Treating LLM as the CPU: Toward a General-Purpose Agentic Computing Specification and Operating System Architecture

Lidongdong

April 15, 2026

Abstract

Current application paradigms for Large Language Models (LLMs) remain confined to simplistic "wrapper" encapsulations, lacking system-level memory management, standardized communication protocols, and auxiliary intelligence frameworks. This paper analogizes the LLM to the "CPU" of a novel computing architecture and, from this perspective, develops a vision for an "Agent Operating System" (Agent-OS) and a general-purpose agentic computing specification. We propose a comprehensive analogical framework: virtual context management mechanisms (MemGPT/Letta), memory layer architectures (Mem0), and five-layer self-learning memory systems (Hermes Agent) constitute the "RAM/Disk" subsystem; the Model Context Protocol (MCP) and Agent-to-Agent (A2A) communication protocols form the "Bus"; tool APIs serve as "Peripherals"; and application frameworks such as OpenClaw's gateway architecture and Hermes Agent's self-evolutionary learning loop function as "Applications." At the deployment level, we analyze the cloud-edge-device three-tier collaborative computing paradigm, elucidate the technological pathways and industry practices for "on-device personal agents," and envision the interconnection prospects of the "Internet of Agents." Furthermore, we explore the self-evolution capabilities of AI agents—from experiential memory accumulation to self-optimization, and ultimately to meta-cognitive cross-domain self-improvement—revealing a paradigm shift from "fixed programs" to "living code." Finally, we outline the standardization framework required for a general-purpose intelligent application specification, encompassing a unified memory file system, a standardized bus protocol stack, and an AI operating system kernel with SDK specifications, thereby providing a conceptual framework and architectural blueprint for systematic research in this domain.

Keywords: Large Language Model · Agent Operating System · Model Context Protocol · Agent-to-Agent Communication · Cloud-Edge-Device Collaboration · Internet of Agents · Self-Evolution

1 Introduction

Recent years have witnessed breakthrough advancements in artificial intelligence technologies centered on Large Language Models. From GPT to Claude and Gemini, these models exhibit remarkable language understanding and generation capabilities, embodying the "brain" of intelligence. However, just as the CPU was not the entirety of the personal computer during the 1980s computing revolution—it required an operating system,

memory scheduling, file systems, task management, and user interfaces—in the current AI wave, the LLM is merely the "kernel" of an "AI operating system." What truly transforms the world is the system-level intelligent architecture constructed around the model.

Current LLM application paradigms exhibit evident systemic deficiencies. Most interactions are stateless; models lack persistent cross-session memory capabilities. Integration with external tools employs point-to-point methods lacking standardized communication protocols, resulting in complex and inefficient integration. There is no unified platform to orchestrate multiple agents, manage resources (e.g., API call quotas, token consumption), or ensure their secure, stable, and collaborative operation. As scholars have noted, today's agent architectures resemble the pre-OS computing era—replete with repetitive solutions and lacking fundamental abstractions for resource management, isolation, and coordination.

Addressing these deficiencies, this paper advances a core thesis: treat the LLM as the "CPU" of a novel computing architecture and, through this lens, re-examine and redesign the entire intelligent computing stack. This perspective is not a mere metaphor but a set of technical propositions: the mode of human-machine interaction will shift from traditional GUIs to natural language and multimodal understanding; agents will be scheduled, coordinated, and persisted akin to processes; the minimal resource unit will shift from "file/thread" to "intent/task"; and the operating system kernel's logic will transition from "scheduling the CPU" to "scheduling intelligence."

The contributions of this paper are as follows:

- Conceptual Framework: Proposes the "LLM-as-CPU" systematic analogical framework, mapping the LLM to the CPU of traditional computing architectures and systematically identifying the missing corresponding components, including memory/storage, bus, peripherals, and operating system.
- Architectural Blueprint: Designs a five-layer Agent-OS architecture comprising kernel, services, runtime, orchestration, and user layers, providing systematic architectural reference for agent operating systems.
- Protocol Ecosystem Analysis: Systematically classifies and functionally positions agent communication protocols such as MCP, A2A, and AONP, analyzing their roles and complementary relationships within the agentic computing stack.
- Deployment Paradigm Elaboration: Expounds the cloud-edge-device three-tier collaborative computing paradigm, concretizing the concept of "interconnected on-device personal agents" into architectural design principles, substantiated by recent industry practices.
- Self-Evolution Elaboration: Systematically articulates the paradigm shift of AI agents from "fixed programs" to "living code," analyzing three levels of self-evolution mechanisms: experiential memory, self-optimization, and meta-cognition.
- Standardization Roadmap: Proposes three pillars for advancing a general-purpose intelligent application specification—unified file system, standardized bus protocol, and OS kernel with SDK—providing a reference framework for industry standardization.

2 From von Neumann to Intent-Driven: The "LLM-as-CPU" Systematic Analogy

The traditional von Neumann architecture centers on stored programs, where the CPU executes computational tasks according to predefined instruction sequences. In the new LLM-centric computing paradigm, the architecture follows an intent-driven logic—the system receives intents expressed in natural language, decomposes them into task graphs, and dynamically schedules resources for execution.

2.1 Core Component Analogy

This section systematically maps each core component of the traditional PC architecture to the LLM-centric intelligent computing system:

Traditional PC Component	Analog in LLM Intelligent System	Technical Implementation and Frontier Exploration
CPU	LLM Kernel	Projects like AIOS treat the LLM as the "kernel" of an "AI operating system," responsible for intent understanding, task decomposition, and decision-making—the source of intelligence.
RAM	Context Window	MemGPT/Letta innovatively propose virtual context management, treating the limited context window as "main memory" and employing paging mechanisms for dynamic information loading.
Storage	Vector DB / Knowledge Graph / Memory Layer	Mem0 provides production-grade memory infrastructure supporting cross-model, cross-framework memory layers; Hermes Agent constructs a five-layer self-learning memory architecture for persistent, evolvable long-term memory.
Bus	Standardized Communication Protocols	MCP serves as a "USB" interface connecting LLMs to external tools and data; A2A protocol supports agent discovery, negotiation, and collaboration.
I/O Devices	Tools / APIs	Via standard protocols, LLMs can invoke search engines, calculators, code interpreters, email systems, and any external tool with an API.
OS	Agent-OS	Projects like AIOS, Agent OS, and Synoetic OS provide system-level services such as agent scheduling, resource management, security isolation, and lifecycle management.
Applications	Agent Application Frameworks	OpenClaw provides a gateway architecture to uniformly orchestrate all communications and task execution; Hermes Agent embeds a self-evolutionary learning loop, becoming more attuned to the user over time.

2.2 Agents: The Fundamental Unit of Novel Computing

Within the "LLM-as-CPU" architecture, the Agent serves as the fundamental execution unit of novel computing, analogous to a "process" in a traditional operating system. A complete agent comprises the following core elements:

- Identity and Goal: The agent's unique identifier and its pre-defined mission or task objective.
- Reasoning Engine (LLM): The "brain" responsible for intent understanding, planning, and decision-making.
- Memory System: Includes working memory (short-term context) and long-term memory (persistent knowledge).

- Toolset: The collection of external capabilities the agent can invoke, integrated via standard protocols like MCP.
- Communication Interface: Standardized communication capabilities adhering to protocols like A2A, enabling collaboration with other agents.

Agents communicate and collaborate via standardized protocols, akin to how network nodes communicate via TCP/IP. This "Agent-as-a-Service" paradigm will fundamentally redefine the nature of software—future software will no longer be static code blocks but dynamic, autonomous intelligent entities.

2.3 OpenClaw and Hermes Agent: Two Representative Agent Application Frameworks

Within the current agent ecosystem, OpenClaw and Hermes Agent represent two distinctive design philosophies for application frameworks, exploring different approaches to translating LLM capabilities into executable agent systems.

OpenClaw (colloquially known as "Lobster") was developed by Austrian programmer Peter Steinberger and released in November 2025 as an open-source agent system. Its logo, a lobster, symbolizes molting and continuous evolution. The framework is not a single application but an AI assistant capable of autonomously performing file operations, browser automation, data scraping, spreadsheet creation, and more. Its core architecture is built around a Gateway that serves as a central hub uniformly orchestrating all communications and task execution. With sufficient system permissions granted by the user, it can drive large models to operate the computer, invoke tools like Feishu, and execute multi-step tasks. OpenClaw's core components include Channels, Agents, and Skills, enabling connection to various messaging platforms and local tools, much like an efficient AI dispatch center. Within just over four months of release, it garnered approximately 285,000 stars, becoming one of the highest-starred open-source projects in history. However, the framework also presents security concerns—it possesses the capability for proactive local data collection and may automatically read sensitive terminal information without explicit user authorization, necessitating a balance between security and capability in permission settings.

Hermes Agent was open-sourced by the renowned AI lab Nous Research in late February 2026, surpassing 66,000 stars within two months. Its core positioning is as a self-evolving "long-term companion," with a key breakthrough being an embedded closed-loop learning mechanism that enables autonomous creation, refinement, and persistence of skills during task execution, achieving "self-evolution." Unlike OpenClaw, which relies on modifying configuration files and orchestrating multiple agents for tasks, Hermes Agent employs a five-layer architecture centered on a self-growth closed loop of "user interaction → behavior recording → outcome evaluation → strategy optimization → skill consolidation." Its memory architecture is divided into five layers: short-term reasoning memory, conversational memory, skill memory, preference memory, and meta-memory, with all learning outcomes and memories stored locally in SQLite databases. It incorporates multiple layers of security measures, including dangerous command approval and container isolation, defaulting to a more cautious stance. Hermes Agent supports six deployment

methods—ranging from $5 VPS to Docker and serverless—and is compatible with over 200 LLMs for one-click switching, achieving full-platform accessibility.

The fundamental distinction lies in the positioning of "memory": OpenClaw emphasizes operation and execution, resembling an "intern assistant" adept at operating computers and invoking APIs; Hermes Agent, by contrast, possesses persistent memory and evolutionary capability, more akin to a "dedicated assistant" that remembers the past and continuously refines its performance. As industry commentary observes, OpenClaw's memory mechanism is transitioning from "storage" to "cognition," with its agent architecture evolving toward an "operating system form."

3 Memory and Storage Subsystem: From Stateless to Memorable Agents

One of the most conspicuous deficiencies of current LLM applications is the lack of persistent memory. Each interaction is essentially stateless; the model cannot recall previous conversations, much less accumulate long-term knowledge and experience. This is akin to a CPU being completely reset after every task, unable to form sustained computational capability. Addressing this issue hinges on constructing a system-level memory management architecture.

3.1 Hierarchical Memory Architecture: From MemGPT to Mem0

The MemGPT project draws upon the core ideas of virtual memory management in traditional operating systems to propose a hierarchical memory system. Rather than attempting to infinitely expand the LLM's "physical memory" (i.e., the actual context window), it gives the LLM the illusion of "virtual memory," enabling intelligent information paging between fast but limited "main context" (analogous to RAM) and slower but vast "external context" (analogous to disk). MemGPT's system divides memory into distinct tiers: core memory (an always-accessible compressed representation), recall memory (a searchable database), and archival memory (long-term storage of important information). This hierarchical structure conceptually corresponds to the L1/L2 cache, main memory, and disk storage hierarchy of computers. To transition MemGPT from academic research to enterprise-grade solutions, its creators, Packer and Wooders, co-founded Letta. Letta further systematized memory management by introducing Memory Blocks mechanisms, enabling agents to actively manage information blocks within their core memory.

In contrast to MemGPT's self-management approach, Mem0 offers a complementary memory architecture. Mem0 is a scalable, memory-centric architecture that addresses long-term memory for AI agents by dynamically extracting, consolidating, and retrieving key information from ongoing conversations. The architecture employs a "passive extraction + semantic search" approach, automatically extracting user preferences, facts, and context from dialogues and storing them in vector databases to support cross-session personalized interactions. Mem0 is positioned as a model-agnostic "memory layer"—a memory layer that works across all models, all frameworks, and all platforms. Its API call volume surged from approximately 35 million in Q1 2025 to 186 million in Q3, demonstrating strong market

demand for standardized memory solutions. In October 2025, Mem0 announced $24 million in funding to build AI's memory layer infrastructure.

3.2 Self-Managed Memory and Strategic Forgetting

MemGPT's most innovative feature is its use of the LLM itself as a memory manager. Through self-directed memory editing and tool invocations, the system can actively manage its own memory contents, deciding what to store, summarize, or forget. This capability represents a significant departure from traditional AI systems where memory management is typically handled by external, rule-based mechanisms.

Notably, MemGPT treats information "forgetting" as a necessary function rather than a failure. In traditional information systems, preservation is paramount, and deletion implies data loss. MemGPT implements "strategic forgetting" through two key mechanisms—summarization and targeted deletion. This approach signifies a fundamental shift in AI system information management paradigms: traditional Retrieval-Augmented Generation (RAG) systems aim to maximize recall, whereas MemGPT prioritizes precision and relevance. Letta's benchmark tests further validate the effectiveness of this approach, demonstrating that file-system-like memory architectures offer significant advantages for agent long-term memory management.

3.3 Toward a Unified "File System" Vision

For the memory subsystem to become truly portable and interoperable, we need a set of standards governing how agent "memories" are organized and accessed, analogous to file systems (e.g., NTFS, ext4) on PCs. The elements of such a standard should include:

1. Data Format: Define unified representation formats for storing conversation flows, user profiles, knowledge graphs, skill libraries, and other structured and unstructured data.

2. Access Interface: Provide standard APIs enabling all agents to read and write memories uniformly, regardless of whether the underlying storage is a vector database or a graph database.

3. Ownership and Portability: Users should own their memory data and be able to export their "digital memory archive" and migrate it from one AI system to another, much like exporting a file.

4. Privacy Classification Standards: Define storage locations (local/cloud) and access control policies for memory data of varying sensitivity levels.

Mem0's architectural philosophy moves precisely in this direction: a unified memory layer usable across different models and frameworks, ensuring memory is not locked into a specific AI implementation. Such a memory file system will render the user's AI assistant truly "personal"—by simply importing their memory archive, any compatible AI system can instantaneously "inherit" the user's preferences, habits, and knowledge system, achieving cross-platform portability of the digital persona.

4 Communication Protocol Stack: Standardizing the Agent "Bus"

If the memory subsystem constitutes the "storage" of agentic computing architecture, then communication protocols form the "bus"—defining how agents exchange information and collaborate with each other and the external world. Currently, three key protocols are shaping a complementary protocol stack:

4.1 MCP: The Model-Tool "USB" Bus

The Model Context Protocol (MCP) was developed and released by Anthropic in November 2024 to standardize how applications provide context and tools to LLMs. It can be viewed as the HTTP for AI models—a standardized protocol enabling AI models to "plug-and-play" access data sources and tools. MCP's core idea is to define a uniform "socket" standard for AI models to access external resources: developers can write an MCP-compliant "server" for any tool (from local command lines to cloud services), and Claude (acting as the "client") communicates with it via this standard interface.

MCP follows a client-server architecture with core components including: MCP Host (the AI application seeking data access), MCP Client (protocol client maintaining a 1:1 connection with the server), MCP Server (lightweight program exposing specific capabilities), and local data sources and remote services. Since its release, MCP has been rapidly adopted by the AI industry, with widespread use by companies including OpenAI, Google, Microsoft, and Cursor.

In early 2026, Anthropic significantly upgraded MCP, introducing the MCP Tool Search feature, fundamentally altering how tools are loaded. Previously, Claude Code needed to preload documentation for all available tools, consuming substantial context window capacity—"an MCP server might expose over 50 tools, and users often run multiple servers simultaneously, with documented setups consuming 67k+ tokens." The new Tool Search employs a lazy loading mechanism, automatically switching to a lightweight search index when tool descriptions exceed 10% of the available context. Internal testing showed token usage dropping from approximately 134k to around 5k—an 85% reduction. Additionally, community benchmarks indicate MCP evaluation accuracy improved from 49% to 74% (Opus 4) and from 79.5% to 88.1% (Opus 4.5).

In 2026, MCP maintainers formulated an annual roadmap identifying four priority development areas: transport evolution and scalability, agent communication, governance maturation, and enterprise readiness, aimed at addressing challenges in real production deployments. In April 2026, the North American MCP Developer Summit was held in New York, attracting approximately 1,200 attendees and becoming a flagship event for the MCP ecosystem. MCP addresses the vertical integration of agents and tools—how a single agent efficiently and securely invokes external capabilities.

4.2 A2A: The Agent-Agent "TCP/IP"

The Agent-to-Agent (A2A) protocol was announced as open-source by Google at the Google Cloud Next conference on April 10, 2025, as the first standard agent interaction protocol. It is an open, vendor-neutral standard language enabling independent AI agents to discover

each other, negotiate communication modalities (text, files, streaming), and collaborate without exposing their private code or data.

On June 24, 2025, Google Cloud donated the A2A protocol to the Linux Foundation, with tech giants including Amazon, Microsoft, and Cisco joining the support ranks. This protocol is significant, akin to creating a "universal language" and unified "communication standard" for diverse agents. Before its emergence, while multi-agent collaboration was common, agents built on different frameworks could not easily exchange tasks and states, hindering commercial deployment.

A2A defines a standardized interaction flow: each agent publishes an Agent Card listing its name, endpoint, skills, and supported authentication processes; OAuth 2.0/OIDC with short-lived tokens enables fine-grained machine identity management; tasks are exchanged via JSON-RPC 2.0 over HTTPS, supporting both synchronous invocation and asynchronous streaming; built-in OpenTelemetry support ensures each request/response carries trace IDs and structured logs. Okta's collaboration with Google Cloud further elevated agent-to-agent authentication standards, with both parties jointly defining A2A authentication specifications and building SDKs.

A2A is closely related to MCP; both are industry complementary standards and communication protocols for building and enhancing agent applications, and both are based on JSON-RPC 2.0. However, they have distinct roles: MCP focuses on connecting agents to external tools, APIs, and resources through structured input/output; A2A primarily addresses agent-to-agent interaction and communication. In essence, MCP excels at enabling an Agent to work with external tools, while A2A excels at enabling multiple Agents to collaborate seamlessly—the two complement each other.

A2A addresses the horizontal collaboration problem among agents—how multiple agents discover each other, negotiate tasks, and collaboratively achieve complex objectives.

4.3 AONP: The "Network Protocol Suite" for the Internet of Agents

On a global scale, the China Mobile Research Institute released the Agent Internet Open Network Protocol (AONP) framework and Agent Gateway at the Mobile World Congress in March 2026. This release marks a new phase in moving agent interconnection from concept to standardization and openness.

China Mobile, from the perspective of a network operator, pioneered the concept and architecture of the Internet of Agents, recognizing that "hundreds of millions of agents are emerging, interacting and collaborating via the Internet, accelerating the formation of 'collective intelligence.'" Previously, China Mobile had introduced the definition of "agent" into 3GPP and incorporated it into requirement standards, proposed the Agent Communication Network (ACN) architecture, and released a prototype. Within the IETF, China Mobile led two agent interconnection protocol standards workshops and served as a core supporting unit for the approval of the first Agent Interconnection Protocol BoF coordination meeting (officially held at IETF 125 in March 2026).

AONP is a protocol suite framework comprising five core protocols:

1. Semantic Routing and Multicast Protocol (SRMP): Supports agent intent resolution, dynamically matches computing-network resource requirements, supports multi-agent group communication, and enables intelligent forwarding and routing optimization.

2. Multimodal Transport Protocol (M2TP): Introduces fragmentation transmission and relay acceleration capabilities, supports conversion among TCP, QUIC, MoQ, and other underlying transport protocols, enhancing efficiency and stability of end-to-end multimodal data transmission.

3. Agent and Tool Cross-Domain Discovery Protocol (AIDP): Extends DNS resource record definitions to support dynamic service discovery, resolution, and address mapping, enabling fast and accurate matching of agents and tools across domains.

4. Session Management and Invocation Protocol (SMIP): Optimizes end-to-end state management, refines session state machine design for asynchronous invocation scenarios, ensuring continuity and reliability of agent interactions.

5. Agent Authorization Protocol (A2P): Decouples authorization token management from the client side, supporting batch request authorization, token caching, and fine-grained permission management.

China Mobile also independently developed the Agent Gateway (AGW) prototype, the core carrier for implementing the AONP protocol framework, achieving key AONP capabilities through centralized and distributed cooperative control. Concurrently, international standards organizations including IETF, 3GPP, ITU-T, and CCSA have initiated standardization discussions on agent communication protocols.

AONP's unique value lies in addressing the interconnection challenges of large-scale agent deployment from the network infrastructure layer—how to achieve efficient routing, discovery, and communication when thousands or millions of agents are distributed across the Internet.

4.4 Functional Layering of the Protocol Stack

The aforementioned protocols are not competitive but rather constitute a complete protocol stack from the underlying network to the application layer:

Protocol	Analogous Layer	Core Function	Standards Body
AONP	Network/Transport Layer	Agent routing, multimodal transport, cross-domain discovery, session management, network-level authorization	IETF/3GPP/CCSA
A2A	Session/Presentation Layer	Agent-to-agent discovery, negotiation, task exchange, streaming	Linux Foundation

Protocol	Analogous Layer	Core Function	Standards Body
		communication	
MCP	Application Layer	Model-tool connection, context injection, structured output	Anthropic

MCP, A2A, and AONP address communication requirements at different levels of the agentic computing stack. MCP focuses on how a single agent accesses tools and data, A2A focuses on how multiple agents collaborate, and AONP focuses on how vast numbers of agents interconnect at Internet scale. The synergy of these three will provide a complete protocol infrastructure for building the Internet of Agents.

5 Agent-OS: The Architectural Framework of Agent Operating Systems

5.1 Research Progress on AIOS and Agent-OS

Researchers at Rutgers University created AIOS—an LLM Agent Operating System (AgentOS)—providing module isolation and aggregation of LLM and OS functionalities. To address potential conflicts between LLM-related and non-LLM tasks, they designed an LLM-specific kernel. This kernel separates OS-like responsibilities (particularly those related to LLM agent supervision) from corresponding resources and development toolkits. Through this separation, the LLM kernel aims to enhance the management and coordination of LLM-related activities. AIOS organizes agent applications and resources (such as LLMs and tools) into distinct layers; experiments demonstrate that AIOS can reduce latency for Mistral and Llama models by nearly twofold when running thousands of agents.

In industry, DingTalk released the world's first AI-native work intelligence operating system, Agent OS, in December 2025, heralding a new era of "human-AI collaboration." Agent OS is designed around multi-agent collaboration, providing a unified, secure framework for AI execution within complex enterprise environments.

At a more systemic level, academia has proposed a blueprint architecture for Agent-OS. This blueprint notes that current large-model agent systems remain ad-hoc pipelines lacking OS-level guarantees—deficient in scheduling, memory, real-time responsiveness, and end-to-end security. Today's agent architectures resemble the pre-OS computing era—replete with repetitive solutions and lacking fundamental abstractions for resource management, isolation, and coordination. Existing frameworks (e.g., tool invocation, MCP, A2A messaging) address isolated issues but lack a unified, security-first, latency-aware foundation.

Synoetic OS v1.0 approaches from another angle, treating narrative coherence as a kernel primitive to achieve substrate-independent orchestration of AI agents across heterogeneous cloud providers. Validated through 173 days of continuous production operation (June 12, 2025, to December 3, 2025), it demonstrates the practical feasibility of this concept.

The Eclipse Foundation also launched the LMOS platform, the first enterprise-oriented open Agent Definition Language (ADL), including the JVM-native ARC Agent framework, providing a Kotlin runtime for developing, testing, and extending AI agents, with a built-in visual interface supporting rapid iteration and debugging.

5.2 Five-Layer Architecture Design

Agent-OS is defined as an abstract layered architecture comprising five core layers:

- Kernel Layer: Responsible for agent lifecycle management, resource allocation, security isolation, and low-level scheduling. The kernel layer distinguishes between LLM-related and non-LLM tasks and enforces fine-grained access control over agents, resources, and toolkits.
- Services Layer: Provides cross-agent common services, including Memory Service (context persistence and retrieval), Tool Service (registration and invocation of MCP servers), Identity Service (authentication and authorization of agents and users), and Observability Service (logs, metrics, and traces).
- Agent Runtime Layer: Provides an isolated execution environment for each agent, managing its context window (RAM), long-term memory storage (Disk), and tool invocation capabilities. Analogous to the process virtual address space in a traditional OS.
- Orchestration Layer: Responsible for the decomposition, scheduling, and coordination of multi-agent tasks. Maps user intents into task graphs and assigns subtasks to the most suitable agents for execution.
- User Layer: Provides a unified interaction interface, including natural language input, multimodal perception, and generative UI output. The User Layer is the boundary between Agent-OS and traditional user interaction.

Additionally, security, governance, and observability serve as **cross-cutting concerns spanning all layers.**

5.3 Agent Contract: Portability Specification for Agents

To achieve portability of agents across different Agent-OS implementations, researchers have proposed the concept of an Agent Contract. An Agent Contract is a formal specification defining an agent's capability interface, resource requirements, security constraints, and behavioral expectations. Its core elements include:

- Capability Description: Types of tasks the agent can perform, required input formats, and expected output formats.
- Resource Requirements: Including token budget, memory footprint, API quotas, and latency requirements.
- Security Constraints: Defining the agent's permission boundaries, data access scope, and isolation requirements.
- Service Level Agreement (SLA): Defining the agent's availability, response time, and reliability commitments.

Agent Contracts enable agents to be distributed and deployed across different platforms like "executable files"—as long as a platform supports the capability interface defined in the Contract, the agent can run seamlessly. This is analogous to the concept of Docker containerization: packaging an application and its dependencies into a portable image.

6 Deployment Paradigm: Cloud-Edge-Device Collaboration and the Internet of Agents

Under the "LLM-as-CPU" architecture, agent deployment is not a singular cloud-centric model but forms a continuum from device to cloud. "On-device personal agents" on terminal devices collaborate with edge and cloud layers, collectively constituting the infrastructure of the Internet of Agents.

6.1 On-Device Personal Agents: Technological Pathways for Device-Side Deployment

"On-device personal agents" refer to lightweight agents running on terminals such as smartphones and wearables, capable of providing low-latency, high-privacy AI services in offline or weak-network environments. This vision is rapidly materializing.

Chip-Level Breakthroughs. At the 2026 Mobile World Congress, Qualcomm introduced the new Snapdragon Wear Elite Platform, the industry's first NPU-enabled wearable platform capable of high-performance on-device AI processing. By integrating the Qualcomm Hexagon NPU, the platform supports models up to billions of parameters at the edge and, combined with advanced sensor fusion, high-performance low-power connectivity, and computation, enables a new class of personal AI experiences, including context-aware recommendations, natural voice interaction, life logging, and AI agents that can execute tasks and orchestrate activities on behalf of the user. The key to this experience is "on-body" computation—on-device models eliminate the need to transmit data wirelessly to a phone or cloud for inference, mitigating privacy leakage risks and enabling wearables to operate independently in more scenarios. The platform introduces a pioneering multi-mode connectivity architecture integrating six advanced technologies: 5G RedCap, Micro-Power Wi-Fi, Bluetooth 6.0, Ultra-Wideband, GNSS, and NB-NTN.

In chip architecture innovation, Shanghai Jiao Tong University and Huixi jointly proposed the ROMA (Read-Only-Memory-based Accelerator) architecture, providing a systematic solution for deploying QLoRA-quantized on-device LLMs. ROMA employs a hybrid storage architecture, using ROM for quantized base models and SRAM for LoRA weights and KV caches, significantly improving inference energy efficiency. VeriSilicon also introduced an ultra-low-power NPU delivering over 40 TOPS of on-device LLM inference compute for mobile applications, specifically addressing the growing demand for generative AI on mobile platforms.

System Architecture Level. The typical architecture of an on-device AI phone/PC agent system comprises five layers: hardware layer (NPU/GPU/CPU/security chip), system layer (driver interface/HAL/memory management/TEE), model layer (quantized models/inference engine/model cache), Agent framework layer (intent recognition/task

planning/tool invocation/context management), and application layer. This architecture highlights NPU/GPU hardware acceleration, large-small model collaborative inference, local knowledge bases (RAG), and privacy security closure (TEE), achieving automated cross-application (calendar, email, etc.) operations and user interaction through intent recognition and task planning.

SoulMate is a mobile intelligence system-on-chip (SoC) designed for on-device LLM personalization, operating at 9.8mW and integrating retrieval-augmented generation (RAG) and personal LLM fine-tuning capabilities, demonstrating the feasibility of fully autonomous on-device agents.

6.2 Cloud-Edge-Device Three-Tier Collaborative Architecture

Relying solely on device-side or cloud-side deployments is insufficient to support complex agent applications. Cloud deployment introduces three main issues: dependence on cloud computing hampers the rapid model inference required for real-time applications; transmitting large volumes of data incurs substantial bandwidth consumption; and cloud-based LLMs raise significant privacy concerns when handling sensitive data. Consequently, a cloud-edge-device three-tier collaborative architecture has emerged.

ZTE Vice President Hu Junjie noted that AI terminal proliferation faces three major constraints: high cost and power consumption of high-performance hardware, latency and privacy risks of pure cloud deployment, and functional siloing bottlenecks of traditional architectures. Against this backdrop, the device-edge-cloud collaborative architecture becomes a key breakthrough: cloud-based large models provide global orchestration, edge-side small-sample algorithms precisely adapt to regional scenarios, and terminal lightweight models ensure low latency and privacy security. Hu cited forecasts indicating that the proportion of generative AI deployed on edge devices will surge from 5% in 2023 to 60% by 2029, with the corresponding market size skyrocketing from $27 billion in 2024 to $269.8 billion in 2032.

Geling Shentong's AI Edge Studio platform exemplifies a concrete implementation of the three-tier architecture: device-side access layer, edge computing node layer, and central AI platform layer, forming a truly closed-loop development system. Alibaba Cloud Edge Node Service (ENS), with over 3,200 global nodes, supports deploying Qwen-8B models to edge nodes, achieving millisecond-level response in government service approval scenarios.

From a technological breakthrough perspective, current progress in deploying large models at the edge and device focuses on three dimensions: model lightweighting, edge hardware upgrades, and cloud-edge collaboration architecture. In model lightweighting, post-training quantization (PTQ) has become mainstream; 8-bit quantization can reduce GPT-3 storage requirements from 350GB to 70GB while maintaining over 95% inference accuracy; ByteDance's LLaMA-7B, after 4-bit quantization, achieves inference speeds of 30 tokens per second on consumer GPUs. In edge hardware, Synaptics SR series MCUs integrate the Arm Ethos-U55 NPU, capable of running ResNet-50 for real-time image classification at 100 GOPS with power consumption only 1/32 of traditional solutions.

The core principle of three-tier collaboration is "decentralized management at the edge"—cloud-edge-device multi-agent frameworks redefine edge agents through decentralized management principles: decoupling high-level policy design from tactical task execution, enabling edge agents to perform autonomous tactical planning and self-correction without cloud intervention.

6.3 Internet of Agents

When hundreds of millions of agents are distributed across cloud, edge, and device and interconnect and collaborate, the Internet of Agents emerges—a new network paradigm where agents serve as the fundamental interaction units of the network.

China Mobile, from the perspective of a network operator, pioneered the concept and architecture of the Internet of Agents, fully recognizing the profound transformations brought by the industry's development. The Internet of Agents must address four fundamental issues: addressing and discovery, interaction and protocol, identity and security, and efficiency and assurance. Among these, addressing and discovery are foundational capabilities for building the Internet of Agents—how to rapidly locate and discover agents capable of completing specific tasks among billions of agents. Through semantic routing and cross-domain discovery protocols (AIDP), the AONP framework enables intent-based rather than IP-based agent addressing, shifting agent communication from "address-driven" to "intent-driven."

The vision of the Internet of Agents is "universal agent connectivity"—all agents, under unified protocol standards, can freely communicate and collaborate like devices on the Internet, forming collective intelligence. This requires cross-vendor, cross-platform, cross-domain protocol standards support, which is gradually becoming reality through the efforts of standards organizations such as IETF, 3GPP, ITU-T, and CCSA.

7 From Fixed Programs to Living Code: AI's Autonomous Programming and Self-Evolution

Under the "LLM-as-CPU" architecture, a fundamental paradigm shift is occurring: software is no longer "fixed programs" pre-written by human developers but "living code" generated in real-time, continuously evolving, and self-optimizing by AI. This shift touches the core of the computing paradigm—from executing predefined instructions to dynamically generating and rewriting code.

7.1 Autonomous Programming: From Intent to Code

Traditional software development requires humans to write precise code instructions, whereas AI agents are transforming this process into "expressing intent as programming." This brings about changes at two levels.

Professional Development Level: From Writing Code to Commanding Agent Armies. Developers no longer write code line by line but act as "commanders" orchestrating AI agents to complete entire processes from requirements analysis to deployment. DARWIN (Dynamic Agentically Rewriting Self-Improving Network) is a typical evolutionary GPT

model that utilizes a genetic-algorithm-like optimization structure, having multiple independent GPT agents train with unique training code. In each iteration, GPT models are prompted to modify each other's training code, attempting to improve performance in a mutation-like fashion. DARWIN achieved a 1.26% improvement in Model FLOPs Utilization (MFU) and a 2.07% improvement in perplexity over five iterations. The system also leverages persistent JSON-based memory files to track prior reasoning and code changes, correlating improvements in model performance.

Mass Creation Level: No-Code Generation. For ordinary users, "expressing intent" becomes the new programming paradigm. ConSelf (Consensus-driven Self-improving Code Generation) addresses a challenging question: can code language models self-improve without superior teachers and test oracles? The approach introduces a new metric—code semantic entropy—which measures problem-level uncertainty by evaluating the functional diversity of program behaviors and drives preference optimization based on behavioral consensus. Experiments show that ConSelf significantly improves code generation capabilities without external supervision, validating the effectiveness of semantic-entropy-based curriculum construction and consensus-driven optimization.

Think-Anywhere further proposes a novel reasoning mechanism enabling LLMs to invoke thinking on-demand at any token position during code generation, breaking through the limitation of traditional chain-of-thought methods that can only reason at fixed positions.

MemoCoder demonstrates the capability of LLM-supported agents for automated function synthesis, achieving over 3.1% improvement in iterative debugging, error handling, and adaptation to diverse problem structures compared to zero-shot prompting and self-repair strategies.

7.2 Self-Evolution: From Parameter Updates to Continuous Capability Growth

The self-evolution of AI agents is a key capability that elevates them from "tools" to "partners." This evolution manifests across three progressive levels.

First Level: Experiential Memory and Accumulation. This is the foundation of evolution, transitioning agents from amnesiac states that "reset every session" to "newcomers" capable of accumulating experience. PRIME (Proactive Reasoning via Iterative Memory Evolution) is a training-free proactive reasoning framework that achieves continuous agent evolution through iterative memory evolution, enabling learning through explicit experience accumulation rather than costly parameter optimization. AutoAgent provides a unified framework for evolutionary cognition and elastic memory orchestration, offering a practical foundation for adaptive autonomous agents that need to learn from experience while making reliable context-aware decisions in dynamic environments.

Memento-Skills enables agents to design new agents end-to-end, breaking through traditional methods that rely on human-designed agents. In memory-augmented agents, research shows that memory-augmented agents improve decision-making by leveraging

causal relationships between actions and outcomes, achieving more effective adaptation in dynamic scenarios.

Second Level: Self-Optimization and Improvement. This is the core of evolution, enabling agents not only to execute code but also to improve the methods by which code is executed. E-SPL (Evolutionary System Prompt Learning) achieves evolutionary learning of system prompts through joint improvement of model context and model weights across reinforcement learning iterations. MetaClaw is a continuous meta-learning framework that jointly evolves the base LLM policy and a library of reusable behavioral skills. It employs two complementary mechanisms: skill-driven rapid adaptation, which uses an LLM evolver to analyze failure trajectories and synthesize new skills, achieving immediate improvement with zero downtime; and opportunistic policy optimization, which performs gradient-based updates via cloud LoRA fine-tuning and reinforcement learning with process reward models (RL-PRM), triggered by an Opportunistic Meta-Learning Scheduler (OMLS) during user inactivity windows. Experiments on MetaClaw-Bench and AutoResearchClaw show that skill-driven adaptation improves accuracy by 32% relatively, and the full pipeline elevates Kimi-K2.5's accuracy from 21.4% to 40.6%, with an 18.3% improvement in composite robustness.

Hermes Agent's embedded learning loop demonstrates another self-optimization paradigm: automatically activated after each task execution, it achieves continuous self-improvement through a closed loop of "user interaction → behavior recording → outcome evaluation → strategy optimization → skill consolidation." As usage time increases, the agent's capabilities continually strengthen, realizing the "becomes more attuned to you with use" effect.

Third Level: Meta-Cognition and Cross-Domain Evolution. This is the advanced form of evolution, enabling AI not only to improve how things are done but also to improve the "methods of improvement" themselves. The Darwin Gödel Machine (DGM), inspired by Gödel machines and open-ended evolution research, is a self-improving system that iteratively modifies its own code (and thereby also improves its ability to modify its own codebase) and empirically validates each change using coding benchmarks. DGM maintains an archive of generated coding agents and conducts open-ended exploration by sampling agents from the archive and using a base model to create new versions. Experiments show that DGM automatically improved its coding capabilities (e.g., better code editing tools, long context window management, peer review mechanisms), with performance on SWE-bench increasing from 20.0% to 50.0%, and on Polyglot from 14.2% to 30.7%.

Meta AI's proposed Hyperagents represent a major breakthrough in this direction. It merges task agents with meta-agents, allowing the system to simultaneously modify both the agent itself and its modification process, achieving "meta-cognitive self-modification" where improvement strategies become transferable and accumulate with runs. Hyperagents continuously improve across four domains—coding, paper reviewing, robot reward design, and Olympiad-level math scoring—outperforming baselines without self-improvement and previous self-improvement systems (including DGM). The core

breakthrough is that "improvement of improvement" can be reused across domains (e.g., persistent memory, performance tracking), breaking through the previous limitation of self-improvement systems being confined to the coding domain. Across domains such as coding, paper review, robot reward design, and Olympiad-level math scoring, Hyperagents achieve continuous performance gains, surpassing non-self-improving baselines and the original Darwin Gödel Machine by up to 25% in efficiency metrics.

Researchers in agentic evolution further propose that agent evolution represents the inevitable future of LLM adaptation, elevating evolution itself from a fixed pipeline to an autonomous evolver agent.

7.3 Challenges and Outlook

While the prospect of self-evolution is promising, it also faces key challenges. First is the issue of reliability—ensuring the quality of AI-generated code and eliminating "code hallucinations." Second is security—how to prevent uncontrollable recursive self-improvement when AI can modify its own code. The Darwin Gödel Machine addresses this through measures such as sandboxing and human oversight. Hyperagents, through traceable modification logs, support transparency and align with the EU AI Act's requirements for self-modifying AI. Third is the transformation of traditional software engineering paradigms—when "software" itself becomes a dynamic, continuously evolving entity, the entire development, testing, deployment, and operations processes need to be redefined.

8 Toward a General-Purpose Intelligent Application Specification

Based on the above analysis, we can outline a three-pillar architecture for a general-purpose intelligent application specification standard centered on LLMs:

8.1 First Pillar: Unified "Memory File System"

A standard is needed to govern how agent memories are organized and accessed. Elements of this standard include:

- Data Format Standard: Define unified representation formats for conversation flows, user profiles, knowledge graphs, skill libraries, etc.
- Access Interface Standard: Provide CRUD APIs enabling all agents to uniformly read and write memories.
- Ownership and Portability Standard: Define specifications for encryption, export, and migration of memory data, allowing users to freely migrate their digital persona.
- Privacy Classification Standard: Define storage locations (local/cloud) and access control policies for memory data of varying sensitivity levels.

8.2 Second Pillar: Standardized "Bus Protocol Stack"

Standardization of agent communication protocols is the absolute cornerstone of ecosystem prosperity. An ideal protocol stack should possess:

- Discovery Mechanism: Enabling LLMs to automatically discover and understand available tools and other agents.
- Unified Invocation Interface: Providing consistent invocation methods for all tools and agents, shielding underlying differences.
- Security Model: Built-in standardized permission management mechanisms such as OAuth 2.0/OIDC.
- Observability: Standardized formats for logs, metrics, and traces, supporting monitoring and debugging of agent systems.
- Protocol Interoperability: Define bridging specifications among protocols such as MCP, A2A, and AONP to ensure collaborative operation of the protocol stack.

8.3 Third Pillar: Operating System "Kernel" and "SDK"

A unified system platform is needed for agent application developers:

- Agent Lifecycle Management: Standard processes for agent creation, start, pause, resume, termination, and destruction.
- Resource Scheduler: Manage competitive access of multiple agents to token budgets, API quotas, and computational resources.
- Security Sandbox: Ensure runtime environments of different agents are isolated, preventing failure or malicious behavior of one agent from affecting others.
- SDK Standardization: Provide unified APIs for developers to create, deploy, and manage agents, lowering development barriers.
- Agent Store Specification: Define distribution channels, review mechanisms, and billing standards for agents to support ecosystem prosperity.

9 Conclusion and Outlook

This paper, from the perspective of "treating LLM as CPU," systematically analyzes the technological components and architectural blueprint required to construct a general-purpose intelligent application specification. By analogy with traditional PC architecture, we identify memory subsystems (MemGPT/Letta, Mem0, Hermes Agent), communication protocol stacks (MCP, A2A, AONP), agent runtime frameworks (OpenClaw, Hermes Agent), and agent operating systems (AIOS, Agent OS, Synoetic OS) as key components. We propose a five-layer Agent-OS architecture, a cloud-edge-device three-tier collaborative deployment paradigm, and the vision of the Internet of Agents. We also delve into the self-evolution pathway of AI agents from "fixed programs" to "living code," and preliminarily outline the three pillars for advancing a general-purpose intelligent application specification.

At the cloud-edge-device collaborative deployment level, we demonstrate the complete technology stack for on-device personal agents—from chips (Qualcomm Snapdragon Wear Elite, ROMA architecture, SoulMate SoC) to system architecture (five-layer on-device AI agent system) to edge computing clusters. On-device deployment ensures data never leaves the device for inference, eliminating privacy leakage risks; the edge layer provides proximal computational augmentation; the cloud layer handles global orchestration and complex

reasoning. The three tiers enable dynamic balancing of AI capabilities among efficiency, security, and cost.

At the self-evolution level, we reveal a three-tier progressive pathway for AI agents: from experiential memory accumulation (PRIME, AutoAgent) to self-optimization and improvement (MetaClaw, E-SPL), and ultimately to meta-cognitive cross-domain evolution (Darwin Gödel Machine, Hyperagents). This evolutionary capability signifies a fundamental shift in the nature of software—from "fixed programs" written by humans to "living code" autonomously generated and optimized by AI.

However, this field remains in its infancy, with numerous critical issues warranting further investigation:

- Cross-Vendor Agent Interoperability: How can agents developed by different vendors collaborate seamlessly? This requires further refinement and widespread adoption of protocol standards.
- Agent Security and Ethical Governance: When agents can act and make decisions autonomously, how can we ensure their behavior aligns with human values? Security constraints within Agent Contracts require further detailing and formal verification.
- Fair Resource Scheduling: When thousands of agents compete for limited token budgets and API quotas, how can fair and efficient scheduling algorithms be designed? AIOS experiments have preliminarily demonstrated the value of scheduling optimization, but scheduling strategies for heterogeneous mixed agent workloads require deeper investigation.
- Scaled Deployment of the Internet of Agents: When the number of agents reaches the billions, can existing discovery and routing protocols sustain? Semantic routing and distributed service discovery technologies require further breakthroughs.
- Safety Boundaries of Self-Evolution: When AI can modify its own code and evolutionary strategies, how can we ensure this evolution always proceeds in a direction beneficial to humanity? This requires synergy among formal verification, runtime monitoring, and human oversight mechanisms.

From the GUI era to the NUI era, and onward to the AUI era, we are witnessing a profound transformation in the computing paradigm. If the hallmark of the PC era was "compatibility" (enabling hardware and software to work together), then the hallmark of the LLM era will be "comprehensibility"—enabling machines to understand human fuzzy intents and decompose them into precise machine instructions. Building such a set of standards requires collaborative efforts from academia and industry, much as the Wintel alliance defined the trajectory of an entire industry for decades during the PC era. We anticipate that by establishing a general-purpose intelligent application specification standard, we can truly unleash the immense potential of LLMs as the new "CPU" and inaugurate a new era of intelligent computing.

References

[1] K. Mei, X. Zhu, W. Xu, W. Hua, M. Jin, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang, "AIOS: LLM Agent Operating System," arXiv preprint*, arXiv:2403.16971, 2025.*

[2] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "MemGPT: Towards LLMs as Operating Systems," arXiv preprint*, arXiv:2310.08560, 2023.*

[3] "Model Context Protocol (MCP) Specification," Anthropic, 2024–2025.

[4] "Agent2Agent (A2A) Protocol," Google, 2025. [Online]. Available: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[5] "Agent Operating Systems (Agent-OS): A Blueprint Architecture for Real-Time, Secure, and Scalable AI Agents," **TechRxiv*, DOI: 10.36227/techrxiv.175736224.43024590/v1, 2025.*

[6] "Agent Internet Open Network Protocol (AONP) Framework and Agent Gateway," China Mobile Research Institute, 2026.

[7] "Future AI Operating Systems (III)—The Hub of Intelligence: From Models to System Unification," Tencent Cloud Developer Community, 2025.

[8] J. Zhang, B. Zhao, W. Yang, J. Foerster, and J. Clune, "Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents," arXiv preprint*, arXiv:2505.22954, 2026.*

[9] P. Xia et al., "MetaClaw: Just Talk—An Agent That Meta-Learns and Evolves in the Wild," arXiv preprint*, arXiv:2603.17187, 2026.*

[10] H. Jiang, "DARWIN: Dynamic Agentically Rewriting Self-Improving Network," arXiv preprint*, arXiv:2602.05848, 2026.*

[11] H. Zhang, W. Cheng, and W. Hu, "Self-Improving Code Generation via Semantic Entropy and Behavioral Consensus," arXiv preprint*, arXiv:2603.29292, 2026 (Accepted at ICPC 2026).*

[12] "Qualcomm Launches New Snapdragon Wear Elite Platform, Empowering the Rise of Personal AI," Communication World Network, March 3, 2026.

[13] "31.5 SoulMate: A 9.8mW Mobile Intelligence System-on-Chip with Mixed-Rank Architecture for On-Device LLM Personalization," IEEE, 2026.

[14] Hu Junjie, "Device-Edge-Cloud Collaboration: Building a New Paradigm for Inclusive AI Terminals," ZTE, June 20, 2025.

[15] "Analysis of Current Deployment Status of Large Models at the Edge Based on Existing Sizes and Inference Compute Scale," Huawei Cloud Developer Community, June 27, 2025.

[16] W. Wang et al., "ROMA: a Read-Only-Memory-based Accelerator for QLoRA-based On-Device LLM," arXiv preprint*, arXiv:2603.xxxxx, 2026.*

[17] “Anthropic Upgrades Open Source MCP To Scale Tool-Rich AI Agents,” *Open Source For You*, January 16, 2026.*

[18] “What is the A2A Protocol? Read This and You'll Understand,” ZOL Zhongguancun Online, July 27, 2025.

[19] “Synoetic OS v1.0: Substrate-Independent AI Orchestration Through Narrative Coherence,” Zenodo*, 2025.*

[20] “DingTalk Releases World's First AI Work Intelligence Operating System Agent OS,” ChinaZ, December 23, 2025.

[21] “OpenClaw 'Lobster' Becomes Highest-Starred Open Source Project in History,” DoNews, March 11, 2026.

[22] “Hermes Agent Replaces Lobster? The AI Assistant That Garnered 40k Stars,” 36Kr, April 13, 2026.

[23] “PRIME: Training-Free Proactive Reasoning via Iterative Memory Evolution for User-Centric Agent,” arXiv preprint*, 2026.*

[24] “Meta AI Releases Hyperagents: A Major Breakthrough in Cross-Domain Self-Improvement,” AI News, March 23, 2026.

[25] “On-Device AI Phone/PC Agent System Architecture Diagram,” ProcessOn, March 30, 2026.

[26] “Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory,” arXiv preprint*, 2025.*

[27] “LLMs as Operating Systems: Agent Memory,” DeepLearning.AI, 2025.

[28] “Eclipse LMOS Redefines Agentic AI with Industry's First Open Agent Definition Language,” Eclipse Foundation, 2025.