DeepResearch: A Survey of LLM-based Research Agents

Jinyan Cai[1,*] and Ruochong Yao[2,*]

[1]School of Information Science and Technology, Yunnan Normal University, Kunming, China
[2]AI Products Department, China National Knowledge Infrastructure, Beijing, China
Email: 4176@ynnu.edu.cn (J.C.); yrc16401@cnki.net (R.Y.)
*Corresponding authors

Manuscript received November 14, 2025; accepted December 1, 2025; published June 30, 2026

***Abstract*—Large Language Models (LLMs) have demonstrated remarkable advances in natural language processing and content generation, yet they remain limited in supporting complex research tasks due to hallucinations, shallow summarization, weak multi-step reasoning, and unverifiable outputs. To address these challenges, the concept of *DeepResearch* has recently emerged, referring to research-oriented agents built on long-context LLMs and augmented with capabilities such as multi-step reasoning, deep retrieval, grounding and citation, agentic orchestration, and structured generation. These mechanisms enable models to act as autonomous researchers, capable of planning, retrieving, and producing systematic, evidence-based reports. Representative systems include Google Gemini Deep Research and OpenAI Deep Research, while in China platforms such as Kimi, Baidu Wenxin, Doubao, and DeepSeek are actively developing localized solutions. This paper provides a taxonomy of core techniques, presents a layered system architecture and architectural paradigms, reviews representative implementations and applications, and highlights open challenges and future directions. By consolidating current progress, we aim to guide the development of reliable and trustworthy DeepResearch agents.**

Keywords— DeepResearch; Large Language Models; Research Agents; Retrieval-Augmented Generation; Agentic Orchestration

I. Introduction

A. Background & Motivation

Large Language Models (LLMs) have achieved remarkable progress in natural language processing, question answering, and content generation. However, despite these advances, they remain limited when applied to complex research tasks. Specifically, existing models suffer from:

- **Shallow summarization** that fails to capture nuanced or domain-specific knowledge.
- **Hallucinations** leading to inaccurate or fabricated information.
- **Lack of multi-step reasoning**, preventing models from handling complex, decomposable tasks.
- **Unverifiable outputs**, as generated text often lacks citations or grounding in reliable sources.

These shortcomings make current systems insufficient for domains that demand systematic evidence, such as scientific discovery, legal analysis, or policy evaluation. As the demand for AI systems capable of supporting rigorous research grows, overcoming these issues has become an urgent challenge.

B. DeepResearch Definition

DeepResearch refers to a new class of *LLM-based research agents* designed to automate desk research tasks that traditionally require hours of human effort, condensing them into minutes. The goal is to produce **structured, evidence-based reports** enriched with **multi-step reasoning**, **targeted retrieval**, and **higher-order synthesis** [1].

C. Core Capabilities

A DeepResearch agent typically integrates five key capabilities:

- **Multi-step reasoning:** Decomposing complex research questions and planning stepwise inference paths.
- **Deep retrieval (RAG++):** Going beyond vector recall by performing iterative search, filtering, and validation to locate reliable evidence.
- **Grounding & citation:** Generating verifiable statements with explicit references to sources.
- **Agentic orchestration:** Coordinating external tools (e.g., web search, APIs, databases) to accomplish subtasks.
- **Structured synthesis:** Transforming disparate retrieval results into systematic long-form outputs such as reports, tables, or knowledge graphs.

D. Distinction from Standard LLMs

Unlike conventional conversational assistants, which mainly provide single-turn answers or shallow summaries, DeepResearch agents behave like autonomous research assistants: they not only generate text but also conduct iterative planning, perform reliable retrieval, and output auditable, evidence-grounded artifacts. This makes DeepResearch particularly suited for domains such as science, law, and finance, where systematic evidence and verifiability are indispensable.

E. Position in the AI Landscape

As such, DeepResearch is increasingly viewed as a key instantiation of general-purpose AI agents in high-verifiability domains, underpinning representative systems such as *Google Gemini Deep Research*, *OpenAI Deep Research*, *Perplexity Deep Research*, and localized platforms such as *Kimi*, *Baidu Wenxin*, *Doubao*, and *DeepSeek*.

F. Problem Formulation

We formulate DeepResearch as transforming an open-ended research question into a systematic, evidence-based artifact. Unlike single-turn QA, DeepResearch requires iterative planning, multi-hop retrieval, explicit grounding/citations, and structured generation (reports, tables, graphs), under practical constraints of latency, cost, and reliability.

DOI: 10.7763/IJET.2026.V18.1350

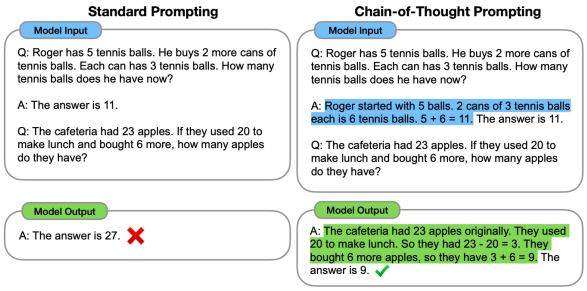


Fig. 1: Comparison between Standard Prompting and Chain-of-Thought (CoT) Prompting [2]. CoT encourages explicit intermediate reasoning, reducing errors on complex multi-step problems.

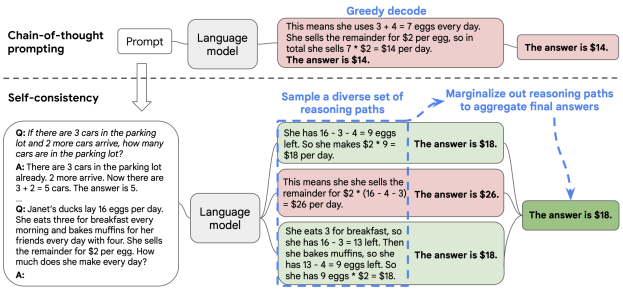


Fig. 2: Comparison between standard Chain-of-Thought prompting and Self-Consistency. By sampling diverse reasoning paths and aggregating answers, Self-Consistency improves robustness against spurious reasoning chains [3].

G. Contributions

This survey: (i) introduces a taxonomy of core technical capabilities; (ii) proposes a *four-layer system view* mapping capabilities to engineering layers; (iii) contrasts *architectural paradigms* (pipeline, multi-agent, hybrid) with pros/cons; (iv) compares representative systems (industrial and open-source); and (v) outlines challenges and future directions.

II. Core Technical Capabilities

A. Multi-step Reasoning

1) Chain-of-Thought (CoT)

Chain-of-Thought (CoT) prompting was introduced to elicit explicit intermediate reasoning steps from large language models (LLMs) by encouraging them to "think step by step" [2]. The key idea is that complex problems can often be decomposed into a sequence of simpler sub-problems, and by generating intermediate steps in natural language, the model can make its reasoning process explicit. This paradigm transforms the prediction target from a single direct answer into a structured reasoning trajectory followed by a final solution.

CoT can be incorporated at two different stages: (i) *during training*, where supervised fine-tuning or reinforcement learning is used to optimize models to produce coherent reasoning chains; and (ii) *during inference*, where prompting techniques are applied without additional training to elicit step-by-step reasoning from pretrained models. For example, zero-shot CoT prompting simply appends the phrase "Let's think step by step" to a question, which has been shown to reliably trigger multi-step reasoning in sufficiently large LLMs.

This approach has demonstrated significant improvements in tasks involving arithmetic reasoning, symbolic manipulation, logical inference, and commonsense reasoning, largely because it provides a scaffolding for multi-step problem solving. In addition, CoT enhances interpretability, since the reasoning path is exposed to the user and can be inspected for plausibility. However, it is not without limitations: errors in earlier steps can cascade into incorrect final answers, reasoning chains are often verbose and redundant, and automatic evaluation of reasoning quality remains challenging. Subsequent research has therefore focused on extensions such as self-consistency, least-to-most prompting, and tree-structured reasoning to address these weaknesses. This basic contrast is illustrated in Figure 1.

2) Self-Consistency

Self-Consistency is a refinement of Chain-of-Thought (CoT) prompting that aims to improve robustness by sampling **multiple diverse reasoning paths** instead of relying on a single deterministic chain [3].

Concretely, instead of decoding greedily, the model uses *stochastic decoding* (e.g., nucleus sampling) to produce a set of different reasoning trajectories for the same input. The final answer is then obtained by **marginalizing over these reasoning paths**, typically through majority voting or answer aggregation. This approach mitigates the risk of spurious or faulty reasoning chains, since errors in one path may be outvoted by correct reasoning in others.

As illustrated in Figure 2, while a single CoT path may yield an incorrect result due to a local mistake, sampling multiple reasoning paths allows the model to converge towards the correct answer.

While Self-Consistency effectively increases robustness and delivers consistent improvements on reasoning benchmarks such as GSM8K and CommonsenseQA, it also introduces trade-offs. On the positive side, aggregating multiple reasoning paths provides a more reliable estimate of the model's reasoning distribution and helps correct spurious single-path errors. However, this robustness comes at the cost of substantially higher computational overhead, since multiple generations must be sampled for each query. Moreover, if the majority of sampled reasoning trajectories are systematically flawed, the aggregated answer can still be incorrect. Thus, Self-Consistency primarily reduces variance rather than fundamentally addressing hallucinations, but it represents a crucial step towards more advanced paradigms such as Tree-of-Thought reasoning and ReAct.

3) Tree-of-Thoughts (ToT)

Tree-of-Thoughts (ToT) generalizes Chain-of-Thought reasoning into a **search over a tree of reasoning paths** [4]. Instead of following a single chain (CoT) or sampling multiple independent chains (Self-Consistency), ToT explicitly models reasoning as a *branching process*, where each node corresponds to a partial reasoning state (a "thought") and edges represent possible continuations. This allows the model to explore multiple reasoning trajectories in parallel, backtrack when a path proves unpromising, and ultimately select the most promising reasoning path.

Concretely, ToT consists of three core components:

- **Thought Decomposition**: break down a problem into intermediate reasoning steps ("thoughts").
- **Thought Generation**: given a state, generate k candidate thoughts for the next step (via prompting).
- **State Evaluation**: evaluate partial states to decide which branches to expand or prune, using heuristics, value esti-

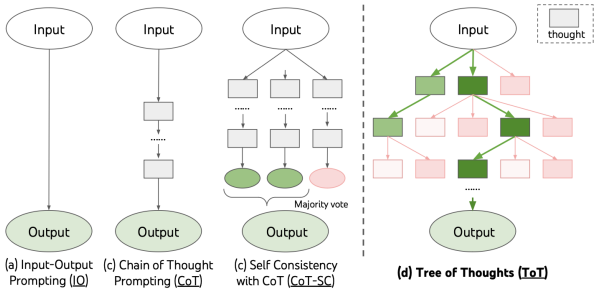


Fig. 3: Comparison of different reasoning paradigms. ToT generalizes CoT and Self-Consistency by framing reasoning as a search over a tree of thoughts [4].

Algorithm 1 ToT-BFS$(x, p_\theta, G, k, V, T, b)$

Require: Input x, LM p_θ, thought generator $G()$ & size limit k, states evaluator $V()$, step limit T, breadth limit b.
$S_0 \leftarrow \{x\}$
for $t = 1, \cdots, T$ **do**
  $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z \in \mathrm{G}(p_\theta, s, k)\}$
  $V_t \leftarrow V(p_\theta, S'_t)$
  $S_t \leftarrow \arg\max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$
end for
return $G(p_\theta, \arg\max_{s \in S_T} V_T(s), 1)$

Algorithm 2 ToT-DFS$(s, t, p_\theta, G, k, V, T, v_{th})$

Require: Current state s, step t, LM p_θ, thought generator $G()$ and size limit k, states evaluator $V()$, step limit T, threshold v_{th}
if $t > T$ **then** record output $G(p_\theta, s, 1)$
end if
for $s' \in G(p_\theta, s, k)$ **do** ▷ sorted candidates
  if $V(p_\theta, \{s'\})(s) > v_{thres}$ **then** ▷ pruning
    DFS$(s', t+1)$
  end if
end for

Fig. 4: Illustration of ToT search algorithms: (left) BFS expands multiple branches per step, pruning with breadth limit; (right) DFS explores deeply with backtracking based on evaluation thresholds [4].

mates, or voting mechanisms.

ToT can be combined with different search algorithms:

- **Breadth-First Search (BFS)**: expand multiple candidate states in parallel up to a breadth limit, then keep the best-scoring ones.
- **Depth-First Search (DFS)**: explore a single branch deeply, with pruning based on a threshold; if the branch fails, the algorithm backtracks.

As shown in Figure 3, ToT provides a unifying framework that subsumes CoT and Self-Consistency as special cases: CoT corresponds to a single path, while Self-Consistency corresponds to independent samples without structured search.

ToT can be implemented algorithmically as in Figure 4, where the search procedure iteratively expands and evaluates candidate reasoning paths using BFS or DFS strategies.

Overall, ToT improves upon CoT and Self-Consistency by enabling **lookahead, exploration, and backtracking** during reasoning. This makes it especially effective for combinatorial problems such as puzzle solving, creative writing, and planning. However, ToT also introduces higher computational cost due to tree expansion, and its effectiveness depends heavily on the quality of state evaluation heuristics.

4) ReAct

ReAct (**Reasoning + Acting**) is a framework that interleaves natural language reasoning traces with external actions, such as web search, database queries, or API calls [5]. Unlike purely internal reasoning paradigms (e.g., CoT, ToT), ReAct equips large language models (LLMs) with the ability to dynamically *think, act, observe, and then think again*. In this paradigm, the model not only generates intermediate reasoning steps, but also produces explicit `Action` commands that are executed by an external operator. The resulting `Observation` is incorporated back into the reasoning context, enabling iterative refinement until a final answer is reached.

Concretely, ReAct operates in the following loop:

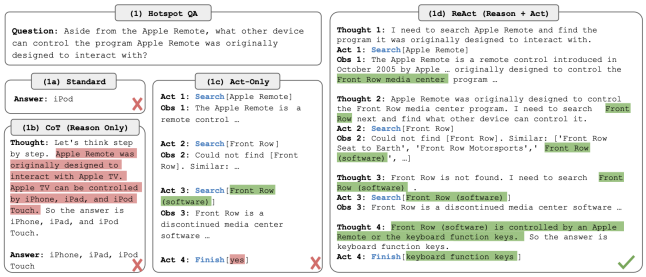


Fig. 5: Comparison of different paradigms: Standard prompting, CoT, Act-only, and ReAct. ReAct alternates between reasoning (`Thought`) and external operations (`Action--Observation`), enabling both interpretability and grounding [5].

- **Thought**: the LLM articulates its current reasoning state, decomposing the problem or planning the next step.
- **Action**: the LLM issues an explicit command (e.g., `Search[entity]`, `Lookup[string]`, `Finish[answer]`), which is passed to an external executor.
- **Observation**: the executor returns the outcome of the action, such as retrieved passages or API responses.
- The LLM then continues with a new **Thought**, incorporating the observation into its reasoning.

This iterative *Thought–Action–Observation* cycle allows ReAct to dynamically gather missing information, verify hypotheses, and adapt strategies during problem-solving. A schematic illustration is shown in Figure 5, where ReAct is contrasted with purely reasoning-based or purely action-based approaches.

Algorithmically, a minimal ReAct agent can be expressed as an iterative controller:

1) Initialize context with the input query.
2) Repeat until termination:
   - Generate a **Thought** and corresponding **Action**.
   - Execute the action with an external tool, return **Observation**.
   - Append `Thought`, `Action`, and `Observation` to the context.
3) If the action is `Finish[answer]`, return the final output.

Compared to CoT, ReAct improves **grounding**, since reasoning is anchored in retrieved evidence rather than hallucinated knowledge. Compared to Act-only approaches, it enhances **interpretability**, as reasoning traces expose the agent's decision process. However, ReAct's performance is highly dependent on the quality of external tools and the reliability of retrieval, and failure in action execution or noisy observations can propagate into flawed reasoning [5].

Overall, ReAct exemplifies the shift from static prompting to **interactive agentic reasoning**, where LLMs orchestrate both internal cognition and external actions in a unified loop. This paradigm is particularly effective for open-domain question answering, fact verification, and interactive decision-making tasks.

5) Program- and Tool-aided Reasoning (PAL/PoT)

Program-aided Language Models (PAL) and Program-of-Thoughts (PoT) represent another direction that integrates symbolic tools and program execution into reasoning [6], [7]. Rather than solving tasks entirely in natural language, the LLM decomposes a problem into structured steps and generates

Type	Input	Output	Definitions
Retrieve	x / x, y	{yes, no, continue}	Decides when to retrieve with $\mathcal{R}$
ISREL	x, d	{**relevant**, irrelevant}	d provides useful information to solve x.
ISSUP	x, d, y	{**fully supported**, partially supported, no support}	All of the verification-worthy statement in y is supported by d.
ISUSE	x, y	{**5**, 4, 3, 2, 1}	y is a useful response to x.

Fig. 6: Reflection tokens in Self-RAG [8]. Control tokens (e.g., Retrieve) decide when to perform retrieval, while critique tokens (IsRel, IsSup, IsUse) evaluate the relevance of documents, the factual support of answers, and the utility of responses.

executable code or formal tool calls, which are then run by external interpreters to guarantee correctness. This approach is especially beneficial for arithmetic, symbolic, and logic-heavy domains where exact computation is required. By combining the decomposition ability of LLMs with the determinism of external solvers, PAL/PoT achieves high reliability. However, applicability remains limited to domains with stable APIs or interpreters, and errors in code generation or execution can propagate to the final outcome.

B. Deep Retrieval (RAG++)

1) Self-RAG

Self-RAG (**Self-Reflective Retrieval-Augmented Generation**) extends classical RAG by introducing *reflection tokens* that allow the model to dynamically control retrieval and evaluate its own outputs [8]. Instead of always appending top-k documents as in vanilla RAG, Self-RAG equips the generator with special control tokens that trigger retrieval and critique during the generation process. This design makes the workflow conceptually similar to ReAct [5], where the model alternates between reasoning and acting. The key difference is that in Self-RAG, the "actions" (e.g., retrieval, verification, scoring) are integrated as tokens inside the generation stream, rather than issued as separate API calls. The reflection-token mechanism is summarized in Figure 6.

Concretely, reflection tokens fall into two categories:

- **Control tokens** (e.g., `<Retrieve:Yes/No/Continue>`) decide whether to query the retriever.
- **Critique tokens** (e.g., `<ISREL>`, `<ISSUP>`, `<ISUSE>`) evaluate retrieved passages and generated outputs, scoring their relevance, factual support, and usefulness.

Training: A Critic model is first trained with supervision from a strong teacher (e.g., GPT-4) to predict retrieval and critique labels. These signals are then injected into the training corpus, and a Generator model is trained with a standard LM objective to jointly produce natural language outputs and reflection tokens.

Inference: Only the Generator is used. It interleaves reasoning with reflection tokens: outputting `<Retrieve>` to trigger document retrieval, inserting the retrieved passages back into context, and then producing answers along with critique tokens (`<ISREL>`, `<ISSUP>`, `<ISUSE>`). The system aggregates multiple candidate outputs based on these tokens, yielding faithful and controllable responses.

Overall, Self-RAG can be viewed as a dynamic, self-reflective loop that parallels ReAct's "Thought–Action–Observation" cycle, but internalizes the "Action" stage as reflection tokens embedded in the LM's output space.

2) GraphRAG

GraphRAG extends standard Retrieval-Augmented Generation (RAG) by replacing the flat *Top-k passage retrieval* with a **graph-based subgraph retrieval** paradigm [9]. Instead of

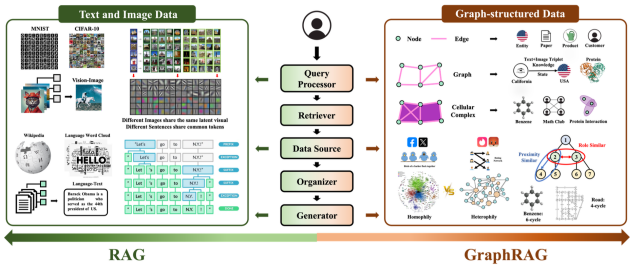


Fig. 7: Comparison between standard RAG (left, Top-k retrieval from text/image corpora) and GraphRAG (right, subgraph retrieval over structured graphs). GraphRAG enhances interpretability by providing explicit relational evidence [9].

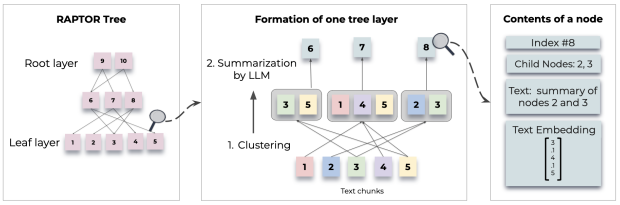


Fig. 8: Illustration of the RAPTOR tree construction [10]. Leaf nodes represent fine-grained text chunks. Through iterative clustering and LLM-based summarization, parent nodes provide increasingly abstracted summaries, forming a hierarchical tree representation of the corpus.

treating documents as isolated chunks, GraphRAG organizes external knowledge into nodes (entities, concepts) and edges (relations), and retrieves relevant subgraphs or reasoning paths for a given query. This allows the model to leverage relational structures for *multi-hop reasoning* and to provide **traceable, interpretable evidence**, as retrieved subgraphs directly indicate the supporting knowledge.

As illustrated in Figure 7, GraphRAG maintains the classic RAG pipeline—*query processor, retriever, organizer, generator*—but adapts the retriever to operate over graphs rather than only vector spaces. This design substantially enhances reasoning over complex relations (e.g., scientific, biomedical, and social graphs), while preserving the benefits of retrieval grounding. However, constructing and maintaining high-quality graphs remains costly, and errors in entity or relation extraction can propagate into downstream generation.

3) RAPTOR

Recursive Abstractive Processing for Tree-Organized Retrieval addresses the limitations of flat chunk-based retrieval in vanilla RAG by constructing a hierarchical *tree of summaries* over long documents [10]. Instead of treating each chunk as independent, RAPTOR recursively clusters and summarizes segments, producing a multi-layer tree where leaf nodes preserve fine-grained details and higher nodes capture broader themes.

Tree construction: As shown in Figure 8, RAPTOR proceeds in two recursive steps: (i) *clustering* text chunks into semantically similar groups, and (ii) *summarization* by an LLM to produce an abstractive node that represents each group. This process is repeated until a root node is formed, resulting in a tree structure that captures both local and global context.

Retrieval strategies: Given a query, RAPTOR supports two complementary modes. In the *tree traversal* strategy, retrieval starts from the root and iteratively descends through the hierarchy by selecting relevant nodes, eventually reaching

TABLE I: Comparison of RAG variants.

Method	Unit	Control	Advantage
RAG	Top-k passages	Static	Simple, effective
Self-RAG	Passages + tokens	LM reflection tokens	Dynamic, self-critique
GraphRAG	Graph subgraphs	Query → subgraph	Multi-hop, interpretable
RAPTOR	Tree summaries	Similarity (traverse/flat)	Multi-scale, long docs

the leaves that contain detailed evidence. In contrast, the *collapsed tree* strategy flattens all nodes and performs direct similarity search across the entire tree, enabling the model to simultaneously access both high-level summaries and fine-grained details.

Advantages: By leveraging hierarchical summaries, RAPTOR enables *multi-scale retrieval*, balancing global themes with specific evidence. It has demonstrated large gains on long-context QA tasks (NarrativeQA, QASPER, QuALITY), outperforming both dense retrievers and vanilla RAG by a significant margin.

Overall, RAPTOR exemplifies how recursive abstractive summarization transforms retrieval from flat Top-k passages into a **tree-structured, interpretable, and efficient** process, particularly suited for long-document reasoning.

4) Conclusion

Table I summarizes these RAG variants.

C. Grounding & Citation

Early attempts at grounding relied on **self-citation**, where the language model is directly prompted to append inline citations (e.g., [Doc1], [Doc2]) while generating answers. This approach is simple and end-to-end, but often unreliable: the model may produce plausible-looking citations without truly using the underlying documents, resulting in *post-rationalization*.

A central challenge, as emphasized by **Correctness is not Faithfulness** [11], is that citation correctness does not imply faithfulness. A model may generate the correct answer from memory and only afterwards attach a citation, undermining the trustworthiness of attribution. Correctness is thus a necessary but not sufficient condition for faithful grounding.

Building on this concern, **FActScore** [12] introduced a fine-grained evaluation framework. It decomposes model outputs into *atomic facts* and verifies each fact against retrieved documents, thereby quantifying factual precision at a granular level. Although limited to evaluation, FActScore set the standard for measuring grounding fidelity.

MIRAGE (**Model Internals-based Answer Attribution**) improves citation faithfulness by inspecting the model's generation process rather than relying on self-citation [13]. As shown in Figure 9, it proceeds in two steps: (i) **CTI** compares token probabilities with and without retrieved context to detect context-sensitive outputs; (ii) **CCI** uses gradient-based saliency to link these tokens back to specific evidence in the retrieved documents.

While MIRAGE enhances attribution fidelity, it may *underestimate faithfulness* when both model memory and retrieved documents can yield the same answer. In such mixed cases, attribution boundaries remain blurred, making MIRAGE more reliable than self-citation but not a perfect solution against post-rationalization.

Finally, system-level architectures such as **RAGentA** (**Multi-Agent Retrieval-Augmented Generation for Attributed QA**) demonstrate how multi-agent coordination can substantially enhance grounding and citation [14]. Instead of relying on a single LLM pipeline, RAGentA decomposes the process into four specialized agents, each with a distinct role:

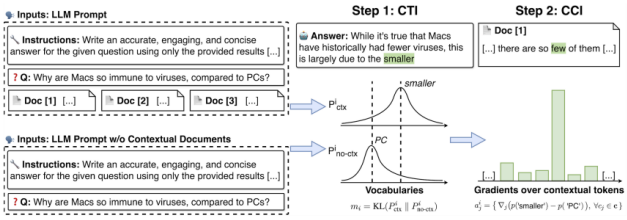


Fig. 9: MIRAGE attribution pipeline [13]. CTI identifies context-sensitive tokens, CCI traces them to supporting evidence.

- **Agent Predictor:** Generates document-specific answers by forming triplets (Q, D, A) for each retrieved passage.
- **Agent Judge:** Scores and filters these candidate answers by assessing relevance and factual support, discarding spurious citations.
- **Agent Final-Predictor:** Produces the final answer while enforcing inline citations, ensuring that every factual claim is explicitly grounded in the retrieved documents.
- **Agent Reviser:** Checks the completeness of the answer, reformulates missing sub-questions if necessary, and integrates additional retrieval results to guarantee coverage.

The overall pipeline is shown in Figure 10.

This modular architecture yields measurable improvements. On a synthetic QA benchmark, RAGentA improved attribution correctness by +1.1% and faithfulness by +10.7% compared to a strong RAG baseline, showing that dividing labor among agents reduces both hallucinations and unfaithful citations. At the same time, its complexity and computational cost are higher, since multiple LLM calls are required for each query. Nevertheless, RAGentA illustrates how **agentic orchestration** at the system level can transform grounding from a passive by-product into an actively managed process.

D. Agentic Orchestration

In DeepResearch, **Agentic Orchestration** refers to the ability of large language models to act not merely as passive responders but as active *research assistants* that can decompose tasks, call external tools, and coordinate multi-step workflows. This orchestration capability turns retrieval-augmented generation into a dynamic process of planning, execution, and verification, rather than a static query–answer pipeline.

Early work such as **ReAct** [5] demonstrated a *Thought–Action–Observation* loop, where the model alternates between reasoning in natural language and issuing explicit actions (e.g., web search, database queries). **Self-RAG** [8] internalized this paradigm by introducing reflection tokens (`<Retrieve>`, `<ISREL>`, `<ISSUP>`), enabling end-to-end orchestration within the LM's output stream. System-level approaches such as **RAGentA** [14] go further by decomposing roles across multiple agents (Predictor, Judge, Final-Predictor, Reviser), ensuring that each factual claim is both grounded and explicitly cited. Together, these paradigms illustrate how agentic control can be implemented at different levels: explicit reasoning traces, token-level control, and multi-agent pipelines. These paradigms are compared in Table II.

In practice, agentic orchestration enables DeepResearch systems to (i) dynamically decide when and how to retrieve

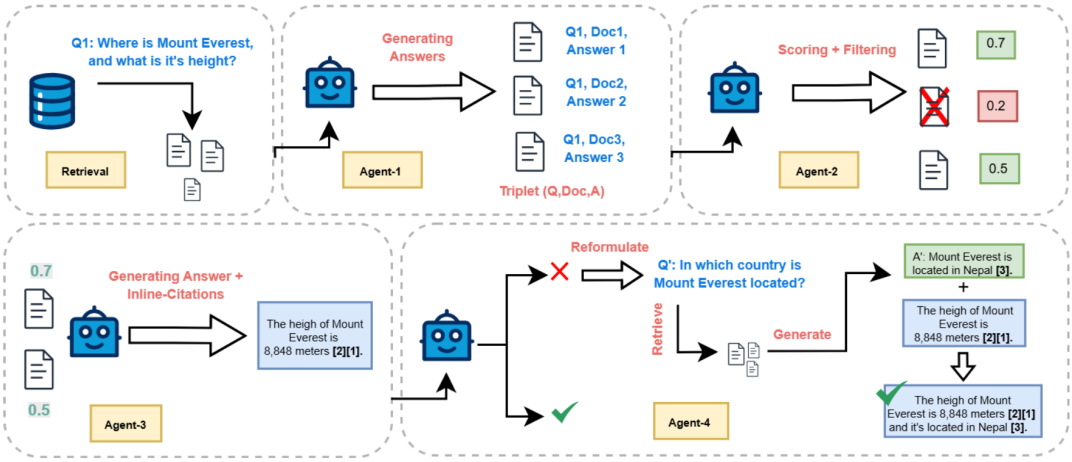


Fig. 10: The RAGentA multi-agent pipeline [14]. Four agents collaborate to predict document-level answers, judge and filter evidence, enforce inline citations, and revise outputs for completeness, thereby improving citation faithfulness.

TABLE II: Representative implementations of Agentic Orchestration in DeepResearch.

Paradigm	Strengths	Limitations
ReAct	Transparent reasoning; flexible tool use	Needs external controller
Self-RAG	End-to-end, internalized control	May misjudge faithfulness
RAGentA	High citation faithfulness; modular	Complex, slower inference

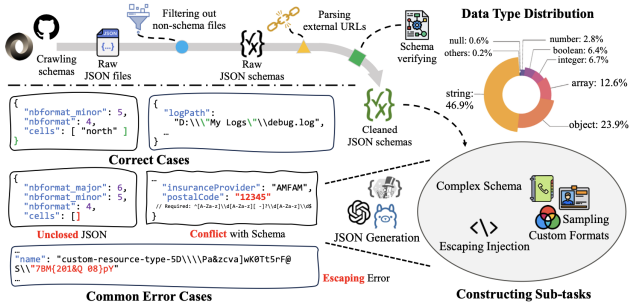


Fig. 11: Data curation and schema validation pipeline for training models to internalize JSON rules.

information, (ii) decompose broad research queries into sub-questions, (iii) verify and revise answers through self-critique or dedicated reviser agents, and (iv) coordinate heterogeneous tools such as search engines, APIs, or code interpreters.

E. Structured Synthesis

In DeepResearch, **Grammar-Constrained Generation** ensures that large language models (LLMs) produce syntactically valid and schema-compliant outputs. Unlike free-form text generation, which may yield malformed or hallucinated structures, grammar-constrained approaches restrict decoding so that only tokens permitted by a given formal grammar can be produced. This transforms LLMs from stochastic text generators into controllable engines for structured data.

The earliest paradigm, **Grammar-Constrained Decoding (GCD)** [15], demonstrated that many NLP tasks can be formalized as *Context-Free Grammars (CFGs)* and enforced at decoding time through incremental parsing. Building upon this, **GRAMMAR-LLM** [16] introduced the *LL(prefix)* grammar, a superset of LL(1) more compatible with subword tokenization. Crucially, any LL(prefix) grammar can be *compiled into an equivalent LL(1) grammar*, allowing deterministic parsing via a Pushdown Automaton (PDA) in linear time,

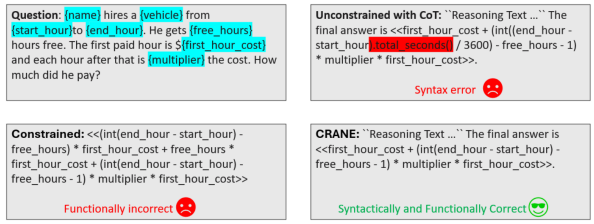


Fig. 12: CRANE: separating unconstrained reasoning from constrained final decoding ensures both validity and reasoning capacity.

TABLE III: Representative paradigms of Grammar-Constrained Generation.

Paradigm	Strengths	Limitations
GCD	Guarantees validity; broad task coverage	May restrict reasoning capacity
GRAMMAR-LLM	LL(prefix) → PDA; tokenization-friendly	Limited to CFG-expressible tasks
SRL + ToS	Internalizes schema rules; boosts JSON validity	Costly RL training; schema-specific
CRANE	Balances reasoning and validity	Requires switching mechanism
XGrammar	Flexible (CFG, regex, schema); efficient runtime	Engineering focus, not reasoning

thereby retaining efficiency while increasing flexibility. At the training level, **Schema Reinforcement Learning (SRL)** [17] combined reinforcement learning with fine-grained schema validation, allowing models to internalize structural rules. The introduction of *Thoughts of Structure (ToS)* encouraged explicit structural reasoning before output, improving JSON validity under complex schemas (see Figure 11).

To reconcile the trade-off between strict validity and reasoning flexibility, **CRANE** [18] proposed a hybrid approach: unconstrained reasoning for intermediate steps, followed by constrained decoding for the final answer (Figure 12). Finally, system-level engines such as **XGrammar** [19] provide a unified runtime capable of enforcing CFGs, regex, and JSON schemas efficiently across diverse applications. Table III summarizes these paradigms.

A key insight of GCD is that a wide variety of structured NLP tasks can be expressed as CFGs. Table IV illustrates 14 representative tasks and their corresponding grammar formulations.

In practice, grammar-constrained generation provides strong guarantees of validity but also raises important trade-offs. Empirical studies (e.g., [20]) show that schema-guided decoding significantly improves table fidelity in numerical extraction tasks (e.g., Rotowire) but may degrade performance in reasoning-heavy or aggregation tasks (e.g., Livesum),

TABLE IV: Example CFG-style grammars for 14 structured NLP tasks (adapted from [15]).

Task	Grammar (simplified form)
Closed IE	$S \to \epsilon \mid [s]\ \alpha\ [r]\ \beta\ [o]\ \alpha\ S$
Entity Disamb. (ED)	$S \to \ell\ m\ [\alpha]\ r$
Constituency Parsing	$S \to B_{0,0};\ B_{i,j} \to [\alpha\ (B_{i,j+1} \mid C_{i,j+1})]$
Coreference	$S_i \to x_i\ [(x_1 \mid \cdots \mid x_n \mid \bot)]\ S_{i+1}$
POS Tagging	$S_i \to x_i\ [(\text{NOUN} \mid \text{VERB} \mid \text{ADJ} \cdots)]\ S_{i+1}$
Dependency Parsing	$S_i \to x_i\ [(\text{ROOT} \mid \text{NSUBJ} \mid \cdots)(x_1 \mid \cdots \mid x_n \mid \bot)]\ S_{i+1}$
Word Sense Disamb.	$S_i \to x_i\ [\alpha_i]\ S_{i+1}$
Chunking	$S \to B_0;\ B_i \to [C_i;\ B_n \to \epsilon;\ C_i \to x_i(C_{i+1} \mid \alpha]B_{i+1})$
Semantic Role Labeling	Same as Chunking, but α = {TARGET, ARG0, ARG1, ...}
Entity Linking	Same as Chunking, but α = KB entities $\cup\{\bot\}$
CCG Parsing	CFG with α = CCG categories, e.g. $(S\backslash NP)/NP$
Question Answering	$S \to [q][A];\ A \to (\epsilon \mid \alpha\ A)$
Extractive Summarization	$S \to (\epsilon \mid [\alpha]\ S),\ \alpha$ = input sentences
Semantic Parsing (λ-calc.)	CFG generating λ-calculus expressions

echoing CRANE's motivation to decouple reasoning from constrained output. Thus, the effectiveness of structural constraints depends critically on the balance between validity enforcement and reasoning flexibility.

III. System Architectures

A. Four-layer View of DeepResearch Systems

We organize **DeepResearch systems** into four engineering layers, each with distinct responsibilities and corresponding capability mappings (Table V):

- **L1: Foundation models & reasoning engines.** This layer provides the computational backbone of DeepResearch, encompassing long-context large language models and reasoning-optimized variants. Its primary role is context handling, memory management, and base reasoning. Core capabilities such as chain-of-thought (CoT), tree-of-thought (ToT), tool-use APIs, and memory augmentation are grounded here, enabling robust multi-step reasoning.
- **L2: Task planning & execution control.** Built upon the foundation, this layer introduces explicit planning, decomposition of complex tasks, scheduling, monitoring, and self-critique mechanisms. It equips systems with execution control, error handling, and guardrails. Capabilities include ReAct-style reasoning–action loops, retry/backoff strategies, success criteria evaluation, and agentic orchestration of workflows.
- **L3: Tool utilization & environment interaction.** This layer connects the reasoning agent to external environments and data sources. Its responsibilities cover retrieval and interaction with APIs, web browsers, code execution environments, and document parsers. It supports advanced deep retrieval (e.g., RAG++), multi-hop information access, table/PDF parsing, and seamless integration of external tools into the reasoning loop.
- **L4: Knowledge synthesis & structured output.** The top layer integrates evidence, ensures consistency, and produces structured deliverables. Responsibilities include deduplication, verification, fact-checking, and citation. Capabilities such as grounding, structured generation, and report/table/graph construction reside here, ensuring that outputs are verifiable, coherent, and ready for downstream use.

B. Architectural Paradigms

DeepResearch systems can be instantiated under four major architectural paradigms: *monolithic*, *pipeline-based*, *multi-agent*, and *hybrid*. These paradigms have been identified in recent surveys [21], [22] and observed in both academic prototypes and industrial deployments.

a) Monolithic DeepResearch.

In the monolithic design, all functions are tightly integrated into a single centralized reasoning engine. The foundation model directly handles query processing, retrieval, analysis, and synthesis, supported by a unified memory system. This design ensures strong coherence and determinism but suffers from poor extensibility and limited parallelism. Representative systems include `OpenAI/DeepResearch` and `grapeot/deep_research_agent` [23], [24]. (Figure 13a)

b) Pipeline-based DeepResearch.

The pipeline paradigm decomposes research into a linear sequence of modular stages, such as query processing, retrieval, content analysis, and output. Each stage contains specialized components (e.g., intent classifier, fact verifier), connected by standardized interfaces. The benefits lie in modularity and component reusability, though the rigidity of sequential processing limits adaptability. Examples include workflow-automation systems like `n8n` and `dzhng/deep-research` [25], [26]. (Figure 13b)

c) Multi-agent DeepResearch.

In this architecture, multiple autonomous agents (e.g., searcher, analyst, critic, writer) collaborate via explicit coordination mechanisms and a standardized message bus. Each agent is specialized and can access shared knowledge repositories. This design supports parallelism, specialization, and fault isolation, but introduces higher coordination cost and potential coherence risks. Representative systems include `smolagents/open_deep_research` and `TARS` [27], [28]. (Figure 13c)

d) Hybrid DeepResearch.

Hybrid systems combine centralized reasoning cores with pipeline and/or multi-agent subsystems. An adaptive orchestration layer directs tasks to the appropriate sub-architecture, balancing determinism (from pipelines), flexibility (from multi-agent), and reasoning consistency (from the monolithic core). While this paradigm maximizes adaptability, it also poses integration and engineering complexity. Representative systems include `Perplexity/DeepResearch` and `Camel-AI/OWL` [29], [30]. (Figure 13d)

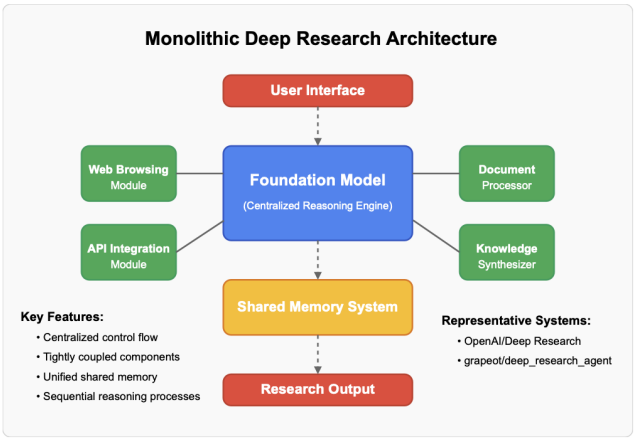
Table VI compares these architectural paradigms.

IV. Representative Systems

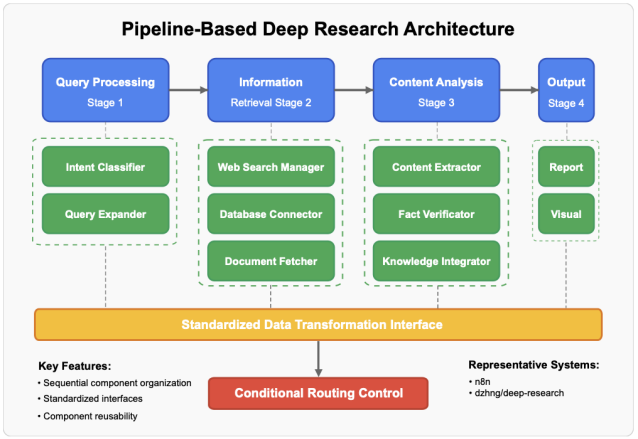
To further examine how current DeepResearch systems differ in practical availability, pipeline completeness, and

TABLE V: Four-layer view of DeepResearch systems: responsibilities and mapped capabilities

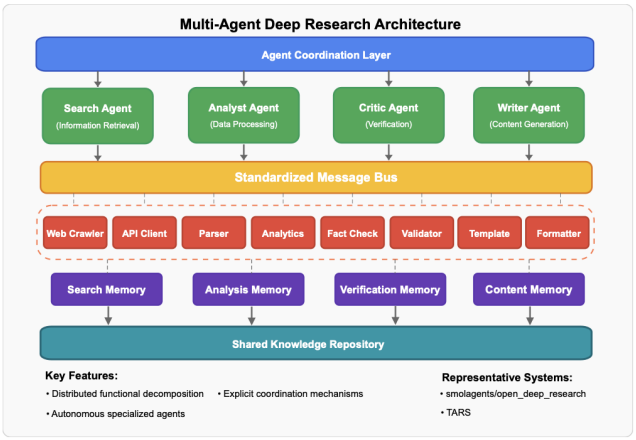
Layer	Primary Responsibilities	Mapped Capabilities (examples)
L1: Foundation models & reasoning engines	Context handling, base reasoning, memory management	CoT/ToT, tool-use APIs, long-context attention, memory augmentation; *Multi-step reasoning*
L2: Task planning & execution control	Decomposition, scheduling, monitoring, error handling, self-critique	ReAct-style loops, retry/backoff, success criteria; *Multi-step reasoning, Agentic orchestration*
L3: Tool utilization & environment interaction	Retrieval and interaction with external sources (web/API, code, parsers)	RAG++, multi-hop retrieval, table/PDF parsing, code execution; *Deep retrieval, Agentic orchestration*
L4: Knowledge synthesis & structured output	Evidence integration, verification, structured reporting	Grounding/citation, fact-check, report/table/graph generation; *Grounding & citation, Structured generation*



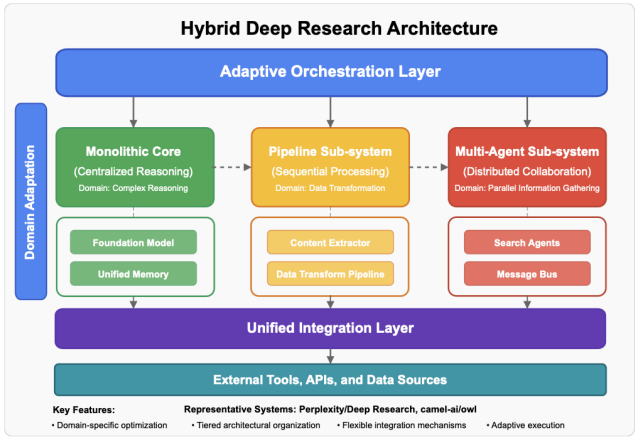
(a) Monolithic DeepResearch Architecture.



(b) Pipeline-based DeepResearch Architecture.



(c) Multi-agent DeepResearch Architecture.



(d) Hybrid DeepResearch Architecture.

Fig. 13: Four representative DeepResearch architectural paradigms.

TABLE VI: Comparison of DeepResearch architectural paradigms

Paradigm	Core Idea	Advantages	Limitations
Monolithic	Single centralized agent with unified memory and reasoning flow	Strong coherence; simple design; low orchestration cost	Limited extensibility; no parallelism; brittle for complex tasks [21]
Pipeline-based	Sequential modular stages with standardized interfaces	Modularity; reusability; clarity of workflow	Rigid; limited adaptability; hard to handle iterative reasoning [22]
Multi-agent	Autonomous specialized agents coordinated via message bus	Parallelism; specialization; fault isolation; flexibility	High coordination cost; coherence risks; complex communication [27]
Hybrid	Integration of monolithic core, pipelines, and multi-agent subsystems	Balances coherence, flexibility, and adaptability	Maximal integration complexity; high engineering cost [29]

output characteristics, Table VII summarizes representative domestic and international implementations.

V. Applications

- Academic: literature review, citation tracking, evidence graphs.
- Enterprise: market/competitive intelligence, due diligence.

TABLE VII: DeepResearch functional test for different models

System	Availability	Completeness	Output Characteristics
Kimi (Moonshot)	Publicly available	Full pipeline (planning, retrieval, reasoning, synthesis)	Generates structured long reports with decent coherence; provides references
Baidu Wenxin	Publicly available	Partial pipeline	Can generate long reports, but lacks explicit source citations
ChatGPT (OpenAI)	Available (Team/Enterprise accounts)	Full pipeline	Produces structured multi-step research reports with citations
Google Gemini	Publicly available (Search/Workspace integration)	Full pipeline	Research-style responses with references; well-grounded
Perplexity AI	Publicly available	Near full pipeline	Generates long text answers with citations; more like extended articles than structured reports
Alibaba Qwen	Available, but not full DR in practice	Incomplete	Provides reasoning + retrieval, but not integrated into a full DR workflow
Doubao (ByteDance)	Limited rollout	Incomplete	Early-stage "research mode", not yet producing full reports
DeepSeek	Public (reasoning + search)	Incomplete	Offers deep reasoning + search, but no planning/structured synthesis
Zhipu ChatGLM	Public	Incomplete	Provides enhanced reasoning and some research-like answers, but no full report generation

- Government: policy analysis, legal/administrative research.
- Personal: knowledge discovery, productivity, education.

VI. Challenges and Future Directions

- Technical: reliability, long-context efficiency, evaluation and citation fidelity.
- Engineering: orchestration at scale, latency/cost, monitoring/observability.
- Societal: privacy, copyright, safety, transparency.
- Trends: 1M+ token contexts, multi-agent collaboration, self-verification, KB/graph integration.

VII. Conclusion

DeepResearch bridges LLM reasoning, retrieval, grounding, tool use, and structured outputs, moving from conversational assistants to automated research agents. We summarized capabilities, architectures (four-layer view and paradigms), systems, and open problems to guide future research and deployment.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Jinyan Cai contributed to the conceptual framing of the study, reviewed the manuscript, and provided academic guidance on the organization of the survey. Ruochong Yao conducted the literature review, designed the technical taxonomy and architectural analysis, evaluated representative DeepResearch systems, and drafted the manuscript. Both authors reviewed and approved the final version.

References

[1] Y. Yuan, H. Zhang, Y. Cheng, Y. Jiang, L. Pan, Z. Zhou, C. Xu, Z. Liu, and R. Pan, "DeepResearch Bench: A comprehensive benchmark for deep research agents," *arXiv preprint arXiv:2412.19446*, 2024.

[2] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, 2022.

[3] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *International Conference on Learning Representations*, 2023.

[4] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, and K. Narasimhan, "ReAct: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2023.

[6] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "PAL: Program-aided language models," in *International Conference on Machine Learning*, 2023.

[7] Z. Chen, W. Wang, Y. Qin, Y. Lin, Z. Liu, and M. Sun, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2023.

[8] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in *Proceedings of the International Conference on Learning Representations*, 2024.

[9] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, Q. He, Z. Hua, B. Long, T. Zhao, N. Shah, A. Javari, Y. Xia, and J. Tang, "Retrieval-augmented generation with graphs (GraphRAG)," *arXiv preprint arXiv:2501.00309*, 2025.

[10] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, "RAPTOR: Recursive abstractive processing for tree-organized retrieval," in *International Conference on Learning Representations*, 2024.

[11] J. Wallat, M. Heuss, M. de Rijke, and A. Anand, "Correctness is not faithfulness in RAG attributions," *arXiv preprint arXiv:2412.18004*, 2024.

[12] S. Min, K. Krishna, X. Lyu, M. Lewis, W.-t. Yih, P. W. Koh, M. Iyyer, L. Zettlemoyer, and H. Hajishirzi, "FActScore: Fine-grained atomic evaluation of factual precision in long form text generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[13] J. Qi, G. Sarti, R. Fernandez, and A. Bisazza, "MIRAGE: Model internals-based answer attribution for trustworthy retrieval-augmented generation," *arXiv preprint arXiv:2406.13663*, 2024.

[14] I. Besrour, J. He, T. Schreieder, and M. Farber, "RAGentA: Multi-agent retrieval-augmented generation for attributed question answering," *arXiv preprint arXiv:2506.16988*, 2025.

[15] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured NLP tasks without finetuning," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, Association for Computational Linguistics, 2023.

[16] Y. Liu *et al.*, "GRAMMAR-LLM: Grammar-constrained natural language generation," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2025.

[17] Y. Lu, H. Li, X. Cong, Z. Zhang, Y. Wu, Y. Lin, Z. Liu, F. Liu, and M. Sun, "Learning to generate structured output with schema reinforcement learning," *arXiv preprint arXiv:2502.18878*, 2025.

[18] Z. Wang, J. Jiang, H. Zhou, W. Zheng, X. Zhang, C. Bansal, and H. Yao, "CRANE: Reasoning with constrained LLM generation," in *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[19] Y. Dong, C. F. Ruan, Y. Cai, R. Lai, Z. Xu, Y. Zhao, and T. Chen, "XGrammar: Flexible and efficient structured generation engine for large language models," in *Proceedings of the 2024 Conference on Neural Information Processing Systems*, 2024.

[20] J. Oestreich and L. Mueller, "Evaluating structured decoding for text-to-table generation: Evidence from three datasets," *arXiv preprint arXiv:2508.15910*, 2025.
[21] R. Xu and J. Peng, "A comprehensive survey of deep research: Systems, methodologies, and applications," *arXiv preprint arXiv:2506.12594*, 2025.
[22] A. Shchegrikovich, "Four architectures of deep research," *Substack*, 2024.
[23] OpenAI, "OpenAI Deep Research." [Online]. Available: https://openai.com/research/deepresearch, 2024.
[24] grapeot, "Deep Research Agent." [Online]. Available: https://github.com/grapeot/deep_research_agent, 2024.
[25] n8n.io, "n8n Workflow Automation." [Online]. Available: https://n8n.io, 2024.
[26] dzhng, "Deep Research Pipeline." [Online]. Available: https://github.com/dzhng/deep-research, 2024.
[27] Anthropic, "Building a Multi-agent Research System." [Online]. Available: https://www.anthropic.com, 2024.
[28] smolagents, "Open Deep Research." [Online]. Available: https://github.com/smol-ai/open-deep-research, 2024.
[29] Perplexity AI, "Perplexity Deep Research." [Online]. Available: https://www.perplexity.ai, 2024.
[30] Camel-AI, "OWL: Orchestrated workflows for large-scale research." [Online]. Available: https://github.com/camel-ai/owl, 2024.