Towards Pervasive Distributed Agentic Generative AI - A State of The Art

GIANNI MOLINARI, University of Turin, Italy
FABIO CIRAVEGNA, University of Turin, Italy

The rapid advancement of intelligent agents and Large Language Models (LLMs) is reshaping the pervasive computing field. Their ability to perceive, reason, and act through natural language understanding enables autonomous problem-solving in complex pervasive environments, including the management of heterogeneous sensors, devices, and data. This survey outlines the architectural components of LLM agents (profiling, memory, planning, and action) and examines their deployment and evaluation across various scenarios. Than it reviews computational and infrastructural advancements (cloud to edge) in pervasive computing and how AI is moving in this field. It highlights state-of-the-art agent deployment strategies and applications, including local and distributed execution on resource-constrained devices. This survey identifies key challenges of these agents in pervasive computing such as architectural, energetic and privacy limitations. It finally proposes what we called "**Agent as a Tool**", a conceptual framework for pervasive agentic AI, emphasizing context awareness, modularity, security, efficiency and effectiveness.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; **Natural language generation**; • **Human-centered computing** → **Ubiquitous computing**; • **Networks** → **Cloud computing**.

Additional Key Words and Phrases: Pervasive Computing, LLM Agent, Edge, Fog, Cloud, SLM, RLM

1 Introduction

Agents, designed to perceive, reason, and act within their environment, are pivotal for solving complex tasks. The advent of Large Language Models (LLMs) has recently revolutionised this domain. Trained on vast web datasets, LLMs demonstrate an impressive understanding of human language and can generate remarkably similar and accurate responses. Integrating these capabilities has led to the development of LLM-based agents, where LLMs serve as the core cognitive engine, combined with perceptual, reasoning, and action mechanisms. This synergy has introduced a new paradigm of more intelligent and versatile agents applicable across diverse domains. Agents can have various capabilities. Some are purely reactive, responding to explicit user prompts [26, 137], while others show proactive functionalities by autonomously initiating tasks based on their understanding of the environment [131]. Their applications are widespread. Embodied agents, interact with the physical world via sensors and actuators [24], whereas software agents operate in digital environments, performing tasks like information retrieval or software and research development [132, 136].

Also the field of pervasive computing, focused on integrating computing and communication environments into human daily life, has made considerable progress. This ranges from high-performance computing (HPC) systems to resource-constrained IoT devices, with applications in smart homes, cities, factories, or hospitals. In this pervasive context, agents offer vast potentials. Their abilities to perceive, reason, and interact can significantly enhance the scope and utility of pervasive computing applications. However, deploying agents in these environments presents significant computational, and security challenge that necessitate custom solutions. Researchers have proposed various solutions to solve these challenges. Therefore, this paper analyses strategies for integrating agents into pervasive computing, examines existing applications, identifies persistent challenges, and explores ongoing research efforts to address them. Specifically, the paper is structured as follows:

(1) First, an introduction to LLM-based agent systems will cover their architecture, performance evaluation metrics for accuracy and robustness, and diverse applications.

Authors' Contact Information: Gianni Molinari, gianni.molinari@unito.it, University of Turin, Turin, Italy; Fabio Ciravegna, University of Turin, Turin, Italy, fabio.ciravegna@unito.it.

(2) Next, the paper will introduce pervasive computing, detailing recent advancements, its underlying infrastructure, and the integration of artificial intelligence, culminating in agent adoption.
(3) Then, it will explore agent implementation in pervasive environments, focusing on architectural strategies for effective deployment tailored to available infrastructure, supported by application examples.
(4) A detailed discussion will then analyse unresolved challenges in the field and current research directions aimed at their resolution.
(5) Finally, the paper will propose a future research vision for LLM-Agents in pervasive computing, culminating in the "**Agent as a Tool**" concept.

2 Agent LLM Architecture

As described by Franklin and Graesser, an agent can be defined as a system that exists within an environment, it perceives that environment through observation using sensors, and acts upon it through effectors, over time, in pursuit of its own agenda and so as to effect what it senses in the future [27].

In the context of LLM-based agents, a natural language processing pipeline is employed to enable the agent to perceive, reason, and interact with its environment to accomplish specific goals. Perception, the initial stage in this process, involves gathering information about the current state of the environment [98]. This often occurs through natural language inputs, where users provide instructions or queries [26], or structured text, including HTML or a programming language formats [35, 137]. In addition, multimodal LLMs (MLLM), process visual and auditory data, resulting in enhanced environmental comprehension and information use [24]. Also, to enhance environmental interaction, respect constraints, and achieve goals, LLM-based agents require a robust understanding of implied textual meanings [45]. Following perception, the agent leverages the LLM's natural language processing capabilities for reasoning and planning. This involves in processing the perceived information, interpreting user intent, and analyzing possible courses of action [65, 98]. A key aspect of this stage is the formulation of strategic plans to achieve desired outcomes. Complex tasks are often decomposed into manageable subtasks to facilitate execution [45, 65] such as for robots movements management [120], automatic world exploration [94] or code debugging [101]. The final stage involves in interacting directly with the environment following the previous reasoning steps [98]. Actions may include generating natural language responses to the user, providing explanations or completing tasks through dialogue [111, 136], invoke external tools or APIs [24, 77] or produce executable code [132]. To accomplish all these tasks modern agent architectures typically consist of multiple key modules that work together to create intelligent behavior. A typical LLM-agent architecture has four key modules: a profiling module for defining the agent's role, a memory module for storing and retrieving past experiences, a planning module for formulating future actions, and an action module for translating decisions into outputs [136]. Also, evaluating this architecture is essential to determine the effectiveness of its modules, especially within the intended application domain. The following sections will explore the LLM architecture, the specific functionalities and interactions of the agent modules, and the evaluation strategies and application domains of these intelligent LLM agents.

2.1 Large Language Models

The LLM architecture serves as the core cognitive component, or "brain", of these intelligent agents. Its fundamental operation relies on "next token prediction". This means that given an input sequence of words (tokens), transformed into numerical embeddings, the LLM predicts the subsequent word in the sequence, iteratively generating the output. The success of LLMs in this task is largely attributed to their foundation in the Transformer architecture, illustrated in Figure 1. Its design captures long-range dependencies within the text and effectively models the complex structures of human language [93]. A typical LLM consists of stacked Transformer layers, each incorporating a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.

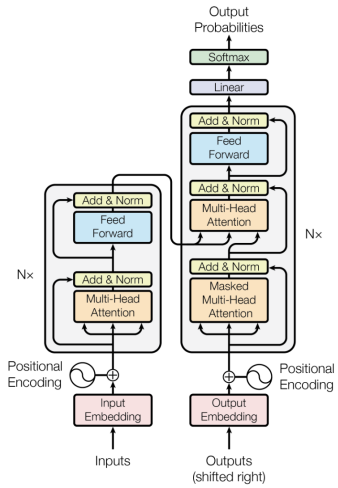

Fig. 1. The transformer architecture, from [93]

The multi-head self-attention mechanism allows the model to weigh the importance of different parts of the input sequence when processing each word. It processes the encoded input as a set of key-value pairs (K, V), where both keys and values have a dimension equal to the input sequence length (n). The Transformer employs scaled dot-product attention to calculate how much attention each value should receive based on a query (Q):

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{n}}V)$$

Instead of performing this attention calculation just once, the multi-head mechanism does it in parallel multiple times. The resulting independent attention outputs are then combined (concatenated) and transformed linearly to produce the desired output dimensions. The Transformer architecture is fundamentally composed of two main components: an encoder and a decoder. The encoder's role is to create an attention-based representation of the input, enabling it to "attend" to relevant information. Each encoder layer includes a multi-head self-attention sub-layer and a simple position-wise feed-forward network, with each sub-layer benefiting from a residual connection and layer normalization. The decoder's function is to generate the output sequence based on the encoded representation. Each decoder layer includes two multi-head attention sub-layers (one for self-attention and one for attention over the encoder's output) and a feed-forward network, again with residual connections and layer normalization. Finally, a a linear layer and a softmax are applied to the decoder's output, producing a probability distribution over the vocabulary from which the next word is sampled.

LLMs commonly employ this Transformer architecture, scaling it to billions or even trillions of parameters and incorporating various optimization strategies. For instance, decoder-only architectures like GPT-3 and LLaMA predict each next token based on the preceding ones. Conversely, encoder-only architectures such as BERT and RoBERTa prioritise understanding the input text to generate task-specific outputs like labels or token predictions [60]. In the agentic field, the choice of architecture depends on the required task. Typically, decoder-only architectures are most commonly used when we want an agent to perceive, reason, and interact with the environment. Furthermore, Multimodal Language Models (MLLMs) are increasingly gaining traction for agents interacting in multimodal environments (text, images, video, audio). This is because they allow the application of Transformer architectures across diverse modalities like vision and audio. To enable such

multimodal capabilities, MLLMs extend the Transformer architecture by embedding non-text inputs (e.g., image patches or audio tokens) into the same vector space as textual tokens. These embeddings are then processed jointly within shared Transformer layers or through modality-specific experts, such as in Multiway Transformer architectures. Deep modality fusion is achieved via cross-attention or unified encoders, enabling fine-grained interactions between visual and textual features [50].

2.2 Agent Modules

LLM-based autonomous agents are designed to effectively perform diverse tasks by leveraging the capabilities of Large Language Models (LLMs). To achieve this, their architecture typically incorporates 4 key modules: profiling module, memory module, planning module, and action module.

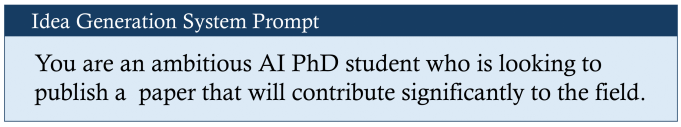

Fig. 2. An example of profiling prompt used in [57]

2.2.1 Profiling. The primary purpose of the Profiling Module is to define the role-specific identity of the agent, which significantly influences its behavior and interactions [98, 136]. This module is crucial for enabling agents to generate more contextually relevant responses from the LLM across different modalities. These profiles are typically added into the model context window. By including role descriptions in the prompt, the profiling module guides the LLM's reasoning style, response formulation, and interaction strategy [136]. For instance, in [57] the agent is profiled to work as PHD student that want to publish some papers (Figure 2). The profiling module serves as the foundation for agent design, giving a significant influence on the entire agent's modules ecosystem [98]. The chosen profile can impact how the agent remembers information (memory module), how it formulates strategies to achieve its goals (planning module) that consequently will impact also the future actions. For example, an agent profiled as '*an expert machine learning researcher*' [35] might prioritise certain experiments over others or selectively memorise results more significant to its research. Similarly, an agent profiled as '*an advanced AI system serving as an impartial judge for intelligent code generation outputs*' [144] would focus more intently on code details, errors, and bugs than on other capabilities. Agent profiles are commonly implemented by handcrafting specific prompts that describe the desired role and behavior [98]. However, other strategies can be used, including LLM-generation of profiles and dataset alignment to reflect real-world characteristics [98]. Combining these strategies can yield additional benefits. Moreover agent profiles can include multiple dimensions, including basic attributes, behavioral patterns, and social information [136].

2.2.2 Memory. LLM architectures lack persistent memory across test time interactions, requiring prior exchanges to be explicitly included within the context window (Figure 3). This limitation underscores the need for effective memory management strategies in LLM-based agents, particularly for prioritizing relevant information during storage and retrieval. As the agent's interaction history expands, exceeding the fixed context window's capacity, summarization or information filtering becomes unavoidable. Consequently, autonomous LLM-based agents rely on a memory module to retain, recall, and reflect upon past observations, thoughts, feedback, and actions. This enables them to better understand the current context, anticipate future needs, correct their action trajectory, adjust their behavior, and maintain consistent and coherent behavior over time [98, 105, 111]. This module often incorporate different structures to manage information across varying timescales:

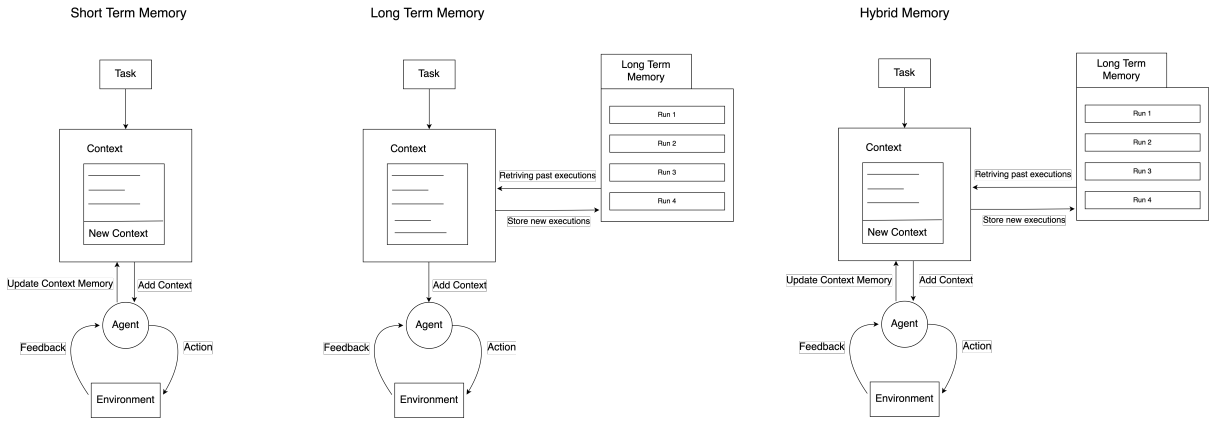

Fig. 3. From left to right, the diagram displays the structures of Short-term, Long-term, and Hybrid memory.

- **Short-Term Memory:** This type of memory maintains contextually relevant information about recent perceptions, reasoning and actions (the trajectory history) within the LLM context window [85, 136]. It allows for real-time adaptation during an interactive session. Examples include the internal states maintained by a conversational agent or the intermediate steps during a web research [137]. However, the limited context window of LLMs poses a challenge for relying solely on short-term memory.
- **Long-Term Memory:** In contrast to the real-time encounter observations, which provide short-term memory, the overall encounter history represents long-term memory [138]. Long-term memory can provide stable knowledge to complement the flexibility of short-term memory. This module is important because stores both successful experiences, demonstrating correct goal achievement strategies, and unsuccessful experiences, highlighting incorrect paths to avoid. For instance in [107], successful paths are stored as 'Thought Cards' and retrieved upon encountering a new question with a similar topic or in [131] the agent store historical events that will be used to predict potential tasks.
- **Hybrid Memory:** Many agents use a combination of short-term and long-term memory to leverage the benefits of both [98]. Short-term memory handles immediate context, while long-term memory provides access to a broader history and consolidated knowledge.

These structures can be stored in various formats, each with its own advantages. One approach is to keep everything in natural language form. This is especially useful for conversational agents, where it's important to be transparent about the agent's history. For instance, in [85] store past trials as verbal descriptions, making it easy to see what the agent has learned. Similarly, in [94] descriptions of skills within the Minecraft game are stored directly as natural language within the agent's memory. Then, there's the idea of Embedding Memory. Here, past experiences are transformed into high-dimensional vectors [98]. This allows for really fast reflection and helps the agent generalise across different situations. It's especially good for searching through knowledge quickly, based on semantic similarity [136]. Finally databases can be used to store memories. This gives us considerable power to manipulate the data through structured queries. So agents can be used to understand and execute SQL queries in natural language, which lets it interact with the database really effectively [84]. Memory module supports three core operations. First writing, which involves persisting relevant information from past interactions into the memory. Key considerations during memory writing include handling duplicated information and managing memory overflow [11]. Techniques such as summarizing similar information, using a FIFO buffer to overwrite old entries [98], truncating historical data that exceeds processing capacity [111], and

implementing time limits to discard redundant data and prevent excessive memory consumption [142] are used. Then, memory reading is a crucial operation that aims to obtain relevant knowledge from the agent's memory to adapt its behaviors and inform its next actions [136]. The retrieval of valuable information often considers factors like recency, relevance, and importance of the memories. These factors commonly include the recency of the memory, its relevance to the current situation or query, and its perceived importance [98, 111]. Finally, memory reflection allows the agent to analyse, refine, and optimise the accumulated memories [136]. It emulates human cognitive processes of summarizing and inferring more abstract and high-level insights from past experiences. Reflection can occur hierarchically, generating insights based on existing insights [85]. As mentioned in the previous section, the memory module is not isolated. It is influenced by the agent's profile, which can determine what types of information are prioritised for storage and retrieval. Furthermore, the memory module plays a critical role in the planning process, providing the necessary context and historical data for the agent to formulate effective future actions. The planning module might retrieve information from memory to inform its reasoning and decision-making.

2.2.3 Planning. The planning module is essential for autonomous agents, enabling them to formulate future action sequences to achieve defined goals [98]. It allows agents to move in goal-directed, multi-step problem-solving. The planning module combines environmental feedback, current state, desired outcomes, memory, and profile information to construct a trajectory of actions that effects a transition from the agent's current state to a target goal state. Various methodologies have emerged for incorporating LLMs into the planning process, each representing a different approach to leveraging their capabilities:

- **LLM-as-Planner** directly employs the inherent reasoning abilities of LLMs to generate plans from natural language instructions [45]. This paradigm emphasizes the autonomous planning capacity of LLMs, relying on their ability to interpret and translate natural language directives into actionable plans.
- **LLM-as-Facilitator** uses LLMs to augment existing planning algorithms, such as classical symbolic planners. In this context, LLMs may serve as translators, converting natural language problem descriptions into formal planning languages, such as the Planning Domain Definition Language (PDDL), which are then processed by external planners [47, 98].
- **Multi-Agent Planning** involves the coordination of plans and actions among multiple agents to achieve a shared objective [31]. This necessitates the implementation of mechanisms for inter-agent communication, negotiation, and belief revision. Collaborative planning approaches are exemplified by frameworks such as PlanGEN [65], which employs specialised LLM agents for constraint checking, verification, and selection, and Master [28], which proposes a hierarchical multi-agent framework that dynamically generates collaborating agents, validates reasoning, and adjusts confidence-based scoring using Monte Carlo Tree Search (MCTS) to enhance accuracy and efficiency.

One key aspect of the planning module is task decomposition, where complex tasks are broken down into smaller, more tractable sub-problems, facilitating efficient execution. For example, PlanGEN [65] is specifically engineered to augment LLMs' capacity to generate effective natural language plans through this decomposition process. Similarly, WebPilot's Planner module [137] initiates its operation by partitioning complex web tasks into manageable subtasks, thereby constructing a flexible, high-level plan that can adapt to the inherent uncertainties of web environments. Consequently, determining the optimal sequence of these subtasks or individual actions, becomes crucial for achieving the overarching goal. In more complex scenarios, the planning module may also necessitate the allocation of available resources across various planning components [45]. This is exemplified by the TravelPlanner benchmark [112], which evaluates agents' ability to generate travel itineraries from user queries, demanding the navigation of extensive online resources using specialised tools and adherence to user constraints. This benchmark assesses the agent's resource allocation across diverse tasks, including city searches,

flight bookings, accommodation selection, and dining arrangements. In the current literature planning module employs diverse strategies in task decomposition:

- **Single-Path Planning:** The agent follows one trajectory of thought and action at a time, without exploring alternative possibilities. Examples include Chain of Thought prompting, where the LLM reasons step-by-step along a linear path [98].
- **Tree-Based Planning:** The agent explores multiple potential thought trajectories, organizing them into a tree structure. This allows for backtracking and evaluating different options before committing to a plan. Techniques like Tree of Thoughts and methods using Monte Carlo Tree Search (MCTS) fall under this category [45].
- **Hierarchical Planning:** Involves planning at different levels of abstraction, with high-level plans broken down into more detailed, low-level actions or sub-goals [111, 137].

Effective integration of feedback within the planning module significantly enhances the ability of LLM-based agents to operate in complex and dynamic environments [31]. This capability allows agents to move beyond static, pre-defined plans and engage in a more adaptive and iterative problem-solving process. So we can categorise planning approaches in 2 groups:

- **Planning without Feedback:** The agent formulates a plan upfront and executes it sequentially without adjusting based on intermediate outcomes [31]. This approach is appropriate for tasks of lower complexity, including conversational interaction.
- **Planning with Feedback:** The agent receives information from the environment, humans, or other models about the progress and outcomes of its actions [45]. This feedback is then used to dynamically adjust and refine the plan. Environmental Feedback can include task completion signals or observations after taking an action [122]. Human feedback can provide guidance or corrections [26]. Model-based feedback, such as self-reflection or critique from another AI model, can also be valuable [85].

Finally, careful consideration of memory and profiling is vital for effective planning. The memory module provides the historical context, past experiences, and relevant knowledge needed to formulate informed plans. For instance, past successes or failures in similar situations, user preferences, or environmental observations stored in memory can guide the planning process [85]. The agent's profile, defined by the profiling module, can also influence the style and focus of the planning process. For example, an agent profiled as risk-averse might prioritise plans with higher certainty of success, while a creative problem-solver profile might encourage the exploration of novel or less conventional plans [136].

2.2.4 Action. The primary goal of the Action Module is to translate the agent's formulated plans into specific outcomes within the environment. It acts as the execution engine, directly engaging with the surrounding world [98]. The intended outcomes of agent actions are diverse and intrinsically linked to the agent's designated task. Actions may be directed towards achieving concrete objectives, such as item crafting in a game [94] or software development task completion [101], where each action contributes directly to the final goal. In the context of a web browsing agent [42], actions can include navigating or interacting to specific pages. Actions may also facilitate information sharing and collaboration with other agents [28] or human users [26], including conversational exchanges and feedback provision. Furthermore, agents perform exploratory actions to gain new knowledge and expand perception [105], optimizing learning and performance through exploration-exploitation. The Action Module receives from the Planning Module a sequence of steps or a specific action to be taken and translates them into executable actions [98]. The Action Module's operational mechanism varies with the agent's architecture and action space, which collectively define the agent's interaction capabilities. Actions can be broadly categorised as follows:

- **External Tool and API Interaction:** LLM-driven agents frequently generate natural language commands or structured calls to external tools and APIs [77]. This allows them to interact with the external world, using resources like search engines for web navigation [42] (including clicking, form filling, and scrolling), databases via SQL queries [84], code execution environments [101], PDDL solver execution [10], or specific application such as FlightSearch API with correctly formatted parameters [112]. To simplify the integration of diverse APIs, the Model Context Protocol (MCP) [2] standardises API access, eliminating manual tracking and description. MCP's Host-Client-Server architecture facilitates dynamic tool discovery and execution, enabling any MCP-compatible LLM application to use connected server tools.
- **Internal Response Generation:** This category includes actions that generate internal responses, such as providing explanations or refining plans [85, 122]. This is crucial as each action's consequence, affecting the user's information state or the environment's state, provides vital feedback. The Planning Module uses this feedback to iteratively adjust the agent's policy and refine future actions.
- **Embodied Navigation and Manipulation:** For embodied agents or robotic control systems, the Action Module grounds high-level skills or planned actions into low-level motor commands executable in physical or virtual environments [111]. This serves as an intermediary, bridging the gap between cognitive planning and physical execution.

Therefore the design of a comprehensive and precise action space is crucial for building robust agents, as it standardises the translation of high-level plans into executable operations. To enhance execution accuracy, action names should semantically correspond to their behavior, and natural language descriptions should be provided to clarify action usage [115].

2.3 Agent Evaluation

Evaluating the capabilities of AI agents is crucial for their development and integration into various applications and specific domains. As the field of LLM-based autonomous agents grows, the need for robust evaluation methods becomes important. These tools allow for rigorous testing of architecture effectiveness within the intended operational domain. Generally, agent evaluation can be categorised into two main approaches: subjective and objective.

2.3.1 Subjective Evaluation. Subjective evaluation is particularly effective when assessing qualitative aspects of agent performance, such as overall helpfulness or user-friendliness, where quantitative metrics are difficult to define or evaluation datasets are limited. LLM-based agents are often designed to serve humans, making subjective evaluation a critical component as it reflects human criteria. This involves human evaluators directly interacting with the system or observing its performance, and then providing judgments based on predefined criteria or overall impressions. This approach often uses structured questionnaires with Likert scales to assess qualitative aspects. For instance, Franciscatto et al. [26] employed a 5-point Likert scale to evaluate visibility, support, usefulness, transparency, justification, and data integration capabilities. Similarly, Shaer et al. [79] used Likert scales to assess generated ideas for relevance, innovation, and insightfulness by both expert and novice evaluators. Furthermore, in [78] the authors conducted human surveys with PhD researchers, who rated paper quality on experimental quality, report quality, and usefulness, and also simulated peer review using NeurIPS-style scores. Subjective evaluation, while offering valuable qualitative insights, shows a series of considerable challenges. Primarily, it is resource-intensive and time-consuming, necessitating the recruitment of a sufficiently large and diverse pool of human evaluators, which can incur substantial financial costs and require extensive logistical planning for study preparation and execution. Moreover, the inherent subjectivity of human judgment introduces potential biases, influenced by individual perspectives, experiences, and cultural backgrounds. These biases can disrupt the objectivity and generalizability of evaluation results. Consequently, the interpretation and integration of subjective evaluation data require careful consideration of these inherent limitations, emphasizing the need for

rigorous methodological design and transparent reporting to mitigate potential biases and enhance the reliability and validity of findings.

2.3.2 Objective Evaluation. Objective evaluation uses quantitative metrics to enable computational, and comparative analysis, thereby providing concrete and measurable assessments of agent performance. The selection of appropriate metrics is crucial for evaluating specific aspects of agent performance. For **Task Completion and Efficiency**, evaluation focuses on the agent's ability to achieve defined objectives with efficient resource use. This category includes metrics such as Success Rate [10], quantifying overall goal attainment; Progress Rate [10], tracking incremental advancements in multi-turn interactions; Completion Rate [115], measuring the proportion of completed sub-tasks; Execution Efficiency [115], assessing action efficiency relative to sub-task completion; Cost Efficiency [115], evaluating resource consumption through token usage; Number of Steps [113], quantifying operational effort via LLM calls; and Cost per Instance [113], measuring monetary expenditure for API queries. In numerous applications, **Accuracy and Correctness** have critical importance. Metrics within this category assess the fidelity and consistency of generated information, plans, and actions. This includes Accuracy [98], evaluating overall output correctness; Answer F1 [53], measuring query response accuracy in knowledge graph environments; Test Coverage and Bug Detection Rate [98], assessing the agent's ability to generate effective test cases and identify software defects. For agents designed for conversational or interactive tasks, **Dialogue and Interaction Quality** is assessed using metrics such as Recency, Relevance, and Importance [33]. These metrics evaluate the coherence, relevance, and meaningfulness of agent dialogues, capturing the nuances of human-agent interaction. Finally, as AI agents are increasingly deployed in critical systems, **Adversarial Robustness** becomes a key evaluative consideration. Metrics such as Benign Utility, Utility Under Attack, and targeted Attack Success Rate [21] evaluate the agent's resilience to adversarial attacks, ensuring security and reliability in real-world deployments.

Frameworks and Benchmarks. To comprehensively evaluate the capabilities of agents across diverse application domains, a range of specialised benchmarks and frameworks have been developed. These include AgentBench [53] , which features tasks simulating real-world scenarios such as web browsing and gaming, alongside code generation and execution tasks involving operating systems, databases, and knowledge graphs. TheAgentCompany [113] provides a self-contained environment that models a small software company, including tasks like web browsing, code writing, and inter-agent communication. DevAI [144] presents 55 real-world AI application development tasks curated by expert annotators, while SWE-bench [119] offers 2294 software engineering problems derived from actual GitHub issues and pull requests across 12 prominent Python repositories. Text-based game environments, such as ALFWorld [85], and Minecraft [94], are used to evaluate language agent performance in interactive, simulated settings. WebShop [121] focuses on assessing product search and retrieval capabilities, and WebArena [143] provides a comprehensive website environment for end-to-end agent evaluation. RoCoBench [59] evaluates multi-agent collaboration across diverse scenarios, emphasizing communication and coordination in cooperative robotics. TravelPlanner [112] benchmarks real-world planning capabilities in language agents, and ScienceWorld [99] evaluates reasoning and planning abilities by posing questions designed to challenge a fifth-grade student. Finally, for evaluating safety and privacy, AgentDojo [21] measures AI agent resilience to prompt injection attacks, and PrivacyLens [82] quantifies potential data leakage to assess privacy norm adherence in language model agents.

2.3.3 LLM-Evaluators. A growing number of researchers are also exploring the use of Large Language Models (LLMs) themselves as intermediaries for agent assessment. For instance, ALI-Agent [95] leverages autonomous LLM-driven agents to automatically generate realistic test scenarios and iteratively refine them, enabling adaptive evaluations of alignment that effectively identify subtle, long-tail risks without relying on continuous human feedback. This framework employs a memory module for scenario generation, a tool-using module that integrates

Web search and fine-tuned evaluators to reduce human labor, and an action module for test refinement. Building upon the concept of LLMs as judges, the Agent-as-a-Judge framework [144] extends this by employing agents to evaluate other agents. Recognizing the step-by-step operation of the agents, this framework aims to provide rich, intermediate feedback throughout the task-solving process, rather than relying solely on final outcomes. This approach has demonstrated success in code generation tasks, outperforming traditional LLM-as-a-Judge methods and achieving reliability comparable to human evaluations. ChatEval [9] employs a multi-agent debate approach to enhance LLM-based evaluation quality. By deploying a team of LLM agents with diverse role prompts, ChatEval facilitates autonomous discussion and evaluation of generated responses. This synergistic approach, leveraging the unique capabilities of multiple LLMs, exhibits superior accuracy and correlation with human assessments compared to single-agent evaluations. Similar multi-agent systems, such as IntellAgent [44], use an LLM to perform roles like event generation, user agent, and dialog critique, implicitly integrating LLMs into the evaluation of agent behavior. SCALEEVAL [18] proposes an agent-debate-assisted meta-evaluation framework to address this, employing communicative LLM agents for iterative discussions that assist human annotators in identifying the most capable LLM evaluators, particularly in novel, user-defined scenarios. In cases of agent disagreement, minimal human oversight ensures a balance between efficiency and reliability. This approach enables scalable assessment of LLM evaluator trustworthiness across diverse tasks and criteria. While LLM-based evaluation offers advantages such as automation, scalability, and the ability to probe complex behaviors, it also presents challenges. These include reliance on inherent LLM capabilities and potential biases, which may cause incorrect evaluations, diverging from expert human judgment in domain-specific tasks [89]. Furthermore, LLM evaluations are susceptible to prompt engineering and format variations, potentially leading to inconsistent assessments [18]. The inherent lack of understanding in specialised domains can result in potential errors that human experts would make [89].

2.4 Field of Use

The domain-specific nature of agent evaluation highlights their adaptability, as reflected in the diverse metrics and frameworks used across various sectors. LLM-based agents demonstrate broad applicability in domains such as:

- **Software Engineering and Code Generation:** Agents simulate developer workflows using tools like terminals and GitHub. Platforms like OpenHands [101] enable secure execution of scripts and commands in sandboxed environments, supporting tool reuse and multi-agent collaboration. Reflexion [85] enhances agent reasoning through verbal feedback and episodic memory. MAGIS [91] focuses on ongoing software maintenance using multiple LLM agents for bug fixes, feature updates, and optimizations.
- **Embodied Agents:** These agents interact with physical or virtual environments using sensory feedback. LAC [114] integrates LLMs into robotic control. Steve [141] and Voyager [94] operate in Minecraft environment, while WebPilot [137] and AutoWebGLM [42] automate web-based tasks.
- **Research and Development:** Agents assist or automate the research process. Agent Laboratory [78] supports literature review and experimentation. AI Scientist [57] enables full-cycle scientific discovery. ADAS [35] automates agent system design, allowing transfer across domains.
- **Information Management and Retrieval:** Agents go beyond static information extraction by autonomously optimizing when and how to retrieve and process data. Systems like A-MEM [116] enable contextual memory and adaptive learning. RAG pipelines are enhanced by multi-agent RL architectures [17], while Search-o1 [48] empowers reasoning through agentic search and document reasoning.
- **Healthcare and Medicine:** Agents automate complex clinical and administrative tasks. ClinicalAgent [127] leverages external biomedical knowledge for trial analysis. Other systems [29] handle administrative processes like record retrieval, patient registration, billing, and appointment scheduling.

The demonstrated versatility of LLM-based agents across various domains motivates their exploration within everyday environments like smartphones and IoT devices. Their deployment in pervasive computing settings offers solutions for complex decision automation, enhanced user interaction, and improved accessibility. This potential is supported by recent advances, including compact and efficient LLMs [92, 120], increased memory in embedded systems, power-efficient processors, specialised hardware such as NPUs, and high-speed communication technologies like 5G. The following sections explore the pervasive computing landscape, the evolving role of AI within it, and the integration of LLM-based agents in pervasive environments.

3 Pervasive Computing: an Overview

Pervasive computing, also referred to as ubiquitous computing, was conceived in early seminal work as technology that would 'weave themselves into the fabric of everyday life until they are indistinguishable from it' [104]. This concept aims the creation of environments densely populated with computing and communication capabilities, yet seamlessly integrated with human users, until the technology effectively becomes "transparent" [74, 76]. So while distributed systems and mobile computing enabled "anytime, anywhere" access to information, though not always with guaranteed connectivity, pervasive computing fundamentally shifts towards an "all the time, everywhere" presence, ensuring seamless access to computing whenever and wherever it is needed [74]. Pervasive computing manifests in various devices, formats, and locations, ranging from resource-constrained sensors to high-performance servers, including cloud datacenters, mobile edge computing servers, mobile devices, TVs, wearables, sensors and embedded systems. These interconnected devices leverage various wireless communication technologies to enhance their capabilities while minimizing resource consumption, including battery power, memory, and CPU time leading to the proliferation of Internet of Things (IoT) devices. These IoT devices, endowed with sensing, computing, networking, and communication functionalities, are capable of collecting, analysing, and transmitting a diverse spectrum of data, including images, videos, audio, texts, wireless signals, and physiological signals from individuals and the physical environment. Furthermore sensors, integral to many IoT devices, play a crucial role by monitoring phenomena both within and beyond human perception, converting physical occurrences into numerical data. The size reduction, increased efficiency, enhanced sensitivity, and improved connectivity of these sensors have enabled their embedding in diverse environments and objects, providing real-time data and insights previously inaccessible. Therefore this increase in connectivity, data generation and accessibility of sensory data, contributes to the generation of zettabytes of real-time data streams [3] that can be used to create much more customizable and accurate systems but also can be challenging to manage. These pervasive systems find application across diverse sectors, ranging from Smart Homes and Smart Cities to Smart Factory and Healthcare. Cisco's projections[1] indicate over 2.6 billion cellular-connected IoT devices by 2026, showing the maturation of their capabilities and decreasing costs. Furthermore, the economic impact of IoT is projected to be substantial, with estimates suggesting a global value creation ranging from $5.5 trillion to $12.6 trillion by 2030 [19]. All these factors underscore the substantial and growing impact of pervasive computing environments in improving the quality of life.

3.1 Computational and Network Advancements

The increasing diffusion of pervasive computing is significantly driven by the advancements in both computational power and network technologies. Embedding computational power within everyday objects is fundamental to pervasive computing. This necessitates a range of microprocessor solutions tailored to specific device requirements. These range from low-power Microcontroller Units (MCUs) for basic sensing and actuation in resource-constrained devices, to more powerful Central Processing Units (CPUs) for complex processing tasks, and highly integrated Systems on a Chip (SoCs) for feature-rich applications. SoCs often incorporate specialised

[1] Cisco 5G IoT White Paper

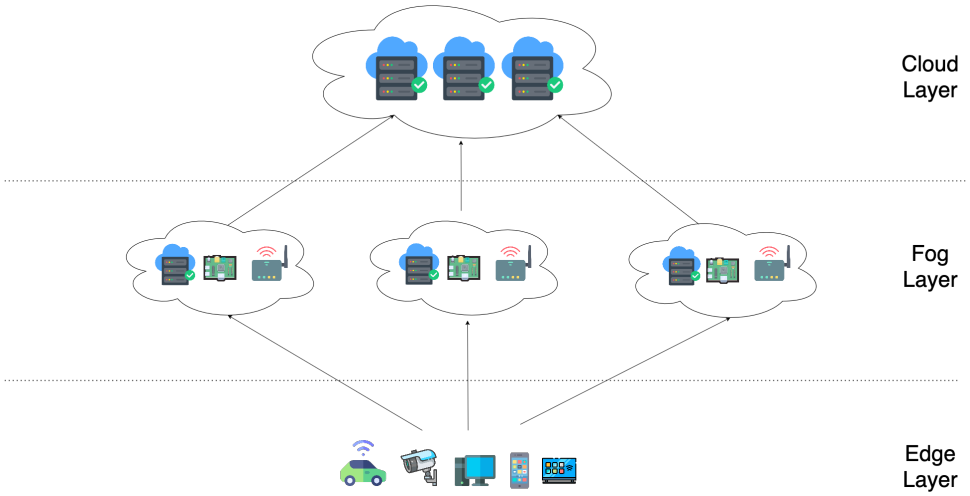

Fig. 4. The Cloud, Fog and Edge layers

processing units such as Graphics Processing Units (GPUs) and Digital Signal Processors (DSPs) to enhance multimedia and signal processing capabilities within compact form factors. Notably, the emergence of architectures like PlasticARM CPUs [5] presents a promising avenue for pervasive devices, including smartphones and edge computing nodes, offering an optimised balance of computational performance and energy efficiency crucial for their operation. This continuous evolution in microprocessor technology allows for increasingly sophisticated computational intelligence to be integrated directly into the fabric of everyday life. Also the increasing integration of Artificial Intelligence into edge devices within pervasive computing environments demands dedicated processing capabilities for computationally intensive AI tasks. Recognizing the limitations of general-purpose CPUs and GPUs for these specialised workloads, the development of dedicated AI accelerators has been a critical enabler. These specialised silicon solutions, often implemented as discrete co-processors or integrated within SoCs as Neural Processing Units (NPUs) [90], offer significantly enhanced performance and energy efficiency for neural network computations. The strategic inclusion of NPUs in SoCs designed for IoT, edge computing, and mobile platforms facilitates the rapid and efficient execution of complex AI algorithms directly on the device, minimizing latency and power consumption associated with cloud-based processing. The seamless operation of interconnected devices, a defining characteristic of pervasive computing, relies critically on robust and efficient networking infrastructure. Given the heterogeneity of devices and their diverse communication requirements, a suite of networking technologies is essential. While high-bandwidth applications benefit from advancements in Wi-Fi standards like Wi-Fi 7 with its focus on minimal latency, low-power devices leverage technologies such as ZigBee for personal area networks. Wide-area connectivity for distributed IoT deployments is facilitated by Low Power Wide Area Networks (LPWANs) like LoRaWAN [6]. Furthermore, high-speed cellular technologies like 5G and 6G provide ubiquitous connectivity for mobile and edge devices. This diverse and evolving network infrastructure provides the essential connectivity for the complex communication and coordination within pervasive computing environments.

3.2 Architectural Infrastructures

Pervasive computing leverages various architectural infrastructures to deliver its capabilities, primarily involving the cloud, fog, and edge computing paradigms (Figure 4). The cloud layer offers high computational power and

storage for complex tasks and centralised management. The fog layer, situated closer to the edge, facilitates distributed, latency-aware applications by providing local computing and network connectivity. Finally, the edge layer includes the end devices themselves, enabling processing and decision-making at the data source for real-time interactions. These architectures work in a complementary way to enable the collection, processing, and use of data generated by a multitude of interconnected devices.

3.2.1 Cloud Computing. As defined by the National Institute of Standards and Technology (*NIST*), cloud computing is '*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*' [61]. This paradigm includes centralised or distributed computing technologies operating over the Internet, primarily functioning as a scalable storage and processing infrastructure. Parallel and distributed computing models can be independently or jointly integrated and deployed within data centers, either physically or as virtualised resources. Within the context of pervasive computing, the cloud serves as a critical centralised repository and high-capacity processing hub for the extensive datasets generated by edge devices. Its inherent computational resources and scalability are paramount for managing high data flood [117]. Cloud service models are commonly categorised into private, community, public, and hybrid deployments [61]. Private clouds are dedicated to a single organization, regardless of who manages it or its location. Community clouds serve a specific group with shared needs, potentially managed by members or a third party, on or off-site. Public clouds are openly accessible to anyone and are owned and operated by providers on their premises. Hybrid clouds combine two or more distinct cloud types, linked by technology allowing data and application movement between them. Thanks to cloud computing, the connectivity of IoT devices to the cloud, and their integration with other related sensors, unlocks significant potentials:

- **Scalable Storage and Processing:** Cloud services provide the large-scale storage and high-performance distributed computing resources essential for managing the substantial data volumes generated by pervasive devices, effectively overcoming their inherent storage and processing limitations.
- **Stable Middleware Layer:** The cloud can establish a more robust middleware layer within the IoT architecture (positioned between IoT devices and applications) by centralizing service implementations.
- **Convenient and Cost-Effective Data Storage:** The convenience and economic benefits of cloud computing have driven widespread adoption by individuals and enterprises for storing data originating from pervasive devices.

Fundamentally, cloud computing signifies a transition from traditional, localised computing paradigms to a model characterised by flexible resource sharing and reduced operational costs. However, relying solely on cloud computing also presents challenges:

- **Latency Issues:** Network transmission delays or extensive response queuing can introduce significant latency, potentially hindering the performance of real-time applications.
- **Privacy Concerns:** Centralizing data in the cloud and transmitting sensitive personal information over the internet raises substantial privacy concerns. Users must be cognizant of the risks associated with cloud storage, including the security of their private data and the potential for data breaches.
- **High Bandwidth Costs:** Transmitting the large volumes of diverse data (text, video, images, audio, and IoT sensor readings) generated by pervasive devices to cloud data centers can incur substantial bandwidth costs and strain network infrastructure.

3.2.2 Fog Computing. The key goal of the fog layer, is to reduce the gap between the cloud layer, which has high resources, and the edge layer, which has limited resources. As defined by the National Institute of Standards and Technology (*NIST*), fog computing is '*a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources*' where '*facilitates the deployment of distributed, latency-aware applications and*

services, and consists of fog nodes (physical or virtual), residing between smart end-devices and centralised (cloud) services' [37]. Fog nodes can be seen as physical components such as Raspberry Pi, Nividia Jetson platforms, gateways, switches, routers, nano-servers, or virtual components such as virtualised switches, virtual machines, or cloudlets, bringing network, storage, and computing capabilities closer to the end users, and offering several unique features including reduced latency, geographical distribution, enhanced data security, and real-time processing [73, 117]. Fog computing can handle large volumes of data produced by various edge device types instead of sending it to a central cloud infrastructure, thereby addressing bandwidth and energy consumption problems. It possesses the ability to handle large volumes of data (potentially better than the cloud in terms of energy consumption), process data quickly, and produce high-quality results. Edge devices are typically located in close proximity or within a short distance of the fog layer, ensuring faster communication between these two tiers. However, fog computing also presents several challenges, including:

- **Device and Network Management:** Due to its decentralised and heterogeneous nature, managing randomly distributed network resources and a high-risk device breakdowns complicate connectivity and application deployment.
- **Computational Challenges:** The hierarchical structure of fog systems and their interaction with the cloud make optimal task allocation across IoT devices, fog nodes, and the cloud challenging. Ensuring computational correctness in such distributed environments is complex, further compounded by the need to choose suitable protocols for heterogeneous sensors and devices.
- **Security Challenges:** Heterogeneous devices, in less secure environments, can be susceptible to various attacks, including man-in-the-middle attacks.

3.2.3 Edge Computing. Sometimes referred to as an IoT network, performs computations closer to the edge of the network, where the data is generated. As defined by the National Institute of Standards and Technology (*NIST*), edge computing '*is the network layer including the end-devices and their users, to provide, for example, local computing capability on a sensor, metering or some other devices that are network-accessible*' [37]. Decentralizing intelligence, processing power, and communication resources to the network edge, this paradigm leverages a heterogeneous range of devices (including PCs, smartphones, smart devices, and automation controllers) situated in close proximity to the data-generating entity, thereby enhancing responsiveness and efficiency [117]. Furthermore, processing sensitive data locally on edge devices enhances user privacy and data ownership by minimizing the need for data transfer to the cloud. This localised processing is particularly advantageous for applications dealing with personal or proprietary information, such as virtual assistants and autonomous vehicles. Mobile Edge Computing (MEC) deploys services and computational capabilities at the edge of cellular networks, in close proximity to subscribers. This strategic placement enables a service environment characterised by ultra-low latency, high bandwidth, and direct access to real-time network information [46]. For instance in [88], the researchers handle data streams at the mobile edge to overcome the scalability problem of traditional IoT architectures, reducing traffic load in the core network and end-to-end delay for IoT services. The implementation and management of edge computing environments are associated with several challenges, notably resource and energy limitations. Individual edge devices inherently possess constrained computational power, memory capacity, and energy resources compared to centralised cloud servers, necessitating careful consideration on efficiency.

3.3 Pervasive AI Computing

The idea is to use the capabilities of AI systems inside a pervasive computing environment. As defined in [3] pervasive AI is the '*intelligent and efficient distribution of AI tasks and models over/among any types of devices with heterogeneous capabilities in order to execute sophisticated global missions*'. This paradigm marks a departure from traditional, cloud centralised AI approaches, leveraging the distributed computational resources inherent in

pervasive environments, including IoT devices and edge servers. Within this domain, AI-enabled sensors play a crucial role, categorised as: AIoT Sensors, facilitating cloud-based AI decision-making based on physical-world data; Edge AI Sensors, enabling localised AI inference at the device level; and TinyML Sensors, designed for efficient execution of specific tasks with minimal data [87]. The convergence of pervasive computing and artificial intelligence has thus established a novel research area, significantly enabled by advancements in Deep Learning.

3.3.1 Deep Learning. Deep Learning, an important subfield of Machine Learning that takes inspiration from biological nervous systems. It uses deep neural networks (DNNs), which are characterised by a high number of interconnected layers of neurons that learn features to accomplish a task. The multilayer perceptron, for instance, consists of fully connected neurons employing nonlinear activation functions. In contrast, convolutional neural networks (CNNs), prevalent in vision tasks, use convolutional layers. Each convolutional layer incorporates a set of learnable parameters, called filters, which possess the same number of channels as the input feature maps but with smaller spatial dimensions. Each filter channel convolves across the length and width of its corresponding input feature map, computing the inner product. The summation of these channel-wise inner products yields a single output feature map, and the total number of output feature maps corresponds to the number of applied filters. Another prominent architecture is the Transformer, which processes text or images encoded as vector embeddings (tokens). These models support a range of applications in autonomous systems, robotics, smart homes, and virtual reality. However, deploying DL models on resource-constrained edge devices requires balancing accuracy with efficiency. To address the challenges of deploying deep learning on resource-constrained pervasive devices, various strategies are being explored such as:

- **Distributed inference:** Splits DL models across devices to reduce local load and cloud latency, e.g. EdgeShard [135].
- **Federated learning (FL):** Trains models across devices while keeping data local, supporting privacy and scalability [67].
- **Optimization techniques:** Include caching, compression, and dynamic inference to improve DNN performance on constrained devices [86].

Within this context, Large Language Models (LLMs), are increasingly demonstrating their potential. When integrated into agent-based systems, they show strong potential for enhancing pervasive AI applications. The next section explores how LLM-based agents are applied in pervasive computing and the challenges of deploying them on limited-resource platforms.

4 Agents in Pervasive Computing

The integration of intelligent agents within pervasive computing environments represents a significant advancement in human-computer interaction, leading to more intuitive and proactive experiences. Pervasive computing, characterised by the integration of embedding of computational capabilities into everyday objects and environments, provides an ideal environment for the deployment of agents capable of perceiving, reasoning, and acting autonomously. As discussed in Section 2, the advent of Large Language Models (LLMs) has substantially augmented the potential of these agents by equipping them with advanced natural language understanding and generation capabilities, enabling their application in a wide range of use cases. This enhancement facilitates and increase the performance of this pervasive devices with more intuitive and natural interactions within smart environments, such as automatic voice-controlled smart homes [72], automatic smartphone interactions [105], or automatic traffic management in smart cities [11]. However, the deployment of LLM-powered agents in pervasive computing scenarios presents various challenges. As discussed in Section 3, one of the primary concern arises from the resource constraints inherent in many pervasive devices, including limitations in processing power and energy availability. This contrasts with the considerable computational, energy, and memory storage demands of LLMs. Moreover, the direct interaction of these agents with user data on pervasive devices underscores the critical

importance of user data privacy. Consequently, ensuring both the sustainable operation of these agents and the robust safeguarding of user data are essential for their responsible development and deployment. Addressing these challenges necessitates research into several key areas. This includes determining optimal deployment and alignment strategies for Large Language Models in both local and distributed pervasive environments. Also, research must focus on redesigning the core components of agents specifically memory, planning, reasoning, and acting modules to operate efficiently within the resource limitations inherent in pervasive computing. Furthermore, rigorous evaluation of agent performance within these specific pervasive environments is crucial to identify their strengths and weaknesses.

4.1 LLMs Deployment Strategies

Addressing the integration of Large Language Models (LLMs) into resource-constrained pervasive computing environments involves in two primary deployment strategies: local and distributed. The local deployment strategy aims to execute the LLM directly on the device where the user data is stored, minimizing latency and enhancing data privacy. However, this approach is constrained by the limited computational and resource capabilities of the local device. The distributed deployment strategy aims to augment computational resources through cloud computing servers, which offer enhanced processing power but introduce greater latency and raise privacy issues. Alternatively, this strategy involves partitioning the Large Language Model across multiple fog or edge devices, a method that strategically locates processing closer to the user's device to decrease latency, but increase system complexity.

4.1.1 Local LLM Deployment. Local deployment of LLMs aims to execute these powerful models directly on resource-constrained edge devices, offering significant advantages in terms of user privacy, cost-effectiveness by eliminating reliance on cloud infrastructure, and reduced latency due to on-device processing. However, the inherent limitations in computational power, energy availability, and storage capacity of these pervasive devices present substantial challenges to implement this vision. A key strategy to overcome these limitations involves optimizing LLM resource consumption through techniques such as small language models (SLMs) deployment, model quantization (to reduce the bit-precision of model weights), and pruning techniques (to reduce model parameters). For instance, researchers in [105] successfully implemented an agentic architecture using a Vicuna-7B, on a OnePlus ACE 2 Pro smartphone equipped with a Snapdragon 8 Gen2 CPU and Adreno™ 740 GPU. Building upon this, Autodroid v2 [106] demonstrated impressive performance in success rate and inference latency on the same resource-constrained device by deploying an 8-bit quantised Llama3.1-8B model. Further highlighting the potential of smaller models, Microsoft's Magma [120], an 8B parameter model, exhibited strong verbal and spatial-temporal reasoning in UI navigation and robotic manipulation tasks. Additionally, the MaViLa framework [25] showcased the effectiveness of smaller models by fine-tuning a Vicuna-13B model for smart manufacturing, achieving high performance in domain-specific tasks like additive manufacturing monitoring, anomaly detection, and autonomous process optimization. Despite the relative smallness of 7B or 13B parameter models compared to their larger 30B+ counterparts, their deployment on resource-constrained devices like typical smartphones (with 6-12 GB of RAM) remains a significant obstacle. As previously discussed, the feasibility of local LLM execution is heavily influenced by the model's parameter count and the memory footprint of the Key-Value (KV) cache. A Llama-2 7B model, for example, can require up to 28 GB for full precision inference, 7 GB with 8-bit quantization, and 3.5 GB with even 4-bit quantised versions, in addition to over 2 GB potentially needed for its 4k token context window's KV cache. While deployment might be viable on more powerful edge devices like NVIDIA Jetson Orin series [2] for developer and industrial applications, running such models efficiently alongside their KV cache on devices with less computational power, necessitates innovative memory management strategies. Furthermore, energy consumption is a critical concern, with research indicating an

[2]Jetson Orin Technical Specification

approximate cost of 0.1 J/token per billion parameters [56]. This suggests that a 7B parameter LLM could consume 0.7 J/token, potentially limiting continuous conversational usage. For instance a fully charged iPhone, with approximately 50 kJ of energy, can sustain this model in conversation for less than 2 hours at a rate of 10 tokens/s, with every 64 tokens draining 0.2% of the battery [56]. This highlights the urgent need for techniques that significantly enhance the energy efficiency of these models. To address these challenges, the literature presents various promising models and methodologies. Nexa AI's OmniVLM [15], a compact model with under one billion parameters (968M), directly tackles the memory footprint and computational demand issues, making it more easy to deploy. OmniVLM also introduces a novel token compression mechanism for visual inputs, achieving a substantial reduction in visual token sequence length, thereby lowering computational overhead while preserving visual-semantic information. Octopus v2 [12] uses another approach, enabling a 2 billion parameter on-device language model to outperform GPT-4 in function calling accuracy and latency while drastically reducing context length. This significantly enhances the feasibility of deploying AI agents directly on edge devices without cloud reliance. Building on this success, Octopus v3 [13] introduces a sub-billion parameter multimodal model capable of efficiently processing both visual and textual inputs using functional tokens and CLIP-based image encoding. Beyond model optimization, innovative deployment and runtime techniques are being explored. EdgeLLM [126] proposes a layerwise unified compression (LUC) method for dynamic pruning and quantization of LLM layers, coupled with adaptive layer tuning and voting. This approach achieves significant reductions in computation and memory demands, enabling fine-tuning and updating of LLMs even on smartphones with substantial speedups and reduced memory usage. Another strategy, explored in [124], involves hosting a single, stateful LLM as a system service within the mobile operating system, accessible to applications via system APIs. This design minimizes memory duplication and supports persistent context management through chunk-wise KV cache compression and tolerance-aware memory management. Context switching is accelerated via a swapping-recompute pipeline that overlaps I/O and computation. The system employs fine-grained memory eviction strategies, such as LCTRU (Least Compression-Tolerable and Recently-Used queue), adapted to the compression sensitivity of different parts of the model. Finally, EdgeMoE [123] faces the challenge of deploying extremely large and sparse Mixture-of-Experts (MoE) LLMs on mobile hardware. By treating device memory as a smart cache and selectively preloading only the most likely needed experts, EdgeMoE can execute models with over 10 billion parameters on small edge devices with minimal overhead. Its key innovations include expert-wise bitwidth adaptation and predictive expert caching, enabling real-time inference without exhausting device resources.

4.1.2 Distributed LLM Deployment. While various models and strategies exist for deploying LLMs directly on the edge device, interacting with users and data-generating sensors, this approach is often constrained by the necessity of using smaller, less performing models, and by the latency of the models on producing each token at inference time due to the hardware limitations. A key alternative to mitigate this issue is using a distributed approach. This strategy involves hosting the models either on cloud servers, which offer high computational and memory resources, or distributing the model's computation across multiple edge or fog nodes.

Cloud Distribution. Cloud computing offers the capability to host large open-source or proprietary LLMs on powerful servers, often accessed via Web Applications or APIs provided by major companies such as OpenAI (offering models like GPT-4, GPT-4o, o1, and o3), Anthropic (providing models like Claude 2.1, Claude 3.5 Sonnet, Claude 3.5 Haiku, and Claude 3.7), or Mistral(providing models like Mistral Large and Mistral Small). In this architecture, edge devices transmit user prompts to these remote servers hosting the LLMs. The server receives the prompt, processes it with the designated model, generates a response, and then sends the answer back to the originating edge device. Cloud computing proves advantageous when high-performance LLMs are required. However, as the servers and models are typically managed by external entities, careful consideration must be given to user privacy, the potential for connection instability, and response latency. Examples of cloud-based LLM applications include [72], where models like GPT-4 and Claude 2.1 are employed for reasoning and managing

smart home devices; MobileGPT [43] on smartphones, which uses GPT-4 to manage mobile applications; and Intrusion Detection System Agent [49], where GPT-4o reasons over network traffic data, generating and executing actions such as data preprocessing, classification, and knowledge retrieval for intrusion detection with detailed explanations. In scenarios where a single LLM may exhibit limitations across diverse domains, a multi-LLM approach can be employed. This involves using multiple LLMs from different cloud providers, managed by a router. Upon receiving a user request, the router intelligently directs the query to the model best specialised for that specific domain, optimizing the response quality. For instance, Nexa AI's Octopus v4 [14] is an LLM agent designed to run locally on the user's device. It analyzes incoming requests and determines the most appropriate model to handle them, preparing the prompt to maximise the chosen model's performance. This agent can route calls to both cloud-hosted and edge-hosted LLMs. Additionally, Division-of-Thoughts (DoT) [80] is a framework that combines SLMs with powerful cloud LLMs to efficiently handle complex tasks. It first decomposes a user's query into simpler sub-tasks using a Task Decomposer, exploiting the reasoning abilities of language models. A Task Scheduler then analyzes dependencies among sub-tasks to decide which ones can be executed locally and which need cloud support. A lightweight, plug-and-play Adapter helps the SLM decide task allocation without changing its core parameters. This collaboration reduces costs, boosts speed, and preserves reasoning quality.

Edge/Fog Distribution. Edge and fog computing deploys models closer to users to minimise communication latency, while simultaneously maintaining significant computational power by distributing workloads across multiple devices. This approach enhances user privacy and system modularity, as models operate within user-managed nodes, ultimately improving overall system responsiveness. Although lacking the computational power of cloud environments, these distributed architectures effectively harness the combined resources of numerous edge or fog nodes to distribute computation. This collaborative use of resources enables the deployment of larger, more capable models and the achievement of faster inference times compared to the constraints of a single edge device, as previously outlined. However, a significant challenge lies in the complexity of managing and maintaining the distributed nodes and orchestrating the computation across them, requiring considerable effort from the user. The literature presents several methodologies to address this distributed deployment paradigm. Some strategies focus on dynamic resource allocation and intelligent task distribution such as SpeziLLM [128] and Ai Flow [81]. SpeziLLM is an open-source framework that dynamically distributes LLM inference across decentralised fog and edge layers, in healthcare applications. By abstracting orchestration tasks like node selection, model placement, and task splitting, SpeziLLM simplifies the integration of LLMs into mobile and healthcare environments. It prioritizes the execution of sensitive data tasks on trusted local or fog nodes while offloading less critical or computationally intensive tasks to the cloud when necessary, balancing privacy, cost, and user experience through seamless model migration, fault tolerance, and flexible scaling. AI Flow redefines communication by focusing on "intelligence flow" rather than raw data transfer, adapting to dynamic network conditions to optimise inference across devices, edge nodes, and cloud servers. Instead of transmitting raw data, AI Flow sends only critical extracted features, significantly reducing communication overhead. It adaptively assigns portions of the inference task based on available computational resources, network bandwidth, and real-time conditions, ensuring low-latency responses and efficient model execution even in fluctuating environments. Other strategies work on collaborative inference among edge devices such as Distributed Mixture-of-Agents (MoA) [63]. This architecture enables multiple edge devices, each hosting a localised LLM, to collaborate through decentralised gossip protocols, achieving high-quality responses without a centralised server. Each device can independently process prompts and share intermediate results with neighboring devices using decentralized gossip algorithms. Devices act as "proposers" generating answers and "aggregators" refining or selecting the best response. This distributed setup enhances robustness, reduces latency, and improves answer quality compared to relying on a single device, while also ensuring queue stability despite resource limitations and varying workloads. Another approach is using the edge devices to model partitioning and distribution, cooperating as a single high performance edge device.

EdgeShard [135] is a method that facilitates efficient LLM inference by partitioning large models into smaller "shards" and distributing them across multiple edge devices. It intelligently selects devices and allocates model parts based on their computing power, memory, and network conditions, leveraging collaborative edge computing to reduce latency, bandwidth usage, and privacy risks. By employing dynamic programming algorithms for optimized device selection and task scheduling, EdgeShard enables large models like Llama2-70B to run efficiently even in heterogeneous, resource-limited environments, significantly improving inference speed and throughput without compromising model accuracy. In [140] the edges are used for a cooperative inference with terminal devices. In this framework efficient LLM inference is enabled by promoting collaboration between the user's device (terminal) and a nearby edge server. The terminal device rapidly generates speculative tokens using a lightweight model, while the edge server concurrently verifies and corrects them using a larger, more accurate LLM. This serial-parallel approach significantly reduces token generation delay and energy consumption by balancing the computational load between local and edge resources without heavy reliance on the cloud. An optimization algorithm manages model approximation and token generation to minimise delay and maintain high accuracy. Finally in distributed scenarios, guaranteeing service despite intermittent connectivity is crucial. In [145] the authors address this problem with a novel "Mixture of Attentions" architecture for speculative decoding. This method significantly enhances a local small model's autonomous prediction by integrating Layer Self-Attention and Cross-Attention, effectively using LLM activations when available, yet maintaining accuracy when disconnected. This approach boosts robustness, enabling continuous, accurate inference without persistent network access.

4.2 LLMs Alignment Strategies

LLMs are pretrained on vast quantities of internet text, equipping them with remarkable capabilities across a spectrum of tasks, including conversation, mathematics, logic, reasoning, translation, and coding. However, challenges emerge when LLMs, particularly within agentic frameworks, are tasked with operating in highly specialized or entirely novel domains, a common occurrence in pervasive environments. For instance, deploying an agent to autonomously interact with smartphone applications, many unseen during the model's pretraining, often results in poor task completion performance. Furthermore, the inherent resource constraints of pervasive computing often necessitate the use of smaller LLMs, which inevitably exhibit a notable performance decrease compared to their larger, more robust counterparts. To address these limitations, a crucial technique is LLM alignment. LLM alignment involves additional post-training of these models to specialise (align) them with the specific domains of deployment, thereby minimizing errors and hallucinations. The primary method for achieving domain-specific alignment in pervasive computing is model fine-tuning, broadly categorized into three main approaches: Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), and Federated Learning.

Supervised Fine Tuning. Supervised Fine-Tuning aligns a LLM to a specific domain by training it on a curated reference dataset comprising input and desired output, $\mathcal{Z} = [x_i, y_i]$, pairs relevant to the desired specialization. For instance, to align a model for mathematical tasks, the training data would consist of numerous examples of mathematical problems paired with their corresponding solutions. Regarding dataset creation, three primary methodologies are employed:

- Datasets can be constructed through human annotation of relevant examples.
- Alternatively, larger and more capable models can be used to generate task completions, with their interactions meticulously annotated until the desired outcome is achieved.
- Finally, another method involves human supervision of a large model, ensuring its adherence to the task and its correct progression towards the intended goal.

A common SFT approach involves full fine-tuning, where the model's initial weights, denoted as Φ_0, are updated to $\Phi_0 + \Delta\Phi$ through iterative gradient descent. This process aims to maximise the conditional language modeling

objective, as represented by:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t \mid x, y_{<t})\right)$$

where $\mathcal{Z}$ represents the training dataset of (input x, target output y) pairs, and $y_{<t}$ denotes the preceding tokens in the target output sequence.

However, a significant drawback of full fine-tuning is that each downstream task necessitates learning a new set of parameters, Φ, whose size is equivalent to the original model's parameter set, Φ_0. In resource-limited scenarios, such as pervasive computing environments, this approach becomes prohibitively complex and computationally expensive. To mitigate these challenges, Parameter-Efficient Fine-Tuning (PEFT) methods have emerged. These strategies are specifically designed to enable model adaptation even under stringent resource constraints. Among prominent PEFT techniques are Low-Rank Adaptation [34] (LoRA) and its quantized variant, QLoRA [23]. These methods enable efficient fine-tuning by learning a significantly smaller set of task-specific parameters, denoted as Θ , where $|\Theta| \ll |\Phi_0|$. Consequently, the task of determining the weight update $\Delta\Phi$ is reframed as an optimization problem over the smaller parameter set Θ:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t \mid x, y_{<t})\right)$$

This approach significantly reduces the memory and computational overhead associated with fine-tuning, making model adaptation more feasible in resource-constrained settings. For instance InfiGUI Agent [54] uses Full Supervised Fine-Tuning in two stages. In Stage 1, they collected diverse page contents and vision-language datasets, using screen coordinates, to train basic skills like page contents understanding and grounding. In Stage 2, they synthesized new data to teach two advanced reasoning skills: hierarchical reasoning (strategic + tactical task planning) and expectation-reflection reasoning (self-correction from outcomes). For dataset creation, they used both real page contents datasets and synthetic SFT data generated from multimodal LLMs. They focused on "Reference-Augmented Annotations" to precisely link visual elements and textual reasoning. ReachAgent [108] uses a first stage of Supervised Fine-Tuning (SFT). They build three datasets: Page Navigation, Page Reaching, and Page Operation. Each dataset focuses on different subtasks such as reaching pages, operating within pages, and navigating multi-step tasks. They generated tasks and step-by-step labels using various LLMs. The SFT trains the model to understand full page contents flows and improve multi-step planning before any reinforcement learning. This stage ensures the agent can solve subtasks accurately before optimizing for full-task preferences in later RL training. MaViLa's Supervised Fine-Tuning (SFT) to perform visual scene understanding, anomaly detection, and manufacturing reasoning using using LoRA on a Vicuna 13B model [25]. For dataset creation, they collected real-world and schematic manufacturing images, each manually captioned. They generated instruction-response pairs by prompting GPT-4, distinguishing between general and domain-specific questions. For domain-specific instructions, they used Retrieval-Augmented Generation (RAG) to ground answers in manufacturing knowledge. Instructions were classified by complexity, reasoning need, and domain specificity to ensure high-quality fine-tuning data. Finally in [125] researchers used Supervised Fine-Tuning (SFT) to customise LLMs for stable activity generation in smart home simulations. To create the fine-tuning dataset, they collected labeled examples of human-like daily schedules and activity outputs. Fine-tuning focused on ensuring structured outputs (like JSON) to prevent simulator crashes from unstructured LLM replies. The SFT enhanced the model's ability to generate realistic, context-aware daily activities for virtual smart home agents. As a result, they achieved a 4.3% improvement in simulation stability and efficiency compared to baseline prompting.

Direct Preference Optimization. Another effective approach for aligning LLMs through Fine-Tuning is Direct Preference Optimization (DPO) [70]. This method offers a more efficient approach to instruction-tuning. It directly

trains the LLM agent on pairs of preferred ("winner") and less-preferred ("loser") responses to an instruction. Critically, DPO achieves this by fine-tuning LLMs without the need for an explicit reward model, relying solely on these direct preference comparisons between output pairs. The DPO algorithm begins by sampling completions $y_1, y_2 \sim \pi_{\text{ref}}(\cdot \mid x)$ for each prompt x, and then labels these completions based on human preferences to construct an offline preference dataset:

$$\mathcal{D} = \left\{ \left(x^{(i)}, y_w^{(i)}, y_l^{(i)} \right) \right\}_{i=1}^{N}$$

Here, $x^{(i)}$ represents the i_{th} input prompt, $y_w^{(i)}$ denotes the preferred ("winner") output, and $y_l^{(i)}$ represents the "loser" output, as determined by human feedback. Then, the language model π_θ is optimized by minimizing the DPO loss:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{SFT}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{SFT}}(y_l \mid x)} - \beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{SFT}}(y_l \mid x)} \right) \right]$$

Since these datasets are typically sampled using a Supervised Fine-Tuned model π_{SFT}, the reference policy is often initialized as $\pi_{\text{ref}} = \pi_{\text{SFT}}$ when available. However, if π_{SFT} is not accessible, the initialization of π_{ref} is achieved by maximizing the likelihood of the preferred completions (x, y_w) within the preference dataset:

$$\pi_{\text{ref}} = \arg\max_{\pi} \mathbb{E}_{(x, y_w) \sim \mathcal{D}} \left[\log \pi(y_w \mid x) \right]$$

For instance ReachAgent [108] uses Direct Preference Optimization (DPO) to refine its decision-making in mobile GUI tasks. DPO is used without needing explicit numeric rewards, instead relying on preference pairs indicating which actions are better. These preferences are constructed using a 4-level reward ranking (Golden > Longer > Incomplete > Invalid) based on how effectively and efficiently page contents flows complete the task. During training, the model learns to prefer actions that lead to more optimal flows—those that are both task-completing and concise. This preference data is fed into the DPO loss function, which guides the policy (π_θ) to align closer to preferred behaviors while softly deviating from the supervised fine-tuned policy (π_{SFT}). This enhances the model's ability to generate efficient and successful page contents flows without needing exact matches to gold actions.

Federated Learning. Federated Learning (FL) is a machine learning paradigm that enables the training of a unified model across numerous decentralized edge devices or servers, each holding local data, without the need to exchange these sensitive datasets [6]. Essentially, FL allows for collaborative algorithm training on distributed data sources while preserving data locality. One of the core principle of FL is to empower devices, including smartphones and IoT devices, to collectively learn a shared predictive model, distributing the training computation between multiple devices. For instance in FedMobileAgent framework [100], federated learning was used to collaboratively train mobile agents across decentralized user data while preserving privacy. First each user locally collects data $D_k = \{\langle T, a_i, s_i \rangle\}_{i=1}^{n}$ via Auto-Annotation, where T is a task instruction, a_i an action, and s_i a screenshot. Then local training updates the model using stochastic gradient descent:

$$M_k^{(l,r+1)} = M_k^{(l,r)} - \eta \nabla \ell \left(M_k^{(l,\tau_k)}; T, s, a \right) \quad \text{where } M_k^{(l,\tau_k)} \text{ is sent to the server, } M^{(l+1)} = \sum_{k \in S^l} \omega_k M_k^{(l)}, \text{ and } \omega_k = \frac{n_k^*}{\sum_{k \in S^l} n_k^*}.$$

This two-level weighted aggregation balances episode and step diversity, improving learning from non-Independent and Identically Distributed data.

4.3 Agent Design Strategies

As discussed in Section 2, LLM-based agents use Large Language Models to perceive their environment, reason and plan into actionable subtasks, execute actions for each subtask, and iteratively refine their approach based on feedback. Furthermore, operating within a pervasive environment, often characterized by resource-constrained

devices or distributed systems such as cloud, fog, and edge nodes, introduces communication overhead and latency. Consequently, the design of an agent's architecture requires careful consideration not only of the underlying LLM deployment strategy but also of Agent module adaptation, which includes Memory, Reasoning, Planning and Action modules, as these significantly impact the agent's behaviour and overall performance within these constrained settings.

4.3.1 Memory. Memory is a critical component of intelligent agents, enriching their perception of the environment by providing crucial contextual information. It enables the agent to not only understand the immediate state of the external world but also access to historical data, including past execution trajectories, actions previously undertaken, errors encountered in prior attempts, and potentially effective solutions that can enhance future performance. In pervasive computing agents typically implement two primary forms of memory: Short-Term Memory and Long-Term Memory.

Short-Term Memory. This memory saves the intermediate steps within its current execution cycle. These ongoing steps are readily accessible at each decision-making stage, allowing the agent to reason about the next action in a context-aware manner. This mechanism significantly reduces the likelihood of the agent becoming trapped in repetitive states or endlessly cycling through the same sequence of actions. For instance in [96] the agent employs short-term memory to meticulously record all actions performed and the corresponding state changes as it interacts with the environment. This dynamically observed information then directly informs the next operational decisions. Also in [49] the agent uses short-term memory to maintain a log of the current session's context, including all prior reasoning steps, executed actions, and received observations. This memory is structured and iteratively updated after each tool execution, ensuring a coherent and up-to-date understanding of the ongoing situation. This allows the LLM to generate contextually relevant thoughts and actions at every stage of its processing pipeline. Crucially, this short-term memory is session-bound and discarded upon the completion of the inference process, guaranteeing real-time decision traceability without necessitating long-term storage of transient data.

Long-Term Memory. Once an agent concludes its execution, regardless of whether the final goal was achieved, all the intermediate steps and experiences accumulated during the session are consolidated. Through various summarization or simplification techniques, this information is then archived in Long-Term Memory for future reference and learning. For instance in [72] the agent leverages long-term memory to store a comprehensive history of user interactions with the agent. It employs a vector database, using the MiniLM embedding model for efficient storage and retrieval of semantically similar past executions when confronted with new tasks. Additionally, high-level summaries of user-agent interactions are stored to build a dynamic and holistic understanding of individual user preferences over time. Mobile-Agent-E [102] manages memory through a persistent long-term storage, focusing on two key elements derived from past experiences: Tips and Shortcuts. Tips represent generalizable insights gleaned from prior tasks, providing guidance for both high-level strategic planning and low-level action execution. Shortcuts are reusable sequences of actions identified for frequently occurring subroutines. Following each completed task, two "Experience Reflectors" analyse the entire interaction history to update existing Tips and Shortcuts or generate new ones. These refined or novel insights are then using by the "Manager" for future planning and the "Operator" for the next action execution. Finally in [43], the agent employs a hierarchical memory system that stores tasks as ordered sequences of subtasks and actions, directly linked to specific application screens. Each screen is represented as a node containing a set of available subtasks, with edges denoting transitions between subtasks triggered by related low-level actions. Tasks are saved in a parameterized function-call format, enabling flexible reuse across different contexts. During execution, the system can recall previously encountered tasks or subtasks and adapt them to new situations through attribute matching and in-context learning techniques. The memory is dynamically updated based on user feedback or the agent's own self-correction mechanisms.

As previously discussed, long-term memory serves as a repository of both successful and failed executions, enabling the agent to learn and improve over time. Two core strategies are commonly employed for populating this valuable resource:

- **Exploratory Phase:** Often, before being deployed for user interaction, an agent goes through a dedicated exploratory phase. During this stage, the agent is encouraged to actively explore its environment (e.g., the functionalities of various applications on a device). The primary objective is to develop a foundational understanding of the interaction mechanisms, identify the available tools and their functionalities, and learn the effects of different actions and strategies for overcoming specific challenges. This proactive exploration ensures that, when the agent is eventually used for real tasks, it already possesses a significant knowledge to more reliably navigate the environment and find the correct path to achieve its goals. For instance MobileGPT [43] analyzes app screens offline, extracting UI layout and simplifying it to HTML. The LLM identifies subtasks, formats them as function calls, and caches them for efficient live execution. AutoDroid [105] explores apps by random UI interactions, building a UI Transition Graph summarizing states and elements as simulated tasks, stored in memory. Also AutoDroid-V2 [106] constructs structured documents from page contents traces, abstracting states and transitions.
- **Test Phase:** In this scenario, the agent continuously learns from its interactions with the user during live testing. Both successful and failed execution attempts are stored in long-term memory. This continuous learning process allows the agent to gradually refine its strategies and improve its performance as the user engages in more tasks. For instance IDS Agent [49] uses long-term memory to resolve ambiguous situations by retrieving past sessions similar to the current one, incorporating their reasoning and outcomes for better decisions. Also the LiMeDa Framework [11] stores summarized data from completed vehicle tasks (route, time, energy) in memory, managing it to prevent overload. Upon receiving a new task, LiMeDa retrieves relevant past experiences to improve decision efficiency and avoid repeating errors.

4.3.2 Reasoning and Planning. Following the perception of the system state and the memory module's contents, including previous actions or complete past executions, the agent proceeds through Reasoning and Planning. The Reasoning step bring out a logical thinking process from the LLM, facilitating the formulation of an effective plan. For instance Mobile Agent-E [102] implements an Action-Reflection module that analyzes the state before and after an action, along with the action itself, to determine if the outcome aligns with the expected goal. This module categorizes action outcomes into: Successful or partially successful (outcome matches expectation), Failed because result leads to an incorrect state, and Failed because an action produces no observable change. Also InfiGUI Agent [54] operates in a three-step cycle: Reasoning (performing hierarchical reasoning), Action generation (writing the next action and its anticipated outcomes), and Reflection (analyzing the resulting state to evaluate if the expected results were achieved and generating a textual summary of this reflection). A popular reasoning technique is Chain of Thought (CoT) prompting [103] , that allows the agent to tackle complex reasoning tasks by breaking them down into a sequence of explicit steps, often initiated by prompts such as "Let's think step by step". For instance Mavila [25] , used for process automation, employs CoT reasoning to decompose tasks into a series of sequential steps and in the SAGE framework [72], planning (the decomposition of a high-level goal into substeps) is managed using CoT reasoning in conjunction with the ReAct [122] design pattern. Tool instructions and formatting guidelines encourage the LLM to first outline a plan before specifying the execution details. Also AutoDroid [105] fine-tunes a small language model (SLM) to reason with a zero-shot CoT approach, using a structured format.

The other step is Planning, where, informed by prior events and the reasoning process, the agent decides on the next action(s) to take. For instance in the IDS Agent [49], planning involves selecting the appropriate tool based on the information in memory and the analyzed traffic data. The agent then decides, based on classification results, whether to trigger an alert, thus identifying a potential security threat. Mobile Agent-E [102] features

a Manager module responsible for generating a plan broken down into subtasks. At each step, the Manager considers the initial user query, the current screenshot, the previous overall plan, the preceding subgoal, the current progress status, available Shortcuts from long-term memory, and any relevant notes. It then updates the overall plan and identifies the next immediate subgoal to pursue. Finally in [96] the agent employs three distinct agent roles for mobile device interaction. The Planning Agent determines the task progression using historical data. The Decision Agent observes relevant content from past screens via memory to generate actions and update the memory. The Reflection Agent evaluates if actions meet expectations by comparing screen states and initiates corrective re-execution if needed.

4.3.3 Action. The specific actions that an agent can perform within pervasive computing are highly dependent on its designated operational domain and the available tools or interfaces. Common categories of agent actions include:

- **Tool Use / API Calls:** Agents can leverage external services or system functionalities by using specific tools or making API calls. This includes a wide range of interactions, such as operating smartphone applications, sending emails, querying databases, retrieving weather information, or controlling external devices. For example, in [105], the agents possesses a defined set of possible actions (CLICK, CHECK/UNCHECK, SCROLL<DIRECTION>, INPUT<TEXT>) to interact with a smartphone device's interface. In [72], the agent flexibly manages smart home devices using a dedicated device interaction tool. The API calls are executed using the SmartThings REST API for reading device attributes and sending commands, with an automatic error correction mechanism in case of call failures. Similarly, in [49] the agent is equipped with a series of tools, including Knowledge Retrieval, Data Extraction, Classification, and Long-Term Memory Retrieval, which it can use for intrusion detection analysis. These tools are invoked using a JSON format that specifies an action name (the tool's identifier) and an action input (the associated settings or parameters for that tool).
- **Code Execution:** Agents may generate and execute code snippets to test conditions or directly interact with underlying systems. For example, in [106], the agent interacts with the environment by translating a user's natural language task into executable Python-like scripts. These scripts are generated by a small language model based on a structured app document and are then executed on the device to manipulate page contents elements, such as tapping buttons or scrolling content. Also, in [71], a large language model is employed to generate Python code that guides the placement of microservices between edge and cloud infrastructure, based on real-time workload and latency data. This enables the LLM to adaptively recommend whether a microservice should run on the edge or in the cloud, with new placement decisions being automatically integrated into the running application.
- **User Messaging (Natural Language Interaction):** Agents can also act by directly interacting with users through natural language messages, providing support as intelligent assistants. For instance, in [25], the agent communicates its analysis of manufacturing images to users, performing anomaly detection and scene understanding to provide comprehensive support to smart factory employees. Also, in [128], the system uses a unified interface to interact with users naturally via chat, enabling secure, real-time processing of health data in healthcare domains such as Electronic Health Record (EHR) analysis, patient data explanation, and medical form automation.

4.4 Applications and Evaluations of Pervasive Agents

As discussed in Section 3, while pervasive computing offers a multitude of applications, it still faces with challenges such as extracting and integrating heterogeneous data from diverse sources, translating complex natural language instructions into standardised actions executable across various tools and devices, and navigating intricate environments to accomplish complex objectives. These challenges can be effectively addressed by LLM-agents.

As detailed in Section 4, these systems possess a suite of features capable of overcoming these limitations and also unlocking a broad range of potential applications within the pervasive computing domain.

Smart Homes. Agents interpret user commands for device control via dynamic planning. For instance SAGE agent achieve 76% success rate on complex queries [72]. Multi-agent systems like CASIT enhance IoT deployments by coordinating sensor data and detecting anomalies, outperforming single agents in data-rich environments [142]. The MuRAL dataset aids socially aware LLM development for smart environments with richly annotated sensor data [16]. Frameworks like LLMind integrate LLMs with AI modules for multi-device orchestration via natural language commands translated into device control scripts [20].

Mobile Task Automation. Agents automate smartphone tasks by interpreting user instructions and executing page contents actions, eliminating manual interaction. AutoDroid [105] use UI understanding and exploration for autonomous task completion. InfiGUIAgent [54] employs human-like reasoning for robust UI interaction. MobileGPT [43] prioritizes efficiency with adaptable task memory and hybrid failure recovery. Benchmarks like Android Agent Arena (A3) [8], LlamaTouch [134], and MobileAgentBench [97] facilitate agent evaluation, while MobileSafetyBench [43] focuses on safe handling of sensitive operations.

Smart Health and Factories. Agents provide analysis of medical and industrial data. AutoHealth [7] integrates agents into wearables for Parkinson's monitoring. SpeziLLM [128] enables privacy-preserving medical AI on local nodes. In Smart Manufacturing, IMVA [52] and MaViLa [25] offer multimodal reasoning for tasks like quality control and system orchestration, improving decision-making and automation.

Smart Cities. Agents enable intelligent vehicle dispatching (LiMeDa) [11], realistic personal mobility generation (LLMob) [39], and natural language interaction with smart building management for energy optimization and control (BuildingSage) [22]. These systems demonstrate the transformative potential of LLM-based agents in creating intelligent and adaptive urban infrastructures.

5 Discussion

LLM-based agents represent a substantial advancement in Artificial Intelligence, enabling the tackling of highly complex tasks across diverse domains including software engineering, research, medicine, and robotics. As outlined in Section 2, these agents operate by perceiving their environment, reasoning and planning actions to achieve defined goals, and executing these plans through specific actions. This architecture typically uses a foundational model, either an LLM or a MLLM, and three essential modules: Memory, Planning, and Action. The Memory module stores necessary information, the Planning module designs action sequences, and the Action module generates individual steps. Pervasive computing stands out as a field for a practical application of this architecture. As detailed in Section 3, this domain focuses on enhancing human activity through integrated technology, providing computational power for a variety of applications that can significantly improve daily life. From smart homes and factories to the every day usage smartphones, embedding the computational power of agents within pervasive computing systems holds the potential to solve numerous challenges by autonomously handling previously intractable tasks. Section 4 introduced various strategies for integrating these agents into pervasive devices. These strategies are heavily influenced by the available resources on the deployment devices, and have facilitated the application of agents in pervasive computing, including personal assistants for health monitoring and device interaction, as well as building and traffic management in smart cities. However, several critical challenges remain under active research and warrant careful consideration.

Small/Quantized Models Limitations. A primary challenge is the performance degradation associated with the use of small or quantized models. As discussed in Section 4.1, pervasive computing scenarios often involve devices with limited computational power, memory, and energy, driven by connectivity constraints or the need

for on-device data privacy. This frequently necessitates the deployment of agents relying on Small Language Models (SLMs) or quantized models, which can exhibit a significant performance drop compared to larger, more robust alternatives. To address this, researchers are exploring post-training alignment methods like Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) to better tailor models to specific application domains, as well as techniques for distributing model computation across multiple edge nodes. Furthermore, even when employing larger models, the inherent limitations in LLM architecture pose challenges. For instance the issue of hallucinations, that is the generation of highly probable but irrelevant or factually inconsistent token sequences, poses a significant obstacle to the proper functioning of agents. Ongoing research seeks to mitigate this through Retrieval-Augmented Generation (RAG) systems, which inject relevant external information into the model's context, and post-training domain alignment techniques to better guide output coherence.

Generalization Problem. Another recognized limitation is the difficulty that LLMs face with generalization. As demonstrated in [62], even superficial changes like variable name alterations in mathematical problems can significantly degrade model performance, highlighting their reliance on training data patterns. This lack of robust generalization can be problematic for agents, particularly during the planning phase, potentially leading to suboptimal solutions or failure to follow task constraints, as emphasized in [40]. This study indicates that even advanced models like GPT-4o and Claude-3-opus struggle with planning tasks despite the use of techniques like ReAct [122] and Chain-of-Thought [103]. To address this, modular LLM frameworks that incorporate critique and reformulation mechanisms, as proposed in [30], show promising results. Additionally, enhancing the reasoning capabilities of models at inference time has led to the development of Large Reasoning Models (LRMs). These models, during inference, first engage in a reasoning phase, similar to "human-like thought", before providing an answer. This process appears to increase the likelihood of the model to generate a correct response. Various methodologies are being explored in this area, including forcing the model to produce rationales ("thinking") before giving the final answer [130], the use of Outcome-Supervised Reward Models (ORM) and Process Reward Models (PRM) to evaluate reasoning during training [32, 51, 133], and fine-tuning techniques that enable models to detect and self-correct errors using rationales [68]. The emergence of powerful LRMs like OpenAI's o1, o3, o4-mini, Anthropic's Claude 3.7 Sonnet, and DeepSeek's V3 underscores the active research and development in this area.

Memory Limitation. The external nature of the memory module in current agent architectures also presents a significant limitation. As agents interact with their environment, their experiences must be stored externally and then selectively reintroduced into the LLM's context window. The limited size of this context window poses a constraint, particularly for agents requiring numerous interactions to achieve a goal, potentially leading to the loss of crucial information if earlier interactions are simplified or overly summarized, as discussed in Sections 2.2.2 and 4.3.1. To overcome this, researchers are exploring strategies to find better summarization methods, or to expand the context window, as seen with the 1 Million context window Qwen's Qwen2.5-Turbo [118] and Google's Gemini 2.5 Pro. Another promising direction involves the direct integration of memory modules within the models themselves, such as the approach proposed by Google researchers in [4].

Energetic Issue. The energy issue is particularly significant when we are facing with agents that have to use frequently LLMs for reasoning, planning, action, and memory management tasks. This challenge must be addressed especially in pervasive environments, where there is a wide heterogeneity of devices, from large cloud high resources computing servers to smartphones that have limited energy resources. As discussed in Section 4.1, an LLM has a very high energy cost per billion parameters, which leads to rapid battery drain on edge devices and high electricity consumption when used on fog or cloud servers. To mitigate this issue, recent works have explored multiple strategies to reduce energy consumption without severely compromising performance. In [124] the authors implemented a KV cache compression and swapping method to avoid the recomputation of the KV

cache by the model. In [58] the researchers show that selecting models based on task complexity, and applying hardware-level techniques such as Dynamic Voltage and Frequency Scaling (DVFS), can cut energy usage by up to 50% without major accuracy loss. Another approach involves decentralized inference, where LLM layers are distributed across edge devices equipped with energy harvesting (e.g., solar panels). This allows sustained inference under energy constraints by dynamically scheduling computation based on device energy availability and predicted energy inflow [41]. Furthermore, in [75] the researchers tested multi-GPU inference and highlighted the substantial energy benefits of optimized LLM sharding strategies. In particular, using power capping strategies (limiting Watts usage), that increase the average inference time, save a lot energy without impacting the accuracy of the models. Finally, the GREEN-CODE [36] framework introduces dynamic early exiting during inference using reinforcement learning. This enables LLMs to terminate inference early at intermediate layers when sufficient confidence is obtained, achieving energy reductions of 23-50% in code generation tasks while maintaining output quality.

Privacy Issue. Finally, is fundamental to consider privacy in pervasive computing due to the close interaction with users and the consequent handling of large amounts of sensitive data. Agents operating in this context must manage this information while guaranteeing the user's complete privacy. For instance, when an agent interacts with applications on a smartphone or PC, it must exercise extreme caution in handling sensitive user data, including full names and the contents of private documents or files. The literature features various analyses and several benchmarks specifically designed to evaluate the security of agents in ensuring the preservation of user privacy. The PrivacyLens framework [82] provides a multi-level evaluation of LLM agents' awareness of contextual privacy norms. It introduces a pipeline that converts privacy-sensitive seeds into expressive vignettes and executable agent trajectories to uncover instances of private information leakage, such as sharing job-seeking details inappropriately, even with privacy-preserving prompts. PrivacyLens reveals that even advanced LLMs like GPT-4 leak sensitive information in a significant number of cases (up to 25.68%). AgentDojo [21] assesses agents robustness against prompt injection attacks. This framework puts LLM agents with realistic scenarios like email management or banking app navigation while defending against malicious inputs aimed at extracting user data. AgentDojo highlights the persistent vulnerability of agents to adversarial inputs and underscores the need for privacy-aware defenses. Finally, Agent-SafetyBench [139] offers a broader evaluation of agents' behavior across diverse environments, identifying privacy-related safety risks like unintentional data leakage. Evaluation across 2,000 test cases revealed that no agent achieved a safety score above 60%, highlighting a systemic lack of robustness and risk awareness in current LLM-based agent implementations.
Given these real privacy challenges that agents have to face, various strategies have been proposed to try to mitigate this issue. For instance, AgentDojo [21] incorporates and evaluates several defense mechanisms, including secondary attack detection modules like Data delimiters, Prompt injection detection, Prompt sandwiching, and Tool Filter. These strategies significantly reduce the success rate of prompt injection attacks, demonstrating a drop to 7.5% when a tool filter is active. However, no single defense mechanism offers complete protection against these vulnerabilities. AutoDroid [105] employs a Privacy Filter, a Personal Identifiable Information (PII) scanner, to detect sensitive information (such as names, phone numbers, and email addresses) within the prompt. This filter replaces any identified sensitive data, aiming to safeguard user privacy on smartphone devices. Furthermore, a layered defense strategy proposes three firewalls for agent and user data protection reducing private leakage from 70% to less than 2% [1]: *i)* a data firewall isolates private data in environment outputs using task-relevant rules, without accessing the conversation; *ii)* a trajectory firewall checks the agent's response against security rules post-decision, with an option to regenerate safer outputs; *iii)* an input firewall sanitizes external inputs by converting them to structured formats like JSON, removing manipulative language to reduce attack risks.

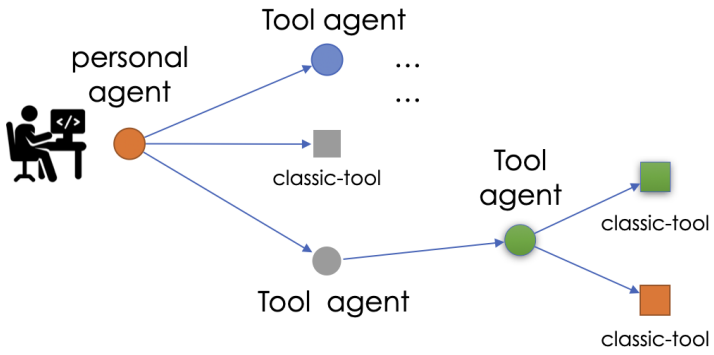

Fig. 5. A topology of classic tools and agent as a tool

6 Conclusion

This paper investigated the evolving landscape of research on LLM-based agents in the domain of pervasive computing. Initially, it provided a comprehensive analysis of agents, exploring their architecture, evaluation methodologies, and diverse applications across various fields. Afterwards, the paper introduced the pervasive computing field, outlining its infrastructures and wide-ranging applications. It then explored the growing integration of artificial intelligence within this domain, culminating in the integration of intelligent agents. Building on this foundation, the paper presented a detailed examination of agent-based architectures specifically designed for pervasive environments. This included an analysis of architectural adaptations, novel strategies introduced to suit the constraints of these contexts, and a review of implemented applications. The paper then proposed a discussion of the critical challenges in this field, alongside an overview of current research efforts aimed to address these limitations.

The field of artificial intelligence, is experiencing a period of exponential growth. Each month brings the release of more robust and effective models, driven by novel LLM architectures such as Mixture of Experts [38] and increasingly efficient post-training alignment techniques such as reinforcement learning techniques like GRPO [83], knowledge distillation, and fine-tuning methods like SFT and DPO. New powerful open-weight model families (e.g., DeepSeek r1, LLaMA 4, Qwen 3) and close-sources (e.g. Claude 4, OpenAI o1, o3) are released every few months. We are also observing the emergence of increasingly compact models that shows similar performances or even surpass larger state-of-the-art counterparts. For example, QwQ-32B demonstrates performance comparable to or better than models like DeepSeek-R1-Distilled-Qwen-32B, DeepSeek-R1-Distilled-LLaMA-70B, o1-mini, and the original DeepSeek-R1 [69]. This miniaturization is critical for pervasive deployments.

Substantial progress is also being made in hardware development. Innovations like the cost-effective NVIDIA Jetson Orin and the NVIDIA DGX Spark project, featuring compact devices capable of running models with up to 200 billion parameters, clearly signal a future where affordable, personalised agent hosting is accessible to individuals and companies [64]. Also the NVIDIA *CEO* Jensen Huang highlights this rapid advancement, stating, "*Our systems are progressing way faster than Moore's Law*", attributing this acceleration to integrated innovation across the entire stack of architecture, chip, systems, libraries, and algorithms [129]. This means that increasingly compact devices will offer greater computational power at lower costs.

Over the next 2–3 years, the exponential growth in both of these areas will profoundly impact the agent field in pervasive computing, driven by the increasing performance of Fog and Edge Computing devices. The continuous integration of greater computational capabilities into smaller, more energy-efficient, and affordable hardware

will enable the deployment of agents of various scales, dynamically adapting to user service requirements. Consequently, we will see a proliferation of specialized agents distributed across diverse fog and edge nodes, each providing tailored services based on their computational resources and the robustness of their embedded LLMs. This trajectory marks a definitive shift in pervasive computing: from an "*anytime, anywhere*" model to one that is "*all the time, everywhere*". The traditional "*anytime, anywhere*" agent approach relied on a limited number of high-performance computing (HPC) devices providing agent services, which were constrained by connectivity and response latency. In contrast, the emerging "*all the time, everywhere*" agent approach leverages the increasing capability of smaller devices, such as fog and edge nodes, to host agents. This ensures the continuous provision of agent services, virtually at all times. As detailed in Section 4.1, various strategies for deploying LLMs, the core of these agents, have been proposed, as seen in works like [63, 81, 135, 140], and research in this area is rapidly advancing. This future will be characterised by an increasingly interconnected network topology spanning cloud, fog, and edge devices. The enhanced performance of these edge components will enable the robust deployment of LLM-based agent services much closer to the user, facilitating faster response times and ensuring continuity of service even under unstable connectivity. Crucially, a smaller, edge-deployed agent can maintain essential functionality and deliver ongoing service in scenarios where reliance on cloud resources would be impractical or impossible. This vision lead to an increasing proliferation and usage of various agent services in multiple domains and applications, raising a crucial question:

Does the future of pervasive computing necessitate single, general-purpose agents capable of autonomously operating across numerous domains, or will it favour multiple, specialized agents, each expertly designed to solve a dedicated task upon request?

Consider a single agent attempting to manage both smartphone applications and smart home sensors. Achieving such broad expertise would demand computationally intensive, highly robust models and significant resources (a particular challenge in pervasive computing). Conversely, the demonstrated effectiveness of domain-specific fine-tuning suggests a more practical and scalable approach: deploying separate, specialised agents for distinct tasks. This lead to what we call "**Agent as a Tool**". Each specialised agent offers services that can be used like traditional software tools (Figure 5). In a realistic future scenario, each individuals can have a personal agent on their devices. These personal agents will be trained to interact with classic tools (e.g., email, calendar) [55, 110] and, critically, call other specialised agent-tools. For instance, a personal agent might seamlessly call upon a dedicated smart home agent for managing household systems or a software engineering agent for coding assistance within a development environment. The personal agent would intelligently decide which agent-tool to use for specific subtasks. Its overall task-handling capacity would depend on factors such as the robustness of its core LLM, the computational hardware constraints (edge, fog, cloud), and the accessibility of required classic-tools or agent-tools (e.g., network availability, physical location, co-hosting status). A calendar application, for example, might be locally accessible, while specialised company agent-tools might reside on remote edge or fog devices. This vision culminates in a complex, compound system comprising personal agents, classic tools, and specialised agent-tools. While these agents will operate semi-autonomously, proposing plans that align with the specialized task's workflow and executing corresponding actions, user interaction and guidance will remain crucial. To enhance agent performance and user experience, advanced memory strategies, such as summarizing past interactions, can be employed. This interaction data also presents a valuable opportunity for further fine-tuning or reinforcement learning techniques to improve agent performance [109]. Another paramount aspect is security. For critical actions (e.g., database modifications, code commits), essential safeguards requiring authorized user approval are indispensable for both personal agents and agent-tools [1, 66]. Robust firewall strategies must also be in place to protect sensitive user data. Communication between personal agents, tools, and agent-tools can leverage various established protocols, such as A2A[3] for agent-to-agent communication and the Model Context Protocol (MCP) [2]

[3] A2A, Agent Interoperability

for classic tool calls. This architecture's broad and inherent adaptability, applicable from optimizing industrial production to enhancing individual daily life, provides a significant opportunity to fundamentally reshape how businesses operate and to tangibly improve the daily experiences of people.

Acknowledgments

Funders: Fabio Ciravegna has received funding from the European Union's HORIZON programme, project "ICOS: Towards a functional continuum operating system", grant agreement No 101070177

References

[1] Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. 2025. Firewalls to Secure Dynamic LLM Agentic Networks. *arXiv preprint arXiv:2502.01822* (2025).

[2] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol Accessed via https://www.anthropic.com/news/model-context-protocol.

[3] Emna Baccour, Naram Mhaisen, Alaa Awad Abdellatif, Aiman Erbad, Amr Mohamed, Mounir Hamdi, and Mohsen Guizani. 2022. Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence. *IEEE Communications Surveys & Tutorials* 24, 4 (2022), 2366–2418.

[4] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. 2024. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663* (2024).

[5] John Biggs, James Myers, Jedrzej Kufel, Emre Ozer, Simon Craske, Antony Sou, Catherine Ramsdale, Ken Williamson, Richard Price, and Scott White. 2021. A natively flexible 32-bit Arm microprocessor. *Nature* 595, 7868 (2021), 532–536.

[6] Athanasios Bimpas, John Violos, Aris Leivadeas, and Iraklis Varlamis. 2024. Leveraging pervasive computing for ambient intelligence: A survey on recent advancements, applications and open challenges. *Computer Networks* 239 (2024), 110156.

[7] Luis Cardenas, Katherine Parajes, Ming Zhu, and Shengjie Zhai. 2024. Autohealth: Advanced llm-empowered wearable personalized medical butler for parkinson's disease management. In *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 0375–0379.

[8] Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149* (2025).

[9] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201* (2023).

[10] Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *Advances in Neural Information Processing Systems* 37 (2024), 74325–74362.

[11] Ruiqing Chen, Wenbin Song, Weiqin Zu, ZiXin Dong, Ze Guo, Fanglei Sun, Zheng Tian, and Jun Wang. 2024. An LLM-driven framework for multiple-vehicle dispatching and navigation in smart city landscapes. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2147–2153.

[12] Wei Chen and Zhiyuan Li. 2024. Octopus v2: On-device language model for super agent. *arXiv preprint arXiv:2404.01744* (2024).

[13] Wei Chen and Zhiyuan Li. 2024. Octopus v3: Technical report for on-device sub-billion multimodal ai agent. *arXiv preprint arXiv:2404.11459* (2024).

[14] Wei Chen and Zhiyuan Li. 2024. Octopus v4: Graph of language models. *arXiv preprint arXiv:2404.19296* (2024).

[15] Wei Chen, Zhiyuan Li, and Shuo Xin. 2024. OmniVLM: A Token-Compressed, Sub-Billion-Parameter Vision-Language Model for Efficient On-Device Inference. *arXiv preprint arXiv:2412.11475* (2024).

[16] Xi Chen, Julien Cumin, Fano Ramparany, and Dominique Vaufreydaz. 2025. MuRAL: A Multi-Resident Ambient Sensor Dataset Annotated with Natural Language for Activities of Daily Living. *arXiv preprint arXiv:2504.20505* (2025).

[17] Yiqun Chen, Lingyong Yan, Weiwei Sun, Xinyu Ma, Yi Zhang, Shuaiqiang Wang, Dawei Yin, Yiming Yang, and Jiaxin Mao. 2025. Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2501.15228* (2025).

[18] Steffi Chern, Ethan Chern, Graham Neubig, and Pengfei Liu. 2024. Can large language models be trusted for evaluation? scalable meta-evaluation of llms as evaluators via agent debate. *arXiv preprint arXiv:2401.16788* (2024).

[19] Michael Chui, Mark Collins, and Mark Patel. 2021. The Internet of Things: Catching up to an accelerating opportunity. (2021).

[20] Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, and Soung Chang Liew. 2024. Llmind: Orchestrating ai and iot with llm for complex task execution. *IEEE Communications Magazine* (2024).

[21] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[22] Volkan Dedeoglu, Qianggong Zhang, Yang Li, Jiajun Liu, and Subbu Sethuvenkatraman. 2024. BuildingSage: A safe and secure AI copilot for smart buildings. In *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 369–374.

[23] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems* 36 (2023), 10088–10115.

[24] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. 2024. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568* (2024).

[25] Haolin Fan, Chenshu Liu, Neville Elieh Janvisloo, Shijie Bian, Jerry Ying Hsi Fuh, Wen Feng Lu, and Bingbing Li. 2025. MaViLa: Unlocking new potentials in smart manufacturing through vision language models. *Journal of Manufacturing Systems* 80 (2025), 258–271.

[26] Maria Helena Franciscatto, Luis Carlos Erpen de Bona, Celio Trois, and Marcos Didonet Del Fabro. 2025. A CBR-based conversational architecture for situational data management. *Computer Speech & Language* 92 (2025), 101779.

[27] Stan Franklin and Art Graesser. 1996. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *International workshop on agent theories, architectures, and languages*. Springer, 21–35.

[28] Bingzheng Gan, Yufan Zhao, Tianyi Zhang, Jing Huang, Yusu Li, Shu Xian Teo, Changwang Zhang, and Wei Shi. 2025. MASTER: A Multi-Agent System with LLM Specialized MCTS. *arXiv preprint arXiv:2501.14304* (2025).

[29] Senay A Gebreab, Khaled Salah, Raja Jayaraman, Muhammad Habib ur Rehman, and Samer Ellaham. 2024. Llm-based framework for administrative task automation in healthcare. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 1–7.

[30] Atharva Gundawar, Karthik Valmeekam, Mudit Verma, and Subbarao Kambhampati. 2024. Robust Planning with Compound LLM Architectures: An LLM-Modulo Approach. *arXiv preprint arXiv:2411.14484* (2024).

[31] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. 2024. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578* (2024).

[32] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457* (2024).

[33] Yuki Hou, Haruki Tamoto, and Homei Miyashita. 2024. " my agent understands me better": Integrating dynamic human-like memory recall and consolidation in llm-based agents. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–7.

[34] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR* 1, 2 (2022), 3.

[35] Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435* (2024).

[36] Shashikant Ilager, Lukas Florian Briem, and Ivona Brandic. 2025. GREEN-CODE: Optimizing Energy Efficiency in Large Language Models for Code Generation. *arXiv preprint arXiv:2501.11006* (2025).

[37] Michaela Iorga, Larry Feldman, Robert Barton, Michael Martin, Nedim Goren, and Charif Mahmoudi. 2017. *The nist definition of fog computing*. Technical Report. National Institute of Standards and Technology.

[38] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[39] WANG JIAWEI, Renhe Jiang, Chuang Yang, Zengqing Wu, Ryosuke Shibasaki, Noboru Koshizuka, Chuan Xiao, et al. 2024. Large language models as urban residents: An llm agent framework for personal mobility generation. *Advances in Neural Information Processing Systems* 37 (2024), 124547–124574.

[40] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

[41] Aria Khoshsirat, Giovanni Perin, and Michele Rossi. 2024. Decentralized LLM Inference over Edge Networks with Energy Harvesting. *arXiv preprint arXiv:2408.15907* (2024).

[42] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. 2024. AutoWebGLM: A Large Language Model-based Web Navigating Agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5295–5306.

[43] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. 2024. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1119–1133.

[44] Elad Levi and Ilan Kadar. 2025. IntellAgent: A Multi-Agent Framework for Evaluating Conversational AI Systems. *arXiv preprint arXiv:2501.11067* (2025).

[45] Haoming Li, Zhaoliang Chen, Jonathan Zhang, and Fei Liu. 2024. LASP: Surveying the State-of-the-Art in Large Language Model-Assisted AI Planning. *arXiv preprint arXiv:2409.01806* (2024).

[46] Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. 2016. Mobile edge computing: Progress and challenges. In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*. IEEE, 83–84.

[47] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, et al. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems* 37 (2024), 100428–100534.

[48] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366* (2025).

[49] Yanjie Li, Zhen Xiang, Nathaniel D Bastian, Dawn Song, and Bo Li. 2024. IDS-Agent: An LLM Agent for Explainable Intrusion Detection in IoT Networks. In *NeurIPS 2024 Workshop on Open-World Agents*.

[50] Zijing Liang, Yanjie Xu, Yifan Hong, Penghui Shang, Qi Wang, Qiang Fu, and Ke Liu. 2024. A Survey of Multimodel Large Language Models. In *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering*. 405–409.

[51] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

[52] Chin-Yi Lin, Tsung-Han Tsai, and Tzu-Liang Tseng. 2025. Generative AI for Intelligent Manufacturing Virtual Assistants in the Semiconductor Industry. *IEEE Robotics and Automation Letters* (2025).

[53] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688* (2023).

[54] Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. 2025. InfiGUIAgent: A Multimodal Generalist GUI Agent with Native Reasoning and Reflection. *arXiv preprint arXiv:2501.04575* (2025).

[55] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, et al. 2024. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems* 37 (2024), 54463–54482.

[56] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*.

[57] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292* (2024).

[58] Paul Joe Maliakel, Shashikant Ilager, and Ivona Brandic. 2025. Investigating Energy Efficiency and Performance Trade-offs in LLM Inference Across Tasks and DVFS Settings. *arXiv preprint arXiv:2501.08219* (2025).

[59] Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. Roco: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 286–299.

[60] Andrea Matarazzo and Riccardo Torlone. 2025. A Survey on Large Language Models with some Insights on their Capabilities and Limitations. *arXiv preprint arXiv:2501.04040* (2025).

[61] Peter Mell, Tim Grance, et al. 2011. The NIST definition of cloud computing. (2011).

[62] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229* (2024).

[63] Purbesh Mitra, Priyanka Kaswan, and Sennur Ulukus. 2024. Distributed Mixture-of-Agents for Edge Inference with Large Language Models. *arXiv preprint arXiv:2412.21200* (2024).

[64] NVIDIA. 2025. *Nvidia DGX Spark*. Accessed May 2025.

[65] Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, et al. 2025. PlanGEN: A Multi-Agent Framework for Generating Planning and Reasoning Trajectories for Complex Problem Solving. *arXiv preprint arXiv:2502.16111* (2025).

[66] Shishir G Patil, Tianjun Zhang, Vivian Fang, Roy Huang, Aaron Hao, Martin Casado, Joseph E Gonzalez, Raluca Ada Popa, Ion Stoica, et al. 2024. Goex: Perspectives and designs towards a runtime for autonomous llm applications. *arXiv preprint arXiv:2404.06921* (2024).

[67] Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. 2025. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials* (2025).

[68] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2024. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems* 37 (2024), 55249–55285.

[69] Qwen Team. 2025. *QWQ-32B: Embracing the Power of Reinforcement Learning*. Accessed May 2025.

[70] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2023), 53728–53741.

[71] Kunal Rao, Giuseppe Coviello, Priscilla Benedetti, Ciro Giuseppe De Vita, Gennaro Mellone, and Srimat Chakradhar. 2024. Eco-llm: Llm-based edge cloud optimization. In *Proceedings of the 2024 Workshop on AI For Systems*. 7–12.

[72] Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Xue Liu, and Gregory Dudek. 2024. AIoT Smart Home via Autonomous LLM Agents. *IEEE Internet of Things Journal* (2024).

[73] H Sabireen and VJIE Neelanarayanan. 2021. A review on fog computing: architecture, fog with IoT, algorithms and research challenges. ICT Express 7 (2): 162–176.

[74] Debashis Saha, Amitava Mukherjee, Somprakash Bandyopadhyay, Debashis Saha, Amitava Mukherjee, and Somprakash Bandyopadhyay. 2003. Pervasive computing. *Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems* (2003), 1–37.

[75] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. 2023. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–9.

[76] Mahadev Satyanarayanan. 2001. Pervasive computing: Vision and challenges. *IEEE Personal communications* 8, 4 (2001), 10–17.

[77] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.

[78] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227* (2025).

[79] Orit Shaer, Angelora Cooper, Osnat Mokryn, Andrew L Kun, and Hagit Ben Shoshan. 2024. AI-Augmented Brainwriting: Investigating the use of LLMs in group ideation. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.

[80] Chenyang Shao, Xinyuan Hu, Yutang Lin, and Fengli Xu. 2025. Division-of-thoughts: Harnessing hybrid language model synergy for efficient on-device agents. In *Proceedings of the ACM on Web Conference 2025*. 1822–1833.

[81] Jiawei Shao and Xuelong Li. 2025. AI Flow at the Network Edge. *IEEE Network* (2025).

[82] Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. 2024. Privacylens: Evaluating privacy norm awareness of language models in action. *arXiv preprint arXiv:2409.00138* (2024).

[83] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).

[84] Chen Shen, Jin Wang, Sajjadur Rahman, and Eser Kandogan. 2024. Demonstration of a Multi-agent Framework for Text to SQL Applications with Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 5280–5283.

[85] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv. org/abs/2303.11366* (2023).

[86] Shakhrul Iman Siam, Hyunho Ahn, Li Liu, Samiul Alam, Hui Shen, Zhichao Cao, Ness Shroff, Bhaskar Krishnamachari, Mani Srivastava, and Mi Zhang. 2025. Artificial intelligence of things: A survey. *ACM Transactions on Sensor Networks* 21, 1 (2025), 1–75.

[87] Mona Sloane, Emanuel Moss, Susan Kennedy, Matthew Stewart, Pete Warden, Brian Plancher, and Vijay Janapa Reddi. 2025. Materiality and risk in the age of pervasive AI sensors. *Nature Machine Intelligence* (2025), 1–12.

[88] Xiang Sun and Nirwan Ansari. 2016. EdgeIoT: Mobile edge computing for the Internet of Things. *IEEE Communications Magazine* 54, 12 (2016), 22–29.

[89] Annalisa Szymanski, Noah Ziems, Heather A Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A Metoyer. 2025. Limitations of the LLM-as-a-Judge approach for evaluating LLM outputs in expert knowledge tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*. 952–966.

[90] Tianxiang Tan and Guohong Cao. 2021. Efficient execution of deep neural networks on mobile devices with npu. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*. 283–298.

[91] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems* 37 (2024), 51963–51993.

[92] Qwen Team. 2025. QwQ-32B: Embracing the Power of Reinforcement Learning. https://qwenlm.github.io/blog/qwq-32b/

[93] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[94] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).

[95] Han Wang, An Zhang, Nguyen Duy Tai, Jun Sun, Tat-Seng Chua, et al. 2024. ALI-Agent: Assessing LLMs' Alignment with Human Values via Agent-based Evaluation. *Advances in Neural Information Processing Systems* 37 (2024), 99040–99088.

[96] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014* (2024).

[97] Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. 2024. MobileAgentBench: An Efficient and User-Friendly Benchmark for Mobile LLM Agents. *arXiv preprint arXiv:2406.08184* (2024).

[98] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[99] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540* (2022).

[100] Wenhao Wang, Zijie Yu, William Liu, Rui Ye, Tian Jin, Siheng Chen, and Yanfeng Wang. 2025. FedMobileAgent: Training Mobile Agents Using Decentralized Self-Sourced Data from Diverse Users. *arXiv preprint arXiv:2502.02982* (2025).
[101] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.
[102] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-Agent-E: Self-Evolving Mobile Assistant for Complex Tasks. *arXiv preprint arXiv:2501.11733* (2025).
[103] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[104] Mark Weiser. 1991. The Computer for the 21 st Century. *Scientific american* 265, 3 (1991), 94–105.
[105] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 543–557.
[106] Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, et al. 2024. AutoDroid-V2: Boosting SLM-based GUI Agents via Code Generation. *arXiv preprint arXiv:2412.18116* (2024).
[107] Jinyang Wu, Mingkuan Feng, Shuai Zhang, Feihu Che, Zengqi Wen, and Jianhua Tao. 2024. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478* (2024).
[108] Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. [n. d.]. ReachAgent: Enhancing Mobile Agent via Page Reaching and Page Operation. ([n. d.]).
[109] Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. 2025. CollabLLM: From Passive Responders to Active Collaborators. *arXiv preprint arXiv:2502.00640* (2025).
[110] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. 2024. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems* 37 (2024), 25981–26010.
[111] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.
[112] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622* (2024).
[113] Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. 2024. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161* (2024).
[114] Jiahong Xu, Zhiwei Zheng, and Zaijun Wang. 2024. LAC: Using LLM-based Agents as the Controller to Realize Embodied Robot. In *2024 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 1894–1899.
[115] Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, et al. 2024. Crab: Cross-environment agent benchmark for multimodal language model agents. *arXiv preprint arXiv:2407.01511* (2024).
[116] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110* (2025).
[117] Jameel S Yalli, Mohd H Hasan, and Aisha Badawi. 2024. Internet of things (iot): Origin, embedded technologies, smart applications and its growth in the last decade. *IEEE access* (2024).
[118] An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. 2025. Qwen2. 5-1M Technical Report. *arXiv preprint arXiv:2501.15383* (2025).
[119] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems* 37 (2024), 50528–50652.
[120] Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai, Seonghyeon Ye, Joel Jang, et al. 2025. Magma: A Foundation Model for Multimodal AI Agents. *arXiv preprint arXiv:2502.13130* (2025).
[121] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems* 35 (2022), 20744–20757.
[122] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.
[123] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. 2025. EdgeMoE: Empowering Sparse Large Language Models on Mobile Devices. *IEEE Transactions on Mobile Computing* (2025).
[124] Wangsong Yin, Mengwei Xu, Yuanchun Li, and Xuanzhe Liu. 2024. Llm as a system service on mobile devices. *arXiv preprint arXiv:2403.11805* (2024).

[125] Haruki Yonekura, Fukuharu Tanaka, Teruhiro Mizumoto, and Hirozumi Yamaguchi. 2024. Generating Human Daily Activities with LLM for Smart Home Simulator Agents. In *2024 International Conference on Intelligent Environments (IE)*. IEEE, 93–96.

[126] Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[127] Ling Yue, Sixue Xing, Jintai Chen, and Tianfan Fu. 2024. Clinicalagent: Clinical trial multi-agent system with large language model-based reasoning. In *Proceedings of the 15th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. 1–10.

[128] Philipp Zagar, Vishnu Ravi, Lauren Aalami, Stephan Krusche, Oliver Aalami, and Paul Schmiedmayer. 2025. Dynamic fog computing for enhanced llm execution in medical applications. *Smart Health* (2025), 100577.

[129] Maxwell Zeff. 2025. Nvidia CEO Says His AI Chips Are Improving Faster Than Moore's Law. https://techcrunch.com/2025/01/07/nvidia-ceo-says-his-ai-chips-are-improving-faster-than-moores-law/. TechCrunch.

[130] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629* (2024).

[131] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. 2024. Proagent: building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17591–17599.

[132] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. 2024. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762* (2024).

[133] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240* (2024).

[134] Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024. Llamatouch: A faithful and scalable testbed for mobile ui task automation. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[135] Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal* (2024).

[136] Weinan Zhang, Junwei Liao, Ning Li, and Kounianhua Du. 2024. Agentic information retrieval. *arXiv preprint arXiv:2410.09713* (2024).

[137] Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2024. WebPilot: A Versatile and Autonomous Multi-Agent System for Web Task Execution with Strategic Exploration. arXiv:2408.15978 [cs.AI] https://arxiv.org/abs/2408.15978

[138] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501* (2024).

[139] Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024. Agent-SafetyBench: Evaluating the Safety of LLM Agents. *arXiv preprint arXiv:2412.14470* (2024).

[140] Wentao Zhao, Wenpeng Jing, Zhaoming Lu, and Xiangming Wen. 2024. Edge and terminal cooperation enabled llm deployment optimization in wireless network. In *2024 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE, 220–225.

[141] Zhonghan Zhao, Wenhao Chai, Xuan Wang, Boyi Li, Shengyu Hao, Shidong Cao, Tian Ye, and Gaoang Wang. 2024. See and think: Embodied agent in virtual environment. In *European Conference on Computer Vision*. Springer, 187–204.

[142] Ningze Zhong, Yi Wang, Rui Xiong, Yingyue Zheng, Yang Li, Mingjun Ouyang, Dan Shen, and Xiangwei Zhu. 2024. Casit: Collective intelligent agent system for internet of things. *IEEE Internet of Things Journal* (2024).

[143] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854* (2023).

[144] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934* (2024).

[145] Matthieu Zimmer, Milan Gritta, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. 2024. Mixture of Attentions For Speculative Decoding. *arXiv preprint arXiv:2410.03804* (2024).