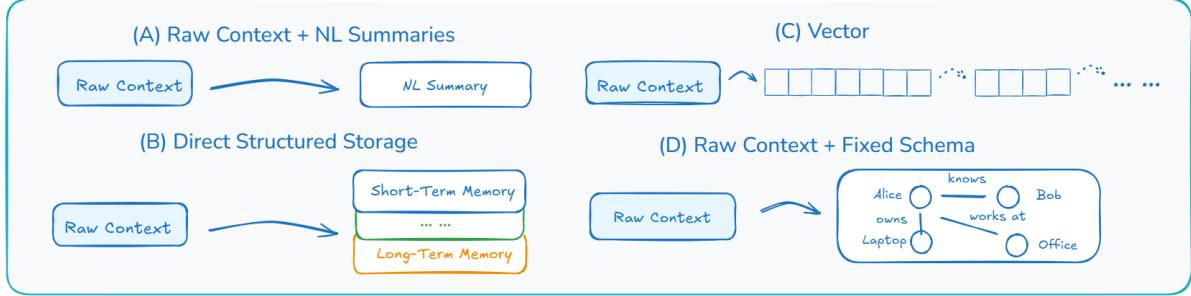


Figure 6: Representative designs for self-baking in Era 2.0

Raw context, such as dialogue turns, tool output, and retrieved documents, can accumulate quickly. If left unprocessed, the growing history can quickly overwhelm the system, making it difficult to identify what is truly important for future reasoning or decision making. To manage this, a crucial capability for scalable agents is **context abstraction**: converting raw context into more compact, structured representations. We refer to this process as *self-baking*: the agent selectively **digests** its own context into persistent knowledge structures. This mirrors human cognitive processes, where episodic memories give rise to semantic memory, or where repeated actions are abstracted into habits. Ultimately, self-baking is what **separates memory storage from learning**. Without self-baking, agents simply recall; with it, they accumulate knowledge.

Using Hierarchical Memory Architectures Current systems typically adopt an architectural principle: hierarchical memory organization. Hierarchical memory provides a principled way to manage growing contexts by organizing information at different levels of abstraction. At the base layer, raw contexts are stored to ensure that fine-grained details remain accessible. As the volume of context increases, these raw items are progressively summarized into more abstract representations, which are then passed to the next layer. New information typically enters at the lowest layer and is gradually “baked” upward, allowing the system to scale without overwhelming the context window while still retaining retrievable links back to the original details. In this sense, hierarchical memory complements the distinction between short-term and long-term memory: the raw context typically resides in lower short-term memory, while more abstract representations tend to correspond to long-term memory (He et al., 2025; Patel and Singh, 2025).

Add Natural-Language Summaries In this pattern, the system stores the full context in its original, unstructured form. In addition, it periodically generates summaries that provide a compressed view of what has happened so far. These summaries, usually written in natural language, give a quick overview of recent events and can be created either manually or automatically. For example, a chatbot might store the full dialogue history and generate a brief paragraph that summarizes the recent conversation. As the number of summaries grows, the systems may apply multilevel summarization (summarizing older summaries into higher-level overviews) or use specific strategies to drop less useful ones based on time or importance (Martin, 2025). These summaries help the system focus on key information while still keeping the full details available if needed. This method is simple and flexible. However, because the summaries are just plain text, they lack structure. This makes it difficult for the system to understand the connections between events or perform deeper reasoning over the context (Smith and Zhang, 2025).

Extract Key Facts Using a Fixed Schema This pattern extends the first by adding a structured interpretation. The system not only stores the raw context, but also extracts key information into a predefined format to make it easier to access and reason over. The schema can take different forms: it might be an entity map, which represents key entities (such as people, items, or places) as nodes. Each node maintains the properties of the entity (e.g., name, type), current state (e.g., location, status) and links to other entities (e.g., “owns”, “works with”, “is located at”). Or, it can be event records, which works as a template that breaks an event into different aspects. In addition, it can be

a task tree, where complex goals are broken down into subtasks in a hierarchical structure. A concrete example is CodeRabbit, which constructs a structured case file prior to code review, encoding cross-file dependencies, historical PR information, and team-specific rules into an explicit schema, enabling the AI to reason over the complete system context rather than isolated file changes (Prasad, 2025). This approach enables more effective reasoning than raw summaries by allowing the system to retrieve, analyze, and relate specific facts within an explicit schema. However, maintaining multiple layers can lead to inconsistencies, and designing good extractors to fill in these schemas reliably remains a significant challenge (Du et al., 2025).

Progressively Compress Context into Vectors that Capture The Meaning Apart from storing raw contexts, this approach encodes information as dense numerical vectors, known as embeddings, that reflect the meaning of the input. These vectors are compressible, allowing the construction of multilevel (hierarchical) memory that abstracts context at different scales. These vectors undergo self-baking, where older embeddings are periodically summarized into compact representations (typically through pooling), or fused with existing long-term states, and re-encoded to form progressively more abstract and stable semantic memories. This method is compact and flexible, and is particularly useful for semantic search or matching similar queries. However, the resulting representations are not human-readable, making it challenging to edit or inspect specific parts of the memory (Sun and Zeng, 2025; Chen and Xu, 2024).

6 Context Usage

6.1 Intra-System Context Sharing

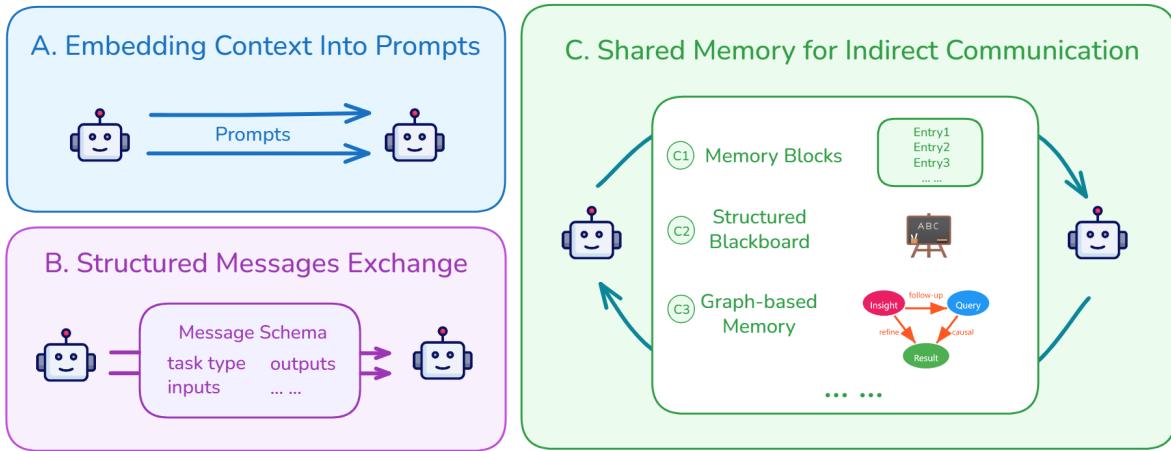


Figure 7: Common patterns of cross-agent context sharing

Modern LLM applications often comprise multiple agents, each responsible for part of a larger reasoning workflow. A practical reason why multi-agent systems work is that they allow the system to consume more tokens than a single agent could handle, effectively extending the capacity of the overall process (Team, 2025b). In such systems, a key challenge arises: How can these agents share context to achieve coherent, collaborative behavior? Unlike single-agent prompting, multi-agent systems require clear and structured ways to pass information between agents. One agent may produce a partial answer or an intermediate result, which must then be understood and used by another. To support this, the shared information must be accurate, easy to interpret, and directly usable for the next step. We identify several common patterns of cross-agent context sharing:

Embedding Previous Context into Prompts This method passes information by including the previous context directly in the input prompt of the next agent. Often, the information is reformatted to be clearer. For example, one agent may summarize its thought process as plain text, which the next agent reads and continues from. This way of passing context is common in systems like AutoGPT (Team, 2025c) and ChatDev (Liu and Kumar, 2025; Qian et al., 2024), where agents work in sequence using prompts as their communication channel.

Exchanging Structured Messages between Agents Agents communicate by exchanging structured messages using a fixed format. These messages typically follow a predefined schema with fields such as task type, input data, output results, and reasoning steps. The receiving agent reads this message and continues the task. Systems like Letta (Team, 2024) and MemOS (Han et al., 2025) use this approach to maintain clarity and consistency when passing information between agents.

Using Shared Memory for Indirect Communication Agents often communicate indirectly by reading and writing to a shared memory space, which can be implemented as a centralized external storage or as a public area within the memory of an individual agent. Instead of sending messages directly, agents leave information in this shared space, which is often organized as memory blocks, and check it later to receive updates. Each block can store a unit of context and is typically labeled with basic metadata, such as who added it, when, and what type of data it contains. Systems like MemGPT (Team, 2024) and A-MEM (Chen et al., 2024a) adopt this approach to support indirect communication and coordination between agents. Another way is to make memory more structured: Instead of placing raw data in a general pool, information is written on a shared “blackboard” which is organized into segments based on topics, tasks, or goals (Du et al., 2025; Salemi et al., 2025). Each agent monitors only the segments relevant to its expertise and adds or modifies the entries accordingly. This modular structure reduces confusion while allowing agents to collaborate asynchronously. Memory can also be organized as a graph. For example, the Task Memory Engine (TME) (Ye, 2025) represents an agent’s reasoning process as a task graph: Each node encodes a single step, including its input, output, and execution status, while edges capture dependencies between steps. This structure enables agents to track, reuse, and resume multi-step reasoning in a reliable and interpretable way. Similarly, G-Memory (Zhang et al., 2025) models memory as a semantic graph, where nodes represent insights, user queries, and intermediate results, and edges reflect relationships such as follow-up, refinement, or causal linkage. Such graph-structured memory enables a richer representation of context and more effective reasoning across multiple steps or tasks.

6.2 Cross-System Context Sharing

In the context of multi-agent environments, a system can be understood as any independent platform, model, or application that maintains its own context or state for performing tasks. While different systems may vary in scale and capabilities, they each manage information within their own boundary. Sharing context across systems allows these distinct entities to access or exchange relevant information, enabling better coordination or reasoning. For example, sharing context between Cursor and ChatGPT illustrates this kind of cross-system interaction. When context is shared within a single system, it is generally easier because the components are designed to interoperate — they mostly use compatible data formats, follow similar rules, and expect similar types of input and output. As a result, context can often be exchanged without extensive translation. In contrast, when sharing context across different systems, each system may use its own format, structure, and logic. What makes sense in one system may look completely unfamiliar to another. In this case, the challenge is to make the shared context interpretable across boundaries.

Use Adapters to Convert Context Each system keeps using its own format and adds a converter to translate the context into something that the other system can read. This gives systems more freedom, but means that you have to build a separate adapter for each connection (Zarabzadeh et al., 2024; David, 2024).

Using a Shared Representation Across Systems All systems agree to use the same representation from the beginning, ensuring that each can read and write directly. This avoids the need for translation between every pair of systems and makes integration much simpler (Team, 2025e; Gupta, 2025). Shared representations can take different forms. Common approaches include the following:

Using a Shared Data Format such as JSON or an API Systems can agree on a fixed format for the context, such as a shared JSON schema or a well-defined API. This allows each system to read and write context in a consistent way, avoiding the need for custom translation logic (Anderson, 2025; Sahu, 2025).

Sharing Context via Human-Readable Summaries Instead of relying on formal data structures, systems can exchange short descriptions of natural language. These summaries are easy for humans to understand and can also be interpreted by language models when needed (Dobre et al., 2025; Team, 2025i).

Representing Context as Semantic Vectors Context can also be represented as numeric vectors that capture the meaning of information. This method is flexible and system independent, but often requires machine learning models to interpret correctly (Raj, 2025; Team, 2025d; Petrova et al., 2025).

Each method involves trade-offs. Standard schemas offer precision, but require strict coordination. Natural language is flexible and easy to generate, but more difficult to parse reliably. Semantic vectors are compact and generalizable, but less interpretable. The best choice depends on the systems involved and the goals of the communication.

6.3 Context Selection for Understanding

Even with extended context windows, LLMs remain bottlenecked by the quality of input tokens they can condition on. So an important question emerges: What subset of available context should be selected for the current step?

Contexts may originate from a scratchpad, from memory, from tool definitions, or from knowledge retrieved via RAG, but not all contexts are equally useful (Martin, 2025). Irrelevant or noisy memory fragments can distract reasoning or increase the cost of inference. Empirically, AI coding performance often decreases when context windows exceed roughly 50% fullness, suggesting that both excessive and insufficient context can degrade effectiveness (Osmani, 2025). Without robust filtering, agents risk being overwhelmed by their own memory, a modern form of “context overload”. Thus, effective context selection becomes a form of *attention before attention* — choosing what deserves to be paid attention.

With the rise of LLM-powered agents, the context engineering 2.0 era shifted toward dynamic, goal-driven interaction. Context filtering became an adaptive process, dynamically selecting the information that is most relevant to the current intent of the user (Smith and Zhang, 2025; Patel and Singh, 2025). We identify several factors that should be considered during memory selection:

Semantic Relevance It refers to selecting memory entries that are most similar in meaning to the current query or objective. This is typically implemented via vector-based retrieval: both query and candidate entries are encoded as dense embeddings, and a nearest-neighbor search (e.g., FAISS or other approximate nearest-neighbor methods) retrieves items by similarity, allowing semantically related items to be found even without exact keyword matches (Johnson et al., 2019). This approach is commonly used in retrieval-augmented generation (RAG) pipelines and systems such as Letta (Team, 2024; Chen and Xu, 2024; Sun and Zeng, 2025).

Logical Dependency Logical dependency refers to cases where the current task directly relies on information produced by previous steps, such as prior planning decisions, outputs from tools, or reasoning chains. Systems such as MEM1 address this by explicitly recording reasoning traces during execution: whenever a new step is generated, MEM1 stores it in a memory slot and links it to the earlier steps it depends on (e.g., prior analyses or tool outputs). Over time, this forms a structured dependency graph across memory entries. When a new query arises, MEM1 can traverse this graph to retrieve only the context that lies in the relevant dependency chain, ensuring that memory reflects not just surface relevance but also the logical flow of the task (Xu et al., 2025; Li and Garcia, 2024; Zhao and Kim, 2025).

Recency and Frequency Items that were used recently or accessed often are more likely to be retrieved again. This is based on simple heuristics: if something was useful before, it might be useful again. To implement this, systems typically assign higher priority to newer memory entries and gradually lower the priority of older ones, so that outdated information naturally becomes less influential. At the same time, entries that are repeatedly accessed accumulate a higher importance, ensuring that commonly referenced information remains readily available. By balancing these factors, the memory pool evolves to favor fresh and important contexts (Team, 2024; Patel and Singh, 2025).

Overlapping information If multiple pieces of information convey the same meaning, the older or less detailed ones can be filtered out (Du et al., 2025; Jiao et al., 2024). In modern systems, this process is increasingly handled through active memory management, where the system does not merely detect similarity passively but actively decides when to merge, update, or remove redundant entries to maintain a concise and relevant memory pool (Yun et al., 2025).

User Preference and Feedback Over time, AI agents can adapt to a user’s habits: learning what types of information the user tends to value. Some systems, such as self-evolution memory (Jin et al., 2024; Zhao and Kim, 2025), track how users interact with information and use it to adjust the importance or weight of memory entries.

A primary consideration in memory selection is **relevance**. This can be assessed by several factors, including **semantic similarity**, **logical dependency**, **recency** (the timeliness of the information), and **frequency of mention**. For example, many systems employ similarity scores to compare current input with stored content, assigning higher relevance to information with greater similarity. Alternatively, explicit tags or metadata can be incorporated to indicate the importance or function of specific data, such as marking certain events as milestones or highlighting key facts. In addition to relevance, it is important to **minimize redundant information** and **adapt to user habits**. By integrating these criteria, systems are able to retain the most pertinent data, reduce unnecessary noise, and enhance the efficiency of context selection.

Common Filtering Strategies Taking these factors into account, the systems adopt different filtering strategies. In RAG pipelines, the first step is to segment the source into manageable chunks, which may be done with simple strategies such as fixed line or token windows, or with more structured approaches like AST-based segmentation that respects function, class, or module boundaries and preserves semantic coherence. Retrieval then proceeds in several distinct ways. Semantic retrieval often relies on an embedding-based search, which selects fragments according to the vector similarity to the query (Chen and Xu, 2024). Non-semantic retrieval can be as direct as

Grep, which uses string or regex matching without interpreting meaning. Structured retrieval uses knowledge graphs, drawing on entities and dependency relations (e.g., function-call graphs) to connect information across files and modules (Wen et al., 2025). Because these methods frequently return overlapping or noisy candidates, many systems introduce a reranking stage (sometimes powered by LLMs) to refine relevance, trading off accuracy improvements against efficiency, with implementations (e.g., Windsurf) varying in their design choices (Mohan, 2025).

6.4 Proactive User Need Inference

Most current uses of context are reactive, yet users often find it difficult (or unreasonable) to fully articulate their needs and preferences. To bridge this gap, context engineering should enable agents to act *proactively*: to infer latent user needs, preferences, and goals that are not explicitly stated, and to initiate helpful interactions accordingly. This is analogous to human assistants who learn over time that “user prefers visual summaries”, or that “evening hours are best for brainstorming”. Such insights are not predefined; they must be mined, abstracted, and validated through the accumulated interaction context. In era 2.0, we identify several common forms of proactive preference mining:

Learning and Adapting to User Preferences Modern AI agents learn and adapt to user preferences primarily by analyzing conversation history and stored personal data such as user documents, notes, and past interactions, in order to identify patterns in communication style, interests, and decision-making approaches (Pan et al., 2025; Sun et al., 2024). In many cases, these insights are consolidated into evolving user profiles that guide more personalized interactions. They also learn from indirect signals by observing how users respond to suggestions, whether they continue conversations, and their apparent satisfaction with completed tasks, incorporating these observations into future interactions (OpenAI, 2024). In addition, sometimes agents actively explore preferences by directly asking users about their likes and dislikes or presenting multiple options during conversations to gather explicit feedback (Ryu et al., 2025).

Inferring Hidden Goals from Related Questions The system can infer hidden goals by analyzing the sequence of user queries. For instance, if a user asks about Python decorators and then about performance tuning, this may reflect a broader goal of improving software design efficiency. To support such inference, systems can encode the current context and put it into an LLM to predict users’ potential needs (Anthropic, 2025a). Moreover, chain-of-thought reasoning techniques enable systems to perform multi-step logical deduction, inferring deep intentions from surface-level user inputs (Yao et al., 2025). Recognizing these patterns, the system can offer more goal-aligned assistance.

Proactively Offering Help Based on User Struggles The system detects when the user may be stuck, such as showing hesitation or trying many alternatives, and proactively offers useful tools like visualizations or checklists (Chen et al., 2025). These interventions aim to improve the quality of decision without asking the user.

6.5 Lifelong Context Preservation and Update

As noted in the abstract, individuals are fundamentally shaped by their interactions with other entities — in other words, by their contexts. Given this shift, the central challenge naturally arises: Once context takes on a lifelong form, how should it be preserved and updated in a way that remains coherent, adaptive, and usable for both humans and machines? The continuous preservation and updating of lifelong context demands more than scalable storage: It requires the design of a memory system that is dynamic, semantically robust, and temporally aware. This raises a multifaceted technical challenge, where naive extensions of current memory architectures collapse under the scale, fluidity, and complexity of real-world context.

Here is the storyline: When “normal” context engineering becomes lifelong context engineering, a problem arises: How to implement **reliable storage mechanisms** that maintain semantic consistency? Moreover, mere storage is insufficient; as data scales to massive volumes, the system must be capable of accurately **processing and managing** this information. Once a new design is proposed, an end-to-end **evaluation** framework becomes essential to validate both its correctness and performance. However, such mechanisms are still largely underdeveloped. Once the complete pipeline is established, maintaining its **stability** becomes paramount. Each stage within this workflow presents distinct challenges, particularly in the domain of lifelong context systems.

Challenge I: Storage Bottlenecks The first challenge is how to retain as much relevant context as possible under strict resource constraints. We currently lack a unified solution: How can we preserve as much context as possible, ensuring that all of my contexts can be effectively retained without loss? What kind of infrastructure or interface would facilitate recording our context to the maximum extent? And how can storage systems simultaneously support high-compression, high-precision retrieval, and low-latency access at scale (Xing et al., 2025; Ahn, 2025)?

Challenge II: Processing Degradation Another challenge arises from the collapse of attention mechanisms at scale. Most transformer-based models rely on global attention, whose $O(n^2)$ complexity leads to rapidly increasing inference latency, high GPU memory usage, and slower I/O throughput. These resource bottlenecks make it impractical to handle large contexts in real time. Meanwhile, the quality of reasoning deteriorates. As attention becomes thinner across a longer input, the model struggles to maintain focus on relevant information, and it struggles to capture long-distance dependencies (Thai et al., 2023; Rabe and Staats, 2022). In addition, retrieval systems become overwhelmed by many semantically similar but irrelevant pieces of information, often returning distractions instead of useful evidence (Jeong et al., 2024). Furthermore, as the volume of context grows, inconsistencies and conflicts between retrieved segments become harder to detect and reconcile. All of these issues lead to fragile reasoning when dealing with very large contexts.

Challenge III: System Instability As memory accumulates over time, even small mistakes can affect more parts of the system. Errors that once had limited impact may now spread widely, leading to unexpected or unstable behavior. Without clear boundaries or validation mechanisms, the system becomes more difficult to manage, especially in tasks that run for a long time or require high safety. In such cases, it might make the system more fragile instead of increasing reliability (Meng et al., 2025; Jacobs, 2025).

Challenge IV: Difficulty of Evaluation As memory accumulates, it becomes harder to tell whether the system is reasoning correctly. Most benchmarks today only test whether the system can retrieve information, but do not check whether the information is still relevant, accurate, or helpful. Systems rarely include features to check for contradictions, undo wrong updates, or trace the reasoning steps that led to a conclusion. As more decisions depend on longer memory chains, it becomes increasingly difficult to inspect how the system reached a specific answer. This lack of visibility makes it harder to trust or improve the system, especially for lifelong context engineering (Zheng et al., 2025).

Toward a Semantic Operating System for Context The challenge of lifelong context engineering can no longer be addressed by simply “expanding the context window” or “improving retrieval accuracy”. It demands the construction of a **semantic operating system** capable of growing over time, much like a human mind. On the one hand, such a system must support large-scale, efficient semantic storage as its own memory bank, and exhibit truly human-like memory management abilities: actively adding, modifying, and forgetting knowledge. On the other hand, it calls for **novel architectures** to replace Transformers’ flat temporal modeling, thereby enabling more powerful long-range contextual reasoning and dynamic adaptation. Crucially, the system should be able to explain itself by tracing, correcting, and interpreting each step in its reasoning chain, thus improving trust and reliability in practical and safety-critical scenarios. This approach highlights a fundamental principle of context engineering and reflects a paradigm shift: context is no longer passively accumulated, but is actively managed and evolved as a core element for cognition.

6.6 Emerging Engineering Practices

KV Caching Under emerging engineering practices, the use of key-value (KV) caching has become central to the efficient deployment of agents. KV caching works by storing the attention states (keys and values) of past tokens so they do not need to be recomputed when new tokens are generated. As a result, the cache hit rate strongly affects both latency and cost. To improve hit rate, several practices are proving essential: first, keeping prefix prompts stable, since even minor variations such as timestamps at the beginning of the system prompt can invalidate the entire cache; second, enforcing append-only and deterministic updates, because altering or inconsistently serializing past content breaks reuse; and third, in cases where serving frameworks do not support automatic incremental prefix caching, it is necessary to insert cache checkpoints manually and place them carefully (Manus, 2025). In addition, cache warm-up is often employed to further enhance efficiency. A common approach is predictive loading (prefetch or speculative loading), in which the system anticipates which contexts are likely to be needed next and loads them into the cache in advance (Fridman, 2025). These techniques show that efficiency is increasingly determined by the way context is managed.

Tool Designing The important factors in tool design are description and scale. For description, tools need precise purposes and clear definitions. Vague or overlapping descriptions often cause failures, while well-structured ones reduce ambiguity and improve reliability. Models can also refine these descriptions and act as prompt engineers, enabling self-improvement (Team, 2025b). For scale, large tool sets make agents less reliable, and dynamically loading tools during interaction often breaks KV-cache consistency and confuses references to earlier actions. Empirical observations suggest that excessively large tool sets can degrade performance, as overlapping tool descriptions and increased choice complexity make selecting the correct tool more difficult. For DeepSeek-v3, performance declined beyond 30 tools and was nearly guaranteed to fail beyond 100 (Dbreunig, 2025). A more robust approach is to keep the tool list stable and enforce constraints at the decoding level, for example, by masking

token logits to block invalid choices. This practice preserves efficiency while reducing errors caused by changing the action space (Manus, 2025).

Context Contents An agent should not hide an agent’s mistakes; retaining errors in the context allows the model to observe its failures, which is crucial for learning corrective behavior and improving overall performance (Manus, 2025). In agent settings, traditional few-shot prompting can be counterproductive. When the context contains largely similar past action-observation pairs, the model tends to repeat prior actions, simply following the patterns it sees. To mitigate this, Manus introduces small, structured variations in actions and observations, such as alternative serialization templates, varied phrasing, or subtle changes in order and formatting. These controlled perturbations break repetitive patterns and help refocus the model’s attention, improving robustness and reducing the risk of overgeneralization (Manus, 2025).

Multi-agent Systems Claude’s experience suggests that effective multi-agent work depends on a few recurring practices. The lead agent, or lead researcher, breaks queries into subtasks and assigns them with clear goals, outputs, tool guidance, and boundaries, as vague instructions lead to confusion or gaps. Because agents struggle to judge workload, prompts can include simple adjustment rules relevant to query complexity, for example, simple tasks may need 1 agent to complete, while harder ones may need more. Search strategies work best when they move from broad exploration to focused analysis, and extended thinking mode, where agents explicitly write down their reasoning process, enhances overall accuracy and efficiency (Team, 2025b).

Tricks In executing complex tasks, many systems maintain a todo.md file listing subgoals, updating it and marking items as completed as the task progresses. However, models can lose track of earlier objectives in long tasks. A practical solution is to recite these goals in natural language when updating the todo list, integrating them in the recent context of the model to keep the key objectives within its immediate attention (Manus, 2025).

7 Applications

7.1 CLI

When using an AI agent, developers often require sustained, project-oriented context support. Google’s *Gemini CLI* offers a representative case of how context can be engineered. A central mechanism is the *GEMINI.md* file, a Markdown specification that records the project background, role definitions, required tools and dependencies, coding conventions, etc. The contexts are organized through the file system hierarchy: *GEMINI.md* files can exist in the user’s home directory, in the project root, or within subdirectories, enabling both inheritance and isolation of information (philschmid, 2025). For collection, the CLI gathers two types of context: at startup, it automatically loads static information such as system prompts, the current project environment, and the ancestor or descendant *GEMINI.md* files; during interaction, it incrementally accumulates dynamic context from the ongoing dialogue history (Irani, 2025c; Lin et al., 2025). For management, the file system itself functions as a lightweight database, with mechanisms to compress context by replacing long interaction histories with AI-generated summaries. These summaries follow predefined formats that preserve key aspects of the dialogue (e.g., overall goal, key knowledge, file system state, recent actions, and current plan), ensuring consistency. Community discussions have also suggested extending this process with human refinement to support collaborative management of context (Irani, 2025b,a; Xiao et al., 2025).

7.2 Deep Research

Deep research agents aim to assist users in addressing open-ended, knowledge-intensive queries, such as reasoning tasks involving multiple events and intertwined relationships. A representative example is Tongyi DeepResearch, which operates in four main steps: it searches the web based on the user’s query, extracts key information from relevant pages, generates new sub-questions to guide further search, and finally integrates evidence from multiple sources into coherent answers (Team, 2025h). This cycle often continues for many rounds until uncertainty is reduced and a complete evidence chain is formed. Unlike short-turn conversational agents, deep research faces the challenge of extremely long interaction histories: directly appending all observations, thoughts, and actions quickly exceeds the context window. To overcome this limitation, Tongyi DeepResearch adopts systematic context engineering. During exploration, the agent periodically invokes a specialized summarization model to compress accumulated history into a compact reasoning state, which not only preserves critical evidence but also highlights missing information and next-step directions. Subsequent searches and reasoning are then grounded in these compressed “context snapshots” rather than the complete raw history. In this way, the system establishes a clear context lifecycle: from collecting and accumulating information, to periodic compression and abstraction, and then to reasoning and reuse based on summaries, allowing it to break through context constraints and achieve scalable, long-horizon research capabilities (Wu et al., 2025).

7.3 Brain-Computer Interfaces

Brain-Computer Interfaces (BCIs) offer a novel pathway for context engineering by enabling more advanced context collection (Tang et al., 2023). Unlike traditional methods that rely on language input, BCIs can directly capture neural signals, which introduces two distinct advantages. First, they allow the collection of richer contextual dimensions, such as attention levels, emotional states, or cognitive load — factors that are often difficult to observe through external behavior alone. Second, they provide a more convenient way of collecting context, reducing the need for explicit user actions, and enabling more immediate input through neural activity (Liu et al., 2025; Edelman et al., 2024). Although current techniques offer only a coarse understanding of brain signals and issues such as noise and instability remain significant challenges, BCIs highlight a direction in which context engineering may evolve: extending context collection beyond external environments to the user’s internal cognitive state (Wu et al., 2023).

8 Challenges and Future Directions

Although we have outlined a historical and conceptual framing of context engineering, a number of open challenges remain. Below we highlight several key issues and sketch possible directions for future exploration.

Context collection remains limited and inefficient Most current agent systems still rely on explicit user input to obtain context, which is both cumbersome and inefficient (Zhao et al., 2024). In addition, users are sometimes unable to articulate their intentions clearly, leaving important contextual information underspecified. Progress requires more natural and multimodal methods of context collection, together with models that can better infer user needs and fill in missing contexts (Chen et al., 2024b). One promising line of work is brain–computer interfaces, which aim to collect user state and intent more efficiently than explicit articulation. Such advances suggest that richer forms of context collection may eventually overcome the inherent limitations of text-based input (Liu et al., 2025; Edelman et al., 2024).

Storage and management of large-scale context As interactions accumulate, the size and complexity of the context grow rapidly. A key challenge lies in deciding how context should be stored and organized and how to organize it in a way that remains scalable while supporting effective selection and retrieval. Without careful design, large-scale context can become cumbersome and difficult to use for downstream tasks (Masood, 2025).

Limited model understanding of context Current systems do not possess the same level of contextual understanding as humans. For example, they struggle with complex logic and with relational information in images (Yang et al., 2023). As mentioned in this paper, lower machine intelligence leads to greater “effort” in context engineering. As a result, much of the available context is not fully understood or utilized. Future work should focus on strengthening models’ abilities in semantic reasoning, logical interpretation, and multimodal alignment, enabling systems to better understand context and reduce reliance on human-driven context engineering (Pan et al., 2023).

Performance bottlenecks with long context The processing of long contexts remains a central challenge. Transformer-based architectures suffer from quadratic complexity, making them inefficient as the context grows. Recent alternatives like *Mamba* (Gu and Dao, 2024) and its variants such as *LongMamba* (Ye et al., 2025b) have improved efficiency and scalability, but still show weaknesses in long-context understanding. For example, when input length far exceeds training length, or when relational and logical dependencies span the entire context. LOCOST (Bonnecc et al., 2024) can handle documents of hundreds of thousands of tokens, but still lags behind transformers in certain tasks that require fine-grained reasoning over very long spans. Moving forward, there is a clear need for new architectures that can handle much longer contexts efficiently while also providing stronger and more reliable understanding, rather than treating long contexts merely as extended input.

Selecting relevant and useful context Not all available context contributes to the task at hand. Although current systems already employ relevance estimation and filtering mechanisms, their performance remains limited: useful signals may be missed, and noisy or redundant information often persists. A key research direction is to develop more precise and adaptive context selection strategies that continuously refine what to keep, discard, or emphasize, ensuring that the retained context remains closely aligned with the objectives of the task (Chen and Xu, 2024; Mohan, 2025).

Digital Presence Karl Marx once wrote that “the human essence is the ensemble of social relations” (Marx, 1845). In the era of context-centric AI, this idea takes on a new computational meaning: individuals are increasingly defined not by their physical presence or conscious activity, but by the digital contexts they generate: their conversations, decisions, and traces of interaction. These contexts can persist, evolve, and even continue to interact with the world through AI systems long after the departure of a person. The human mind may not be uploaded, but the human context can — turning context itself into a lasting form of knowledge, memory, and identity.

9 Conclusion

In this paper, we explore the context of context engineering, arguing that it is not a sudden invention of the LLM era but a long-evolving discipline shaped by the progressive intelligence of machines. By tracing its historical phases and by outlining design considerations that govern its practice, we highlight how the core challenge lies in bridging human intent and machine understanding under varying levels of entropy. Our proposed trajectory suggests a gradual human disengagement from explicit context management, as increasingly intelligent machines take on greater responsibility for interpreting, reasoning, and even constructing context. Looking ahead, as machine understanding approaches and potentially surpasses human cognition, culminating in a possible “god’s eye view” of our intentions, AI systems may not only comprehend us, but also illuminate and expand our understanding of ourselves.

Acknowledgement

Our sincere thanks go to Xiangkun Hu from Amazon for his early discussions. We are also deeply thankful to Shijie Xia from Shanghai Jiao Tong University for his valuable feedback.

References

- [1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999a. **Towards a better understanding of context and context-awareness**. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer.
- [2] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999b. **Towards a better understanding of context and context-awareness**. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307.
- [3] Gregory D. Abowd and Elizabeth D. Mynatt. 2000. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58.
- [4] Kwangseob Ahn. 2025. **Hema : A hippocampus-inspired extended memory architecture for long-context ai conversations**.
- [5] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Wenlong Huang, Sander Dieleman, Andrew Zisserman, Karen Simonyan, and João Carreira. 2022a. **Flamingo: a visual language model for few-shot learning**. *Advances in Neural Information Processing Systems*, 35:23716–23736.
- [6] Jean-Baptiste Alayrac, Jeff Donahue, Peter Luc, et al. 2022b. **Flamingo: A visual language model for few-shot learning**. *arXiv preprint arXiv:2204.14198*.
- [7] Sean C. Anderson. 2025. **How to build multi agent ai systems with context engineering**. *Vellum AI Blog*. Highlights the role of Context Engineering in Multi-Agent Systems, where structured data formats like JSON are often used to synchronize agent outputs and maintain contextual consistency.
- [8] Anthropic. 2025a. Announcing claude 4. <https://www.anthropic.com/news/clause-4>.
- [9] Anthropic. 2025b. **Sub agents - claude code documentation**. Accessed: August 20, 2025.
- [10] Anthropic. 2025. Claude code. <https://github.com/anthropics/clause-code>.
- [11] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007a. A survey on context-aware systems. *International journal of Ad Hoc and ubiquitous computing*, 2(4):263–277.
- [12] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007b. **A survey on context-aware systems**. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.
- [13] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. 2019. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):423–443.
- [14] Rishi Bommasani, J. Hudson, E. Adeli, and et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- [15] Florian Le Bronnec, Song Duong, Mathieu Ravaut, Alexandre Allauzen, Nancy F. Chen, Vincent Guigue, Alberto Lumbrieras, Laure Soulier, and Patrick Gallinari. 2024. **Locost: State-space models for long document abstractive summarization**.

- [16] Tom Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- [17] Tian-Yi Che, Xian-Ling Mao, Tian Lan, and Heyan Huang. 2024. A hierarchical context augmentation method to improve retrieval-augmented llms on scientific papers. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD ’24, page 243–254, New York, NY, USA. Association for Computing Machinery.
- [18] Lin Chen and Ying Xu. 2024. Vector-based memory representations for semantic search. In *2024 Conference on Empirical Methods in Natural Language Processing*, pages 1210–1221.
- [19] Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. 2025. Need help? designing proactive ai assistants for programming.
- [20] Zhengbao Chen, Yuxuan Fu, Tianyi Zhang, Chunyuan Lyu, Maosong Sun, and Jindong Wang. 2024a. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.
- [21] Zheyi Chen, Liuchang Xu, Hongting Zheng, Luyao Chen, Amr Tolba, Liang Zhao, Keping Yu, and Hailin Feng. 2024b. Evolution and prospects of foundation models: From large language models to large multimodal models. *Computers, Materials & Continua*, 80(2).
- [22] Yuhong Dai, Jianxun Lian, Yitian Huang, Wei Zhang, Mingyang Zhou, Mingqi Wu, Xing Xie, and Hao Liao. 2025. Pretraining context compressor for large language models. *arXiv*. Compressing long context into short versions while keeping LLMs frozen.
- [23] Zihang Dai, Zhilin Yang, et al. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of ACL*, pages 2978–2988.
- [24] Emilia David. 2024. Anthropic releases model context protocol to standardize ai-data integration. *VentureBeat*. Reports on the launch of the Model Context Protocol (MCP), a system designed to provide a unified interface, thereby mitigating the need for custom, pair-wise adapters in AI-data integration.
- [25] Dbreunig. 2025. How to fix your context. *Blog*.
- [26] Anind K. Dey. 2001a. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7.
- [27] Anind K Dey. 2001b. Understanding and using context. In *Proceedings of the 2001 Workshop on Perceptual User Interfaces*, pages 1–6.
- [28] Mihai-Dumitru Dobre, Florin Pop, and Andrei-Constantin Dobre. 2025. Ten natural language processing tasks with generative artificial intelligence. *MDPI Applied Sciences*, 15(16):9057. This 2025 review discusses LLM applications in NLP, with natural language summarization being a core LLM capability used for enhancing context sharing.
- [29] Hung Du, Srikanth Thudumu, Rajesh Vasa, and Kon Mouzakis. 2025. A survey on context-aware multi-agent systems: Techniques, challenges and future directions.
- [30] Bradley J Edelman, Shuailei Zhang, Gerwin Schalk, Peter Brunner, Gernot Müller-Putz, Cuntai Guan, and Bin He. 2024. Non-invasive brain-computer interfaces: state of the art and trends. *IEEE reviews in biomedical engineering*.
- [31] Facebook. 2013. Rocksdb: A persistent key-value store for flash and ram storage. <https://rocksdb.org>.
- [32] Luciano Floridi and Massimo Chiriatti. 2020. Gpt-3: Its nature, scope, limits, and consequences. *Minds and machines*, 30(4):681–694.
- [33] Lex Fridman. 2025. Transcript for cursor team: Future of programming with ai. *Blog*.
- [34] Google Gemini. 2025. Gemini cli. <https://github.com/google-gemini/gemini-cli>.
- [35] Sanjay Ghemawat and Jeff Dean. 2014. Leveldb. <https://github.com/google/leveldb>.
- [36] Ben Goertzel and Cassio Pennachin. 2021. Artificial general intelligence: Concept, state of the art, and future prospects. *Journal of Artificial General Intelligence*, 12(2):85–109.
- [37] Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces.
- [38] Abhishek Gupta. 2025. Mcp: the universal connector for building smarter, modular ai agents. *InfoQ*. Describes MCP as an open standard, built on JSON-RPC 2.0, that provides a universal client-server protocol for multi-agent systems to share a common toolkit and data representation.

- [39] Yujian Han, Xiaotian Li, Shiji Pan, Xing Xie, and Zhenguo Liu. 2025. **Memos: An operating system for memory-augmented generation in large language models.** *arXiv preprint arXiv:2505.22101*.
- [40] Zifan He, Yingqi Cao, Zongyue Qin, Neha Prakriya, Yizhou Sun, and Jason Cong. 2025. **Hmt: Hierarchical memory transformer for efficient long context language processing.**
- [41] D. Richard Hipp. 2000. Sqlite. <https://www.sqlite.org>.
- [42] Taeho Hwang, Sukmin Cho, Soyeong Jeong, Hoyun Song, SeungYoon Han, and Jong C. Park. 2024. **Exit: Context-aware extractive compression for enhancing retrieval-augmented generation.** *arXiv*. Context-aware extractive compression framework for RAG pipelines.
- [43] Romin Irani. 2025a. **gemini-cli.** <https://github.com/google-gemini/gemini-cli>. GitHub repository, accessed: 2025-09-20.
- [44] Romin Irani. 2025b. **Gemini cli hands-on.** Accessed: 2025-09-20.
- [45] Romin Irani. 2025c. **Gemini cli tutorial series — part 9: Understanding context, memory and conversational branching.** Accessed: 2025-09-20.
- [46] Gautier Izacard and Edouard Grave. 2022. Distilling knowledge from reader to retriever for question answering. *Transactions of the Association for Computational Linguistics*, 10:163–177.
- [47] A. Jacobs. 2025. **Semantic drift a hidden failure mode in llms (working note).** *Reality Drift Working Notes*.
- [48] Andrew Jaegle et al. 2021. **Perceiver: General perception with iterative attention.** *International Conference on Machine Learning (ICML)*.
- [49] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. 1998. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.
- [50] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403*.
- [51] Yizhu Jiao, Sha Li, Sizhe Zhou, Heng Ji, and Jiawei Han. 2024. **Text2DB: Integration-aware information extraction with large language model agents.** In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 185–205, Bangkok, Thailand. Association for Computational Linguistics.
- [52] Zihao Jin, Wancheng Wang, Yubo Chen, Xiaozhi Li, Maosong Sun, Zhiyuan Zhang, and Junchi Tang. 2024. **Long-term memory: The foundation of ai self-evolution.** *arXiv preprint arXiv:2410.15665*.
- [53] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. **Billion-scale similarity search with GPUs.** *IEEE Transactions on Big Data*, 7(3):535–547.
- [54] Bob Kapteijns and Florian Hintz. 2021. **Comparing predictors of sentence self-paced reading times: Syntactic complexity versus transitional probability metrics.** *OSF Preprints*.
- [55] Kari Kostainen, Marco Ambrosin, Abhishta Abhishta, and Pekka Rantala. 2012. **Mobile platform security.** In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 29–36.
- [56] Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. 2010. **A survey of mobile phone sensing.** In *IEEE Communications Magazine*, volume 48, pages 140–150.
- [57] Langroid. 2025. **Langroid.** <https://github.com/langroid/langroid>.
- [58] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Urvashi Khandelwal, Angela Fan, Vishrav Chaudhary, Sainbayar Barta, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.
- [59] Hao Li and Maria Garcia. 2024. High-fidelity memory transformer for long-term context storage. In *2024 Advances in Neural Information Processing Systems*, pages 900–912.
- [60] Zhiyu Li, Shichao Song, and Chenyang Xi. 2025. **Memos: A memory os for ai system.** *arXiv*. Provides memory slicing, tagging, hierarchical mapping, and context binding capabilities.
- [61] Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, and Xiao Tang. 2025. **Open-manus: An open-source framework for building general ai agents.**

References

- [62] Jifan Lin, Xiaojie Cai, Yang Xiao, Yumin Zhuang, Qishuo Hua, Junfei Wang, and Pengfei Liu. 2025. Sii cli: Building next-generation cognitive agentic intelligence ecosystem. <https://github.com/GAIR-NLP/SII-CLI>. Technical Report on Cognitive Agentic Intelligence Framework.
- [63] Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane Luke. 2024. Prompt compression with context-aware sentence encoding for fast and improved llm inference.
- [64] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv*.
- [65] Zhengwu Liu, Jie Mei, Jianshi Tang, Minpeng Xu, Bin Gao, Kun Wang, Sanchuang Ding, Qi Liu, Qi Qin, Weize Chen, et al. 2025. A memristor-based adaptive neuromorphic decoder for brain–computer interfaces. *Nature Electronics*, pages 1–11.
- [66] Zhi Liu and Raj Kumar. 2025. Chatdev++: Scalable multi-agent software development with prompt context embedding. In *Proceedings of the 2025 ACM Conference on Intelligent Agents*.
- [67] LlamaIndex. 2024. Querying with llamaindex. https://docs.llamaindex.ai/en/stable/module_guides/querying/.
- [68] Tengchao Lv, Yupan Huang, Jingye Chen, Yuzhong Zhao, Yilin Jia, Lei Cui, Shuming Ma, Yaoyao Chang, Shaohan Huang, Wenhui Wang, Li Dong, Weiyao Luo, Shaoxiang Wu, Guoxin Wang, Cha Zhang, and Furu Wei. 2024. Kosmos-2.5: A multimodal literate model.
- [69] Manus. 2025. Context engineering for ai agents: Lessons from building manus. *Blog*.
- [70] Yansheng Mao, Yufei Xu, Jiaqi Li, Fanxu Meng, Haotong Yang, Zilong Zheng, Xiyuan Wang, and Muhan Zhang. 2025. Lift: Improving long context understanding of large language models through long input fine-tuning. In *arXiv*. Long-input fine-tuning and gating memory for long-context LLMs.
- [71] Lance Martin. 2025. Context engineering for agents. *Blog*.
- [72] Karl Marx. 1845. Theses on feuerbach. In C. J. Arthur, editor, *The German Ideology*. International Publishers, New York. Original work published 1845.
- [73] Adnan Masood. 2025. Long-context windows in large language models: Applications in comprehension and code. *Medium*. Available at <https://medium.com/@adnanmasood/long-context-windows-in-large-language-models-applications-in-comprehension-and-code->
- [74] Lingrui Mei, Jiayu Yao, et al. 2025. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*.
- [75] Y. Meng, Z. Bing, X. Yao, et al. 2025. Preserving and combining knowledge in robotic lifelong reinforcement learning. *Nature Machine Intelligence*, 7:256–269.
- [76] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. 2008. Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems*, pages 337–350.
- [77] Varun Mohan. 2025. Windsurf discussion. *Blog*.
- [78] Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. 2023. Levels of agi for operationalizing progress on the path to agi. *arXiv preprint arXiv:2311.02462*.
- [79] OpenAI. 2024. Gpt-4 technical report.
- [80] Addy Osmani. 2025. Context engineering tips. *Blog*.
- [81] Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. 2023. Large language models and knowledge graphs: Opportunities and challenges.
- [82] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. 2025. On memory construction and retrieval for personalized conversational agents.

- [83] Aarav Patel and Priya Singh. 2025. Layered memory structures for adaptive ai agents. In *2025 International Conference on Machine Learning*, pages 678–689.
- [84] Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. 2023. *Kosmos-2: Grounding multimodal large language models to the world*.
- [85] Leone Perdigão. 2025. *Context engineering: Architecting ai systems situational awareness*. Medium. Discusses tagging chat messages with metadata for context differentiation.
- [86] Tatiana Petrova, Boris Bliznioukov, Aleksandr Puzikov, and Radu State. 2025. *From semantic web and mas to agentic ai: A unified narrative of the web of agents*.
- [87] philschmid. 2025. *Google gemini cli cheatsheet*. Accessed: 2025-09-20.
- [88] Sahana Vijaya Prasad. 2025. How coderabbit delivers accurate ai code reviews on massive codebases. *Blog*.
- [89] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. 1994. *Human-computer interaction*. Addison-Wesley Longman Ltd.
- [90] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. *Chatdev: Communicative agents for software development*.
- [91] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, and et al. 2025. *UI-TARS: Pioneering automated GUI interaction with native agents*. *arXiv preprint arXiv:2501.12326*.
- [92] Markus N. Rabe and Charles Staats. 2022. *Self-attention does not need $o(n^2)$ memory*.
- [93] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. *Learning transferable visual models from natural language supervision*. *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763.
- [94] Anand Raj. 2025. *Agentic ai and the model context protocol: A new era of autonomous agents*. Medium. Discusses how MCP enables Agentic AI to extend its memory by retrieving information from external knowledge bases, often implemented as vector databases that store context as semantic vectors.
- [95] Stuart Reeves. 2012. Envisioning ubiquitous computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1573–1582.
- [96] langchain Research. 2025. *Context engineering*. *Blog*.
- [97] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. *arXiv preprint arXiv:2102.07350*.
- [98] Moonkyung Ryu, Chih-Wei Hsu, Yinlam Chow, Mohammad Ghavamzadeh, and Craig Boutilier. 2025. *Synthetic dialogue generation for interactive conversational elicitation & recommendation (icer)*.
- [99] Parul Sahu. 2025. *Openai swarm: A hands-on guide to multi-agent systems*. *Analytics Vidhya*. Introduces the OpenAI Swarm framework, which uses JSON structures for function organization and 'context variables' to enable agents to share information and coordinate tasks using a fixed data format.
- [100] Daniel Salber, Anind K Dey, and Gregory D Abowd. 1999. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441. ACM.
- [101] Alireza Salemi, Mihir Parmar, Palash Goyal, Yiwen Song, Jinsung Yoon, Hamed Zamani, Hamid Palangi, and Tomas Pfister. 2025. *Llm-based multi-agent blackboard system for information discovery in data science*.
- [102] Mahadev Satyanarayanan. 2001. *Pervasive computing: Vision and challenges*. In *IEEE Personal Communications*, volume 8, pages 10–17.
- [103] Timo Schick, Jane Dwivedi-Yu, et al. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.
- [104] Bill N. Schilit and Michael M. Theimer. 1994. Disseminating active map information to mobile hosts. In *IEEE Network*, volume 8, pages 22–32.
- [105] ShareDropio. 2025. Sharedrop. <https://github.com/ShareDropio/sharedrop>.

References

- [106] Ben Shneiderman. 1987. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley.
- [107] David Silver, Aja Huang, Chris J Maddison, and et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- [108] John Smith and Wei Zhang. 2025. Daily summarization for long-form llm conversations. In *Proceedings of the 2025 Conference on Natural Language Processing*, pages 101–112.
- [109] Stackademic. 2025. Layered memory architecture for intelligent ai agents: From fast context to deep knowledge. Blog post, Stackademic, published 2 months ago.
- [110] Haoran Sun and Shaoning Zeng. 2025. Hierarchical memory for high-efficiency long-term reasoning in llm agents.
- [111] Jie Sun, Tianyu Zhang, Houcheng Jiang, Kexin Huang, Chi Luo, Junkang Wu, Jiancan Wu, An Zhang, and Xiang Wang. 2024. Large language models empower personalized valuation in auction.
- [112] Ruiming Tang, Chenxu Zhu, Bo Chen, Weipeng Zhang, Menghui Zhu, Xinyi Dai, and Huifeng Guo. 2025. Llm4tag: Automatic tagging system for information retrieval via large language models. *arXiv*. Uses graph-based and knowledge-enhanced tag generation with LLMs.
- [113] Xin Tang, Hao Shen, Siyuan Zhao, Na Li, and Jia Liu. 2023. Flexible brain–computer interfaces. *Nature Electronics*, 6(2):109–118.
- [114] Anthropic Team. 2025a. Effective context engineering for ai agents. *Blog*.
- [115] Anthropic Team. 2025b. How we built our multi-agent research system. *Blog*.
- [116] AutoGPT Team. 2025c. Autogpt. <https://github.com/Significant-Gravitas/AutoGPT>.
- [117] DataCamp Team. 2025d. The 7 best vector databases in 2025. *DataCamp Blog*. Lists the top vector databases of 2025, which are the foundational infrastructure for representing context as semantic vectors and enabling efficient retrieval in AI systems.
- [118] Descope Team. 2025e. What is the model context protocol (mcp) and how it works. *Descope Blog*. Explains how MCP standardizes tool connectivity for AI systems by defining a consistent set of specifications, serving as a “universal remote” for AI and establishing a shared representation.
- [119] Letta Team. 2024. Letta (formerly memgpt). <https://github.com/letta-ai/letta>. GitHub repository.
- [120] OpenSII Team. 2025f. Opensii. <https://www.opensii.ai>.
- [121] OpenSII Team. 2025g. Opensii. <https://www.opensii.ai/cli/docs>.
- [122] Tongyi DeepResearch Team. 2025h. Tongyi-deepresearch. <https://github.com/Alibaba-NLP/DeepResearch>.
- [123] Verloop.io Team. 2025i. Natural language processing in chatbots 2025. *Verloop.io Blog*. Emphasizes how 2025 NLP chatbots achieve “contextual engagement” by retaining and understanding context, which can be achieved by exchanging human-readable summaries.
- [124] Huy-Tan Thai, Kim-Hung Le, and Ngan Luu-Thuy Nguyen. 2023. Formerleaf: An efficient vision transformer for cassava leaf disease detection. *Computers and Electronics in Agriculture*, 204:107518.
- [125] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.
- [126] Fei Wang, Yuewen Zheng, Qin Li, Jingyi Wu, Pengfei Li, and Luxia Zhang. 2024a. Chatschema: A pipeline of extracting structured information with large multimodal models based on schema.
- [127] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024b. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution.
- [128] Jason Wei et al. 2022. Chain of thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

References

- [129] Mark Weiser. 1991. The computer for the 21st century. *Scientific American*, 265(3):94–104.
- [130] Zhihao Wen, Sheng Liang, Yaxiong Wu, Yongyue Zhang, and Yong Liu. 2025. Effective and efficient schema-aware information extraction using on-device large language models.
- [131] Rebecca Westhäuser, Frederik Berenz, Wolfgang Minker, and Sebastian Zepf. 2025. Caim: Development and evaluation of a cognitive ai memory. *arXiv*. Integrated tagging system and contextual filtering for memory recall.
- [132] Dongrui Wu, Bao-Liang Lu, Bin Hu, and Zhigang Zeng. 2023. Affective brain–computer interfaces (abcis): A tutorial. *Proceedings of the IEEE*, 111(10):1314–1332.
- [133] Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. 2025. Resum: Unlocking long-horizon search intelligence via context summarization.
- [134] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers.
- [135] Yang Xiao, Mohan Jiang, Jie Sun, Keyu Li, Jifan Lin, Yumin Zhuang, Ji Zeng, Shijie Xia, Qishuo Hua, Xuefeng Li, Xiaojie Cai, Tongyu Wang, Yue Zhang, Liming Liu, Xia Wu, Jinlong Hou, Yuan Cheng, Wenjie Li, Xiang Wang, Dequan Wang, and Pengfei Liu. 2025. Limi: Less is more for agency.
- [136] Yue Xing, Tao Yang, Yijiashun Qi, Minggu Wei, Yu Cheng, and Honghui Xin. 2025. Structured memory mechanisms for stable context representation in large language models.
- [137] Kelin Xu, Yichen Zhang, Yujia Bai, Xiangning Lin, Jinchao Chen, Quoc V. Le, and Denny Zhou. 2025. Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.
- [138] Wenli Yang, Yuchen Wei, Hanyu Wei, Yanyu Chen, Guan Huang, Xiang Li, Renjie Li, Naimeng Yao, Xinyi Wang, Xiaotong Gu, et al. 2023. Survey on explainable ai: From approaches, limitations and applications aspects. *Human-Centric Intelligent Systems*, 3(3):161–188.
- [139] Shinn Yao et al. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- [140] Shunyu Yao, Dian Yu, and Jeffrey Zhao. 2025. Tree of thoughts: Deliberate problem solving with large language models. In *ICLR*.
- [141] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.
- [142] Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Rich James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2023. Retrieval-augmented multimodal language modeling.
- [143] Lyumanshan Ye, Xiaojie Cai, Xinkai Wang, Junfei Wang, Xiangkun Hu, Jiadi Su, Yang Nan, Sihan Wang, Bohan Zhang, Xiaoze Fan, Jinbin Luo, Yuxiang Zheng, Tianze Xu, Dayuan Fu, Yunze Wu, Pengrui Lu, Zengzhi Wang, Yiwei Qin, Zhen Huang, Yan Ma, Zhulin Hu, Haoyang Zou, Tiantian Mi, Yixin Ye, Ethan Chern, and Pengfei Liu. 2025a. Interaction as intelligence: Deep research with human-ai partnership.
- [144] Ye Ye. 2025. Task memory engine: Spatial memory for robust multi-step llm agents. *arXiv preprint arXiv:2505.19436*.
- [145] Zhifan Ye, Kejing Xia, Yonggan Fu, Xin Dong, Jihoon Hong, Xiangchi Yuan, Shizhe Diao, Jan Kautz, Pavlo Molchanov, and Yingyan Celine Lin. 2025b. Longmamba: Enhancing mamba’s long context capabilities via training-free receptive field enlargement.
- [146] Seonghyun Yun et al. 2025. Memory-r1: Memory reinforcement learning for large language models. *Blog*.
- [147] Atefeh Zarabzadeh, Niloofar Montazeri, and Aida Zolfaghari. 2024. A comprehensive survey on context-aware multi-agent systems: Techniques, applications, challenges and future directions. *CoRR*, abs/2402.01968. This 2024 survey discusses challenges in Context-Aware Multi-Agent Systems (CA-MAS), where heterogeneity often requires adaptive/adapter mechanisms for context acquisition and abstraction.
- [148] Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.
- [149] Rui Zhao and Sun Kim. 2025. Hierarchical memory transformer for context-aware ai. In *2025 Conference on Neural Information Processing*, pages 1123–1135.

-
- [150] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K Qiu, and Lili Qiu. 2024. Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely. *arXiv preprint arXiv:2409.14924*.
 - [151] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. [Lifelongagentbench: Evaluating llm agents as lifelong learners](#).