DEEPSEARCHDAG: A DAG-STRUCTURED BENCHMARK FOR NODE-LEVEL EVALUATION OF DEEP SEARCH AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing benchmarks for evaluating search agents predominantly verify only the final answer while treating intermediate sub-problems as a black box, providing no structural signal for distinguishing well-supported coverage from a potential lucky hit. Moreover, they typically involve a single target entity or a homogeneous set, lacking the multi-entity heterogeneity of real-world search tasks; benchmarks that jointly stress both search depth and width remain scarce. We present DeepSearchDAG (DSD), a DAG-structured benchmark built on a real-world online business dataset, decomposing queries involving multiple heterogeneous entities into atomic sub-problems connected by dependency edges, enabling node-level evaluation of both depth and width. DSD places representative search-agent benchmarks in a common node-level view and extends it to checked DAG dependencies. We propose NodeF1 for final-answer coverage and a trajectory-aware DSD-Score for tool-augmented search agents. Its Prerequisite Edge Score (PES) measures whether recorded tool trajectories follow annotated prerequisite edges. The dataset comprises 348 instances spanning 19 business domains, 10,617 atomic claims spanning enumerations, numerical values, and factual descriptions (average 30.5 per instance), and 2,411 dependency edges (average 6.9 per instance), with mean Width 45.9, Depth 16.3, and Members 49.7. Experiments show that state-of-the-art agent frameworks powered by frontier LLMs achieve low success rates on DSD, while trajectory-aware PES complements final-answer coverage with a prerequisite-edge execution signal.

1 INTRODUCTION

The rapid advancement of large language model (LLM) agents—exemplified by systems such as OpenAI Deep Research, Gemini Deep Research, and Perplexity Pro—has created an urgent need for evaluation benchmarks that go beyond surface-level answer checking. Real-world search tasks are rarely atomic; they require multi-step reasoning, cross-source verification, and the synthesis of heterogeneous information types. However, current benchmarks impose significant limitations that prevent faithful evaluation of these complex behaviors.

1.1 LIMITATIONS OF EXISTING BENCHMARKS

Existing benchmarks for search agents—BrowseComp, WideSearch, and DeepWideSearch—primarily verify final answers while leaving intermediate sub-problems unchecked. Consequently, they provide no response-level structural signal indicating whether a covered downstream result is accompanied by its prerequisite information. FIRE-Bench (FIRE-Bench Team, 2026) partially addresses this through claim-level evaluation, but its tree structure cannot express cross-branch dependencies or support checked cross-claim dependency analysis. Figure 1(a) visualizes DSD's broader coverage across four structural dimensions: **Width** (W) sums the answer items across all enumeration-type claims, measuring factual breadth; **Depth** (D) is the longest path in a deterministic expanded claim graph that combines milestone-internal claim order, milestone dependencies, and explicit or anaphoric claim references. **Members** (M) is a composite combining total claim volume, type heterogeneity, and distinct entity count (formal definitions in Appendix N); and **CTH** (Claim

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

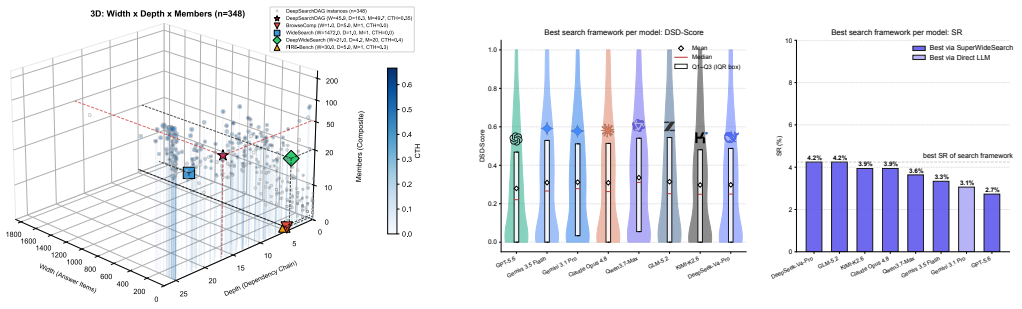

Figure 1: (a) 3D structural comparison of DSD (n=348, blue dots) with five benchmarks (large markers) across Width, Depth, and Members. DSD mean is marked with $\star$; CTH characterizes claim-type heterogeneity. (b) SuperWideSearch DSD-Score distributions on the 330-instance multi-claim subset (excluding 18 single-claim instances), and the best SR across 8 backbone LLMs. Left: violin plots with IQR boxes (1st and 3rd quartiles). Right: best SR across four search frameworks per model. SR peaks at 4.2%; SuperWideSearch is the best framework on most models.

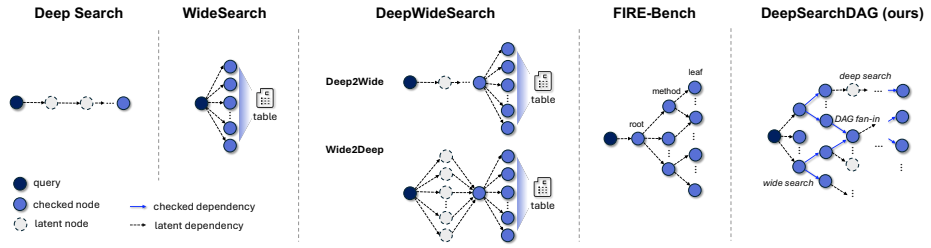

Figure 2: A unified node-level view of search agent benchmarks. Latent multi-hop structure and checked outputs are represented in a common node-level framework. Existing benchmarks primarily verify final outputs, whereas DeepSearchDAG annotates decomposed nodes and dependency edges and evaluates final-answer coverage for all systems together with tool-trajectory adherence for search agents.

Type Heterogeneity), a Simpson diversity index over the three claim types (enumeration, numerical, factual). DSD instances span a broader region of this space than any single baseline, demonstrating simultaneous scale in depth, width, and heterogeneity. On the 330-instance multi-claim subset in Figure 1(b), even the best systems achieve SR $\leq 4.2\%$, confirming that DAG-structured multi-hop reasoning remains extremely challenging for current LLM agents.

In contrast, DeepSearchDAG jointly considers breadth and depth, measures prerequisite-linked tool-trajectory adherence for search agents, localizes downstream-without-upstream execution patterns, handles heterogeneous query types, and provides node-level verification across decomposed sub-problems.

1.2 A Unified Node-Level View of Search Agent Benchmarks

Figure 2 defines *checked nodes* (solid circles) as sub-problems with ground truth and *latent nodes* (dashed circles) as unchecked sub-problems. A *checked dependency edge* (solid arrow) connects two independently annotated nodes, allowing response-level downstream-without-upstream coverage patterns to be identified. Existing benchmarks may involve latent multi-hop sub-problems but check only the final answer. DSD does not infer which hops were executed; it checks whether the response covers annotated prerequisite and downstream sub-problems.

BrowseComp and DeepSearchQA check one final node; WideSearch checks a final report; DeepWideSearch checks a final entity after latent hops; and FIRE-Bench checks claims without cross-claim dependencies. DSD supplies ground truth for decomposed sub-problems and their dependencies across heterogeneous answer types. It evaluates answer coverage and whether covered downstream results are accompanied by covered prerequisites, not the agent's actual reasoning order.

1.3 Core Contributions

DeepSearchDAG's core contributions are threefold:

(1) Real-World Business-Domain DAG Benchmark. DeepSearchDAG starts from 348 genuine commercial search-agent queries spanning 19 business domains and converts them into 10,617 typed atomic claims, reasoning-stage milestones, and human-reviewed dependency DAGs. Unlike artificially constructed multi-hop questions, this resource preserves real user information needs while adding fine-grained, reusable supervision that is absent from raw query logs. A seven-rule quality control process supports annotation reliability.

(2) Node-Level and Trajectory-Aware Evaluation. We compute node-level Recall, Precision, and NodeF1 for final-answer coverage. For tool-augmented search agents, we additionally compute the Prerequisite Edge Score (PES) from recorded tool trajectories and combine it with NodeF1 in DSD-Score.

(3) Unified Node-Level Evaluation Framework. We map representative search-agent benchmark paradigms to node topologies ranging from edge-free checked outputs to multi-hop DAGs. DSD provides NodeF1 for all evaluated systems and extends tool-augmented systems with trajectory-aware PES and DSD-Score when checked prerequisite edges are available.

2 Related Work

2.1 Benchmarks for Search Agents

QA benchmarks range from single-hop factoids (Kwiatkowski et al., 2019; Joshi et al., 2017) to compositional and multi-hop questions (Talmor & Berant, 2018; Yang et al., 2018; Trivedi et al., 2022; Press et al., 2023). Their answer formats remain relatively simple and their dependency structures are mostly linear.

Recent benchmarks stress difficult retrieval (OpenAI, 2025; Google DeepMind, 2026), breadth (ByteDance Seed, 2025), combined depth and breadth (Alibaba Group, 2025), tree-structured scientific claims (FIRE-Bench Team, 2026), or citation-grounded generation (Gao et al., 2023). FanOutQA provides human-annotated multi-hop, multi-document fan-out supervision (Zhu et al., 2024), while RAGChecker diagnoses claim-level retrieval-augmented generation quality (Ru et al., 2024). DeepSearchDAG instead derives typed claims, milestones, and checked cross-branch dependencies from real-world business queries, supporting final-answer coverage and trajectory-aware prerequisite-edge evaluation for search agents.

2.2 LLM-Based Search Agents

Search agents have progressed from retrieve-then-read pipelines (Lewis et al., 2020) to interleaved reasoning, tool use, and browsing (Yao et al., 2023; Schick et al., 2023; Nakano et al., 2021; Li et al., 2025; Khattab et al., 2025). General and web-agent benchmarks (Mialon et al., 2024; Liu et al., 2024; Deng et al., 2023; Zhou et al., 2024) typically emphasize task success or final-answer accuracy. LLM evaluators scale open-ended assessment but require reliability checks (Liu et al., 2023; Zheng et al., 2023). DSD instead localizes missing prerequisite and downstream coverage in a query-derived dependency structure.

3 DeepSearchDAG Benchmark Construction

3.1 Query Collection and DAG-Structured Query Decomposition

Figure 3 illustrates the overall data construction pipeline. **Query Source.** DeepSearchDAG queries are sourced from a large-scale real-world commercial search-agent query log, reflecting genuine user information needs that mirror professional research scenarios. Queries are specifically selected to require multi-step reasoning and multi-dimensional information integration, ensuring they cannot be answered by simple single-hop retrieval. Example queries include researching compressed air energy storage technologies and comparing AI industry policies across countries.

Statistic	Value
Total instances	348
Total atomic claims	10,617
Avg. claims per instance	30.5
Max claims per instance	229
Total milestones	2,554
Avg. milestones per instance	7.3
Total dependency edges	2,411
Avg. edges per instance	6.9
Instances with typed fan-in	190 (54.6%)
Typed fan-in edges	907 (37.6% of all edges)
Instances with excess edges over a spanning forest	191 (54.9%)
Excess edges over spanning forests	569
Business domains	19
Factual claims	5,665 (53.4%)
Numerical claims	3,220 (30.3%)
Enumeration claims	1,732 (16.3%)

Table 1: Dataset statistics for DeepSearchDAG. "Excess-edge" instances satisfy $|E_{\text{dep}}| > K - C$, where K is the number of milestones and C is the number of weakly connected components; 569 is the total excess over this spanning-forest baseline across all instances.

Multi-Stage Query Curation. Collected queries undergo a rigorous three-phase cleaning process. In Phase 1, queries are checked for decomposition correctness—ensuring that sub-questions faithfully preserve the original semantic intent and that modifier relationships (e.g., possessive structures, attributive clauses) are not incorrectly split. In Phase 2, answer uniqueness is verified—if a query's key concept has multiple equally authoritative definitions (e.g., "tier-2 cities"), explicit authoritative source constraints are added. In Phase 3, temporal and subjective filtering removes queries with strong temporal words (e.g., "today," "recently"), prediction-type questions, and subjective recommendation queries. This curation is governed by seven quality rules (detailed in Appendix D).

Fine-to-Coarse DAG Decomposition. First, an LLM decomposes each query into human-reviewed, independently verifiable claims of three types: enumeration, numerical, and factual. A script groups them into a query-dependent number of reasoning-stage milestones based on semantic stage boundaries, assigning each claim exactly once. Dependency edges are annotated between milestones: $M_i \to M_j$ indicates that M_j depends on the result of M_i as input. Milestones without dependencies can be executed in parallel, creating a multi-branch DAG structure. This decomposition provides a query-derived reference framework for trajectory-aware prerequisite-edge evaluation.

3.2 Golden Answer Annotation

Gold standard annotation follows a three-stage pipeline, adhering to the causal order of **query** $\to$ **claim** $\to$ **milestone**:

(1) Query $\to$ Claim Decomposition. An LLM decomposes each original query into atomic-level sub-problems (claims). Each claim is independently verifiable and typed as enumeration, numerical, or factual description. Claims pass staged annotation, quality control, and re-check, supplemented by release-wide automated consistency audits.

(2) Claim $\to$ Milestone Aggregation. A script aggregates claims into milestones by reasoning stage. Dependency edges between milestones ($M_i \to M_j$) are annotated, constructing a query-derived reference dependency structure—the *gold skeleton* (DAG structure). Dependency annotation follows the principle that an edge $M_i \to M_j$ is marked only when M_j cannot be completed without knowing the result of M_i. Three node types—*identify*, *lookup*, and *verify*—define different dependency patterns (detailed in Section 4.3).

(3) Gold Standard Population. Annotation experts search for authoritative sources, fill in gold answers (`gold_answer`) for each claim, and record internal source provenance. URL records are withheld from release as documented in the appendix. The annotation process follows a three-stage human pipeline (annotation, quality control, and re-check) governed by seven quality rules (see Appendix D and Appendix E for concrete examples). These controls improve verifiability, answer specificity, temporal consistency, and release-wide schema consistency.

3.3 Dataset Statistics and Domains

Table 1 summarizes the key statistics of DeepSearchDAG. The dataset averages 30.5 atomic claims and 6.9 dependency edges per instance. Typed fan-in appears in 190 instances and accounts for 907 annotated edges, making multi-source convergence explicit. Beyond fan-in, 191 instances (54.9%) contain at least one dependency edge beyond a weak spanning forest, accounting for 569 excess edges in total and directly quantifying non-tree structure.

The dataset spans 19 business domains (Table 14 in the appendix), with Enterprise & Industry being the largest segment.

3.4 Answer Types

DeepSearchDAG reports NodeF1 for all systems. For tool-augmented search agents with recorded trajectories, a *DAG Answer* with at least one checked edge additionally receives DSD-Score, which combines NodeF1 and trajectory-aware PES with weights w_1 and w_2 ($w_1 + w_2 = 1$), as shown in Table 2. Direct LLM baselines do not receive PES or DSD-Score because they do not execute search tools. We set $w_1 = 0.7$ and $w_2 = 0.3$ based on pilot experiments, keeping final-answer correctness primary while adding a process-sensitive signal for search agents.

Query Type	**DSD-Score**	**Claims**	**Edges**
Answer-only evaluation	NodeF1	≥ 1	any
Tool-augmented DAG evaluation	$0.7{\cdot}\text{NodeF1} + 0.3{\cdot}\text{PES}$	≥ 1	≥ 1

Table 2: Evaluation regimes in DSD. NodeF1 is reported for all systems. Tool-augmented systems with recorded trajectories and checked prerequisite edges additionally receive DSD-Score $= 0.7{\cdot}\text{NodeF1} + 0.3{\cdot}\text{PES}$. Direct LLM baselines are evaluated with NodeF1 only.

Within the DAG structure, individual claims are categorized into three types, each with a distinct evaluation method as summarized in Table 3:

Type	**Answer Form**	**Evaluation**	**Example**
Enumeration	Entity list	Set-based F_1^{set}	Snapdragon X series processors supported by a given laptop
Numerical	Value with unit	Error-bound	Huawei's total revenue in 2024 (billion CNY) $\rightarrow$ 7042B
Factual	Noun / phrase	LLM verification	Founder and chairman of BYD $\rightarrow$ Wang Chuanfu

Table 3: Claim types and their evaluation methods in DSD-Score.

4 DSD-Score Evaluation Framework

4.1 Overview: Two-Step Pipeline

DSD-Score evaluates final answers and recorded tool trajectories for tool-augmented search agents through the pipeline illustrated in Figure 4. Direct LLM baselines use the answer-coverage portion only:

Step 1: Extraction. For each sub-question (claim), the agent's free-form response is searched for relevant content using an LLM-based extraction prompt. If relevant content is found, the extractor preserves original wording and key facts without judging correctness, and the extracted content proceeds to Step 2. If no relevant content exists, the claim does not proceed to coverage checking. Importantly, extraction is *blind*: gold answers are never exposed to the extraction LLM, preventing answer leakage. When a claim's sub-question contains anaphoric references to preceding claims (e.g., "the above enterprises," "C1"), a two-phase process resolves the dependency: non-anaphora claims are batch-extracted first, then anaphora claims are processed sequentially with previously *extracted* answers (not gold answers) injected as context, enabling the LLM to resolve referents before searching the agent's response (details in Appendix G).

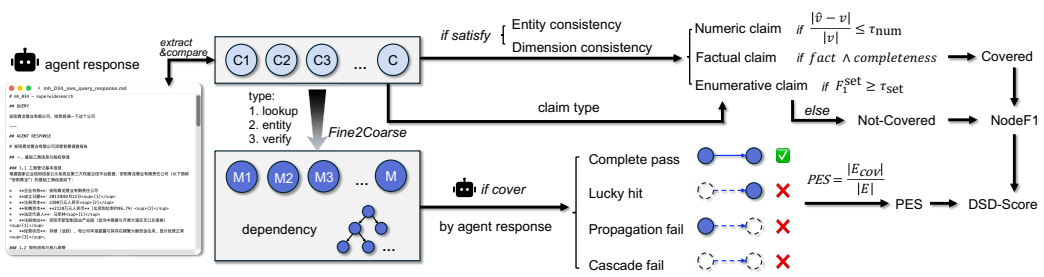

Figure 4: DSD evaluation pipeline. Final responses are extracted and checked against gold answers using type-aware coverage logic to compute NodeF1 for all systems. For tool-augmented agents, recorded search and webpage-visit events are aligned with milestone descriptions and dependency edges to compute trajectory-aware PES. DSD-Score = $0.7 \cdot$NodeF1 $+ 0.3 \cdot$PES for this tool-augmented setting.

Step 2: Coverage Check. The extracted content is checked against the gold answer through a two-stage, type-aware process: first verifying entity and dimension consistency, then branching by claim type—enumeration claims use set-based F1, numerical claims use error-bound matching, and factual claims undergo LLM-based fact consistency and information completeness checks. Claims that fail any check are classified as *not-covered*—no separate contradiction classification is made, following classic binary evaluation.

Node-level Recall, Precision, and NodeF1 are computed from the binary classification for all systems. For tool-augmented systems with checked dependency edges, the recorded trajectory is separately evaluated to compute PES.

4.1.1 Node-Level Metrics

Let an instance contain N claims. After evaluation, each claim is classified as *covered* or *not-covered*. Let Covered denote the number of covered claims, and Attempted denote the number of claims for which relevant content was found in the agent's response (i.e., claims that entered coverage checking). We define:

$$\text{Recall} = \frac{\text{Covered}}{N} \tag{1}$$

$$\text{Precision} = \frac{\text{Covered}}{\text{Attempted}} \tag{2}$$

$$\text{NodeF1} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{3}$$

When Attempted $= 0$, we define Precision and NodeF1 as zero. This classic precision formulation (Covered/Attempted) measures the accuracy of the agent's attempted answers: an agent that answers fewer claims but gets them all correct achieves higher Precision than one that answers many claims with errors. Because the metric is claim-centric, unrelated extra content that cannot be aligned to any gold claim is outside its denominator and is not separately penalized.

4.1.2 Instance-Level Orthogonal Metrics

Beyond NodeF1, we define one instance-level metric:

- **Complete Success Rate (SR)**: All claims covered *and* all sequential edges satisfied; for fan-in edges (multiple edges pointing to the same verify node), at least one incoming edge must be satisfied.

SR jointly requires complete answer coverage and the stated sequential/fan-in edge conditions: an agent can achieve high NodeF1 but SR = 0 when the required prerequisite-linked coverage conditions are not met.

4.1.3 TOTAL DSD-SCORE

As defined in Section 3.4, NodeF1 is the answer-coverage metric for all systems. For tool-augmented search agents, DSD-Score uses the composite $0.7 \cdot \text{NodeF1} + 0.3 \cdot \text{PES}$ when checked prerequisite edges are present, where PES is computed from the recorded tool trajectory as detailed in Section 4.3.

4.2 COVERAGE CHECK

The coverage check (Step 2 in Figure 4) employs a two-stage, type-aware design. In the first stage, each extracted claim is checked for *entity consistency*—whether the agent's answer refers to the same entity as the gold standard—and *dimension consistency*—whether it addresses the same attribute or dimension. Claims failing either check are immediately classified as *not-covered*. When both checks pass, the evaluation branches according to the claim type specified in the gold standard:

Numerical Claims (Error-Bound Matching). Numerical values are compared using domain-specific tolerance bounds: relative error $\leq \tau_{\text{num}}$ for general quantities, exact match for rankings and years, and absolute difference ≤ 5 percentage points for percentages.

Enumeration Claims (Set-Based F1). For set-answer type nodes where multiple entities share the same subject and dimension, we adopt the set-based F1 metric (F_1^{set}) from DeepSearchQA (Google DeepMind, 2026) as the coverage criterion. The gold set G is stored as a native JSON array and the extracted answer is parsed into a set E; their elements are same-type entities (e.g., $\{P_1, P_2\}$ for a list of supported processors). Set-level precision $P^{\text{set}} = |E \cap G|/|E|$, recall $R^{\text{set}} = |E \cap G|/|G|$, and $F_1^{\text{set}} = 2P^{\text{set}}R^{\text{set}}/(P^{\text{set}} + R^{\text{set}})$ are computed. When $F_1^{\text{set}} \geq \tau_{\text{set}}$ (default $\tau_{\text{set}} = 0.60$), the claim is judged as *covered*; otherwise it is *not-covered*.

Factual Claims (Two-Stage LLM Verification). Factual claims undergo two further verification stages: (1) *fact consistency*—whether the core fact is consistent with the gold standard, with numerical tolerance for embedded quantities; (2) *information completeness*—whether the agent's answer covers all key information points. Only when both stages are passed is the claim classified as *covered*.

LLM-Based Evaluation Configuration. Both extraction and factual claim verification use LLM judges at temperature 0 to reduce sampling variability. The binary covered/not-covered design avoids ordinal ambiguity, and the two-stage entity/dimension gate before type-specific checking further constrains the judge's decisions to well-scoped comparisons. The primary benchmark Judge is Qwen3.7-Max; the repeatability and sensitivity study additionally uses KIMI-K2.6 and DeepSeek-V4-Pro. Qwen3.7-Max is also one of the evaluated backbones, so independent-Judge sensitivity is reported explicitly rather than treating judge agreement as human validity. A three-judge, three-run evaluation on a fixed 100-instance subset yields within-judge Fleiss' κ of 0.892–0.919; cross-judge sensitivity (claim-level $\kappa = 0.844$ and a DSD-Score range of 0.066) and the full protocol are reported in Appendix C.

4.3 PREREQUISITE EDGE SCORE (PES)

4.3.1 GOLD SKELETON CONSTRUCTION

We employ a fine-to-coarse strategy in which claims belonging to the same reasoning stage are aggregated into milestones using a script, constructing the *gold skeleton*. The gold skeleton is a DAG where nodes are milestones and edges are dependencies. The concept hierarchy is: *query* (original query) $\rightarrow$ *claim* (atomic sub-problem, decomposed by LLM) $\rightarrow$ *milestone* (reasoning stage, aggregated from claims) $\rightarrow$ *gold skeleton* (DAG, milestones + dependency edges). The aggregation forms a strict partition: each claim belongs to exactly one milestone, and every claim is assigned to some milestone, ensuring no overlap and no omission.

4.3.2 PES DEFINITION

PES is defined for tool-augmented search agents with recorded trajectories. Let the gold skeleton's dependency edge set be E, where $M_i \rightarrow M_j$ means that M_j depends on the result of M_i. For each milestone, the trajectory evaluator identifies observable tool evidence associated with the milestone

description, including search queries, webpage visits, returned documents, and state transitions. This evidence is independent of the final-answer claim decisions used by NodeF1.

Let $t_i \in \{0, 1\}$ indicate whether the trajectory contains valid tool evidence for milestone M_i, and let $o_{ij} \in \{0, 1\}$ indicate whether a downstream action for M_j occurs after an upstream result for M_i and uses that result in the downstream query, state, or tool parameters. The edge is trajectory-covered only when both endpoints are supported and the dependency is observable:

$$\text{score}(M_i \to M_j) = t_i \cdot t_j \cdot o_{ij}. \tag{4}$$

A trajectory-level *dependency break* occurs when the agent performs a downstream action without observable use of the upstream result, yielding $\text{score}(e) = 0$ even if the final answer later contains downstream content. This separates process adherence from answer correctness: NodeF1 evaluates the final report, whereas PES evaluates the recorded tool trajectory.

PES is the fraction of trajectory-covered prerequisite edges:

$$\text{PES} = \frac{\sum_{(i,j)\in E} t_i t_j o_{ij}}{|E|}. \tag{5}$$

The trajectory evidence is evaluated against the query-derived gold skeleton rather than a hidden chain-of-thought trace. The gold annotation specifies the dependency relation and the milestone semantics, while the agent trajectory supplies the observable tool events. Systems with similar NodeF1 can therefore differ in PES when they obtain similar final-answer coverage through different search behaviors. DSD-Score combines final-answer correctness with this process-sensitive prerequisite-edge signal.

PES and DSD-Score are not reported for Direct LLM baselines, which do not execute search tools. For tool-augmented agents, instances without dependency edges use NodeF1 only, while instances with checked edges use $0.7 \cdot \text{NodeF1} + 0.3 \cdot \text{PES}$. All edge types—including fan-out and fan-in—are treated uniformly with the same binary trajectory scoring.

4.3.3 Milestone Types

Dependency edges are annotated only for necessary information dependencies: i.e., "the query for M_j cannot be constructed without knowing the result of M_i." Three milestone types define different dependency patterns:

- **Identify** nodes: Determine what an unknown entity is. No prerequisites; typically the starting point.
- **Lookup** nodes: Find attributes of known entities. Depends on identify (must first know the entity).
- **Verify** nodes: Cross-validation. Exhibits *fan-in*: the same objective question is supported by two or more annotated evidence branches (e.g., official filings and third-party sources). Each incoming edge is counted separately in the PES denominator, so trajectory adherence to corroboration-related sub-problems contributes to prerequisite-linked coverage.

All edge types are scored uniformly using the binary trajectory formula above. DSD-Score combines final-answer coverage (NodeF1) and trajectory-aware prerequisite-edge coverage (PES) with weights w_1 and w_2, respectively.

5 Experiments

5.1 Experimental Setup

We evaluate two categories of systems on DeepSearchDAG: (1) *LLMs* (without tool calls): eight frontier models from seven vendors, including GPT-5.6, Gemini 3.5 Flash, Gemini 3.1 Pro, Claude Opus 4.8, Qwen3.7-Max, GLM-5.2, KIMI-K2.6, and DeepSeek-V4-Pro; and (2) *Agent Frameworks*: SuperWideSearch, WebThinker (Li et al., 2025), and Open Deep Research (Khattab et al.,

2025). Each of the three frameworks is evaluated with all eight backbone models, giving 32 model–setting configurations including the direct-LLM baseline. All agent systems utilize identical tools and API endpoints (details in Appendix M).

5.2 Evaluated Agents

Appendix A lists the evaluated models and systems (Table 5). We evaluate three agent frameworks: **SuperWideSearch**, a general-purpose internal search-agent framework used in an enterprise task-assistant product. It combines multi-hop reasoning with broad information collection through a plan-search-verify loop: a planner subagent decomposes the query into sub-tasks, parallel worker subagents execute web search and page visits, and a verifier subagent consolidates cross-source evidence; **WebThinker** (Li et al., 2025), a ReAct-style agent with configurable search and visit tools; and **Open Deep Research** (Khattab et al., 2025), an open-source multi-agent research system built on LangChain. All three frameworks are configured with identical search and webpage visit tools for fair comparison. The 348 evaluation queries and their gold annotations were not used to train, tune, or select SuperWideSearch.

5.3 Main Results

Figure 1(b) reports DSD-Score distributions and best success rate on the 330-instance multi-claim subset; Table 6 gives all 32 settings. Table 4 summarizes the eight-backbone tool-framework results.

Setting	NodeF1	PES	Potential lucky-hit	Down-only edges (%)
SuperWideSearch	**0.354**	**0.221**	0.449	12.0
WebThinker	0.222	0.106	0.474	6.8
Open Deep Research	0.194	0.100	0.386	8.1

Table 4: Eight-backbone diagnostics for tool-augmented search frameworks. NodeF1 is final-answer coverage; PES is trajectory-based prerequisite-edge coverage; potential lucky-hit is $\Pr(\mathrm{PES} < 0.5 \mid \mathrm{NodeF1} \geq 0.5)$; Down-only is the percentage of dependency edges whose downstream action lacks observable use of the upstream result. Direct LLM baselines do not have tool trajectories and are therefore not included.

SuperWideSearch attains the highest eight-backbone average NodeF1 and PES. PES uses recorded tool behavior, while the last two columns summarize process breaks through the potential lucky-hit and downstream-without-upstream diagnostics.

Direct LLM is an answer-only baseline; its occasional advantage suggests framework-level information loss.

6 Limitations

DeepSearchDAG has three primary limitations. First, the DSD-Score weights ($w_1 = 0.7$, $w_2 = 0.3$) and enumeration threshold ($\tau_{\text{set}} = 0.60$) are empirical; calibration utilities are provided. Second, the gold skeleton is query-derived rather than a hidden reasoning trace, so PES evaluates only observable tool-call adherence. Third, temporal drift requires versioned re-annotation.

7 Conclusion

We present DeepSearchDAG (DSD), a real-world business benchmark that decomposes multi-entity search queries into typed atomic claims and checked dependency DAGs, supporting node-level evaluation of search breadth and depth. Its common node-level representation covers representative search-agent paradigms, while NodeF1 evaluates final-answer coverage for all systems and trajectory-aware DSD-Score combines NodeF1 with prerequisite-edge adherence for tool-augmented search agents. Experiments reveal low success rates for current agents and show that PES localizes breaks in prerequisite-linked tool execution.

AI USE STATEMENT

Generative AI tools were used to assist with (i) the LLM-based query decomposition, milestone and skeleton construction support, answer extraction, and coverage judgments described in the paper; (ii) implementation and debugging of data-processing, scoring, and experimental scripts; (iii) consistency checking and reformatting of benchmark artifacts; and (iv) language editing, translation, organization, and LaTeX formatting of the manuscript. Generative AI tools also provided feedback on experimental design and preliminary interpretation of results. Gold annotations and AI-assisted benchmark artifacts were subject to human annotation, quality control, and re-check procedures as described in the paper. The authors reviewed all AI-assisted work and take responsibility for the final content of this submission, including its text, claims, code, data artifacts, analyses, and conclusions.

REFERENCES

Alibaba Group. Deepwidesearch: Unifying deep reasoning and broad information seeking, October 2025. Preprint.

ByteDance Seed. Widesearch: Benchmarking web search agents for systematic information gathering, August 2025. Preprint.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

FIRE-Bench Team. Fire-bench: A benchmark for research agent capability evaluation. In *Proceedings of the International Conference on Machine Learning (ICML)*, February 2026.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6465–6488, 2023.

Google DeepMind. Deepsearchqa: Evaluating deep search capabilities of language models, January 2026. Technical Report.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Omar Khattab, Harrison Chase, and LangChain. Open deep research: An open-source multi-agent research system, 2025. URL `https://github.com/langchain-ai/open_deep_research`. Open-source software.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Kelcey, Jacob Devlin, et al. Natural questions: A benchmark for question answering research. In *Transactions of the Association for Computational Linguistics (TACL)*, volume 7, pp. 452–466, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Xiaoxi Li et al. WebThinker: Empowering large reasoning models with deep research capability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *Proceedings of the International Conference on Learning Representations*, 2024.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, 2023.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *Proceedings of the International Conference on Learning Representations*, 2024.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Joseph, Ben Saunders, et al. WebGPT: Browser-assisted question-answering with human feedback, 2021. arXiv preprint arXiv:2112.09332.

OpenAI. Introducing browsecomp: A benchmark for browsing agents. In *OpenAI Research Blog*, April 2025. URL `https://openai.com/index/browsecomp/`.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Dongyu Ru, Liang Qiu, Xue Hu, Tian Zhang, Peng Shi, Shuaichen Chang, Jun Cheng, Cheng Wang, Shuai Sun, Hongyu Li, Zhaochun Zhang, Bin Wang, Jing Jiang, Tian He, Zhen Wang, Pengfei Liu, Yichi Zhang, and Zhaoqing Zhang. RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 641–651, 2018.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. In *Transactions of the Association for Computational Linguistics (TACL)*, volume 10, pp. 539–554, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *Proceedings of the International Conference on Learning Representations*, 2024.

Andrew Zhu, Andrew Hwang, Liam Dugan, and Chris Callison-Burch. FanOutQA: A multi-hop, multi-document question answering benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, volume 2, pp. 18–37, 2024.

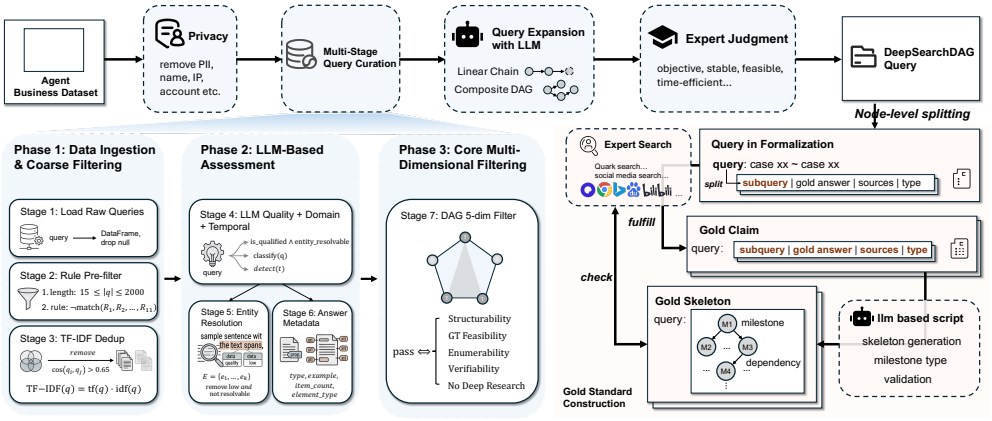

Figure 3: Data construction pipeline for DeepSearchDAG. The pipeline consists of three phases: (1) *Query Collection*—real-world complex search needs are collected and filtered for multi-step reasoning requirements; (2) *Multi-Stage Query Curation*—queries undergo three-phase cleaning including decomposition correctness, answer uniqueness verification, and temporal/objective filtering; (3) *DAG Decomposition and Gold Annotation*—queries are decomposed into atomic claims by LLM, aggregated into milestones, and annotated by domain experts with gold answers, claim types, internally retained source provenance, and dependency edges.

A Evaluated Models and Agent Systems

We evaluate two categories of systems on DeepSearchDAG:

(1) LLMs without tool calls. Eight frontier models from seven vendors are evaluated in a zero-shot setting: GPT-5.6 (OpenAI), Gemini 3.5 Flash and Gemini 3.1 Pro (Google), Claude Opus 4.8 (Anthropic), Qwen3.7-Max (Alibaba), GLM-5.2 (Zhipu), KIMI-K2.6 (Moonshot), and DeepSeek-V4-Pro (DeepSeek). These models answer each query directly using their parametric knowledge, serving as a baseline to isolate the contribution of tool-augmented search.

(2) Agent Frameworks with tool calls. Three agent frameworks are evaluated, each paired with all eight backbone LLMs listed above. Together with the direct-LLM baseline, this yields 32 model–setting configurations:

- **SuperWideSearch**: A general-purpose internal search-agent framework used in an enterprise task-assistant product. It employs a plan-search-verify loop: a planner subagent decomposes the query into sub-tasks, parallel worker subagents execute web search and page visits, and a verifier subagent consolidates cross-source evidence before producing the final report.
- **WebThinker** (Li et al., 2025): A deep research framework that empowers large reasoning models with autonomous web search capability. It interleaves chain-of-thought reasoning with dynamic search actions, allowing the model to decide when and what to search during the reasoning process.
- **Open Deep Research** (Khattab et al., 2025): An open-source multi-agent research system built on LangChain. It orchestrates multiple specialized agents (e.g., researcher, writer, editor) that collaborate to gather, synthesize, and report information.

All three frameworks are configured with identical search and webpage visit tools to ensure fair comparison. The 348 evaluation queries and their gold annotations were not used to train, tune, or select SuperWideSearch. Table 5 summarizes the 32 evaluated configurations.

B Main Results

Table 6 presents the main results across all evaluated systems.

Multi-claim-only results. The full 348-instance set contains 18 single-claim instances (queries decomposed into exactly one atomic sub-question). Fifteen are conventional one-milestone, edge-

Vendor	Backbone model	Evaluated settings
OpenAI	GPT-5.6	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Google	Gemini 3.5 Flash	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Google	Gemini 3.1 Pro	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Anthropic	Claude Opus 4.8	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Alibaba	Qwen3.7-Max	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Zhipu	GLM-5.2	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
Moonshot	KIMI-K2.6	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research
DeepSeek	DeepSeek-V4-Pro	Direct LLM; SuperWideSearch; WebThinker; Open Deep Research

Table 5: Evaluated models and settings on DeepSearchDAG. Each of the eight backbone models is evaluated directly and with all three agent frameworks (32 configurations in total).

free items; the other three retain additional query-stage milestones and edges in their skeletons. We therefore remove all 18 as a conservative sensitivity subset rather than treating them as structurally uniform. Table 7 recomputes answer-coverage metrics for all settings and trajectory-aware PES/DSD-Score for tool-augmented settings on the resulting 330 multi-claim instances. The relative ordering of frameworks is preserved (SuperWideSearch remains best for 7 of 8 models), but SR drops sharply—e.g. SuperWideSearch with GPT-5.6 falls from 5.5% to 2.7%, and with KIMI-K2.6 from 6.3% to 3.9%—because the single-claim group succeeds at 39–56% while multi-claim instances succeed at only 2.7–4.2%. NodeF1 and Precision remain stable within ± 0.02, since they are graded on partial credit rather than all-or-nothing chains.

C LLM-Judge Stability and Sensitivity

We evaluate repeated-run stability and cross-judge sensitivity on a fixed subset of 100 DeepSearchDAG instances. Using seed 2027, we stratify the 317-instance pool with non-empty SuperWideSearch responses from all four candidate backbone response sets by dependency-edge presence and claim-count quartile, then sample 100 instances. Restricting the pool to non-empty responses avoids inflating agreement through trivially identical missing-output decisions; accordingly, this analysis characterizes judge robustness on substantive responses rather than completion behavior over the full benchmark. We hold the SuperWideSearch responses generated with one backbone, gold claims, dependency skeletons, scoring rules, and default 10,000-character response truncation fixed.

Each response is independently evaluated three times at temperature 0 by three different LLM judges, producing nine complete 100-instance jobs. One complete run is reused from an earlier execution with matching judge model, scorer, gold data, skeletons, sample IDs, and response texts; the other eight jobs are newly executed. All nine jobs complete 100/100 instances, and the aggregation audit reports no missing jobs or instance-level score files.

Within-judge repeatability. Table 8 measures stability across the three runs of each judge. Fleiss' κ is computed over claim-level covered/not-covered decisions. Coverage flip is the percentage of claims whose decision changes in at least one run; extraction agreement is the percentage for which all runs agree on whether relevant response content is present. For each score metric, we compute the standard deviation across the three runs separately for each instance and report its mean over the 100 instances. SR agreement is the percentage of instances receiving the same binary SR in all three runs.

Within-judge claim agreement is high across Qwen3.7-Max, KIMI-K2.6, and DeepSeek-V4-Pro ($\kappa = 0.892$–0.919). Coverage flips occur for 6.1–8.1% of claims, while extraction-presence agreement remains 98.1–99.6%. Mean instance-level DSD-Score standard deviations range from 0.0293

Method	NodeF1	Recall	Precision	PES	DSD-Score	SR (%)
GPT-5.6						
Direct LLM	0.289	0.281	0.305	—	—	3.5
SuperWideSearch	**0.334**	**0.319**	**0.363**	**0.200**	**0.2948**	**5.5**
WebThinker	0.287	0.281	0.305	0.176	0.2542	4.0
Open Deep Research	0.252	0.246	0.264	0.141	0.2191	3.4
Gemini 3.5 Flash						
Direct LLM	0.254	0.243	0.288	—	—	3.5
SuperWideSearch	**0.356**	**0.347**	**0.371**	**0.216**	**0.3138**	**5.2**
WebThinker	0.159	0.134	0.300	0.072	0.1329	3.8
Open Deep Research	0.230	0.214	0.288	0.120	0.1966	3.7
Gemini 3.1 Pro						
Direct LLM	0.307	0.305	0.314	—	—	**4.6**
SuperWideSearch	**0.363**	**0.356**	**0.379**	**0.215**	**0.3187**	4.3
WebThinker	0.174	0.164	0.214	0.092	0.1493	3.7
Open Deep Research	0.189	0.174	0.245	0.098	0.1621	4.3
Claude Opus 4.8						
Direct LLM	0.223	0.216	0.243	—	—	2.3
SuperWideSearch	**0.355**	**0.347**	**0.372**	**0.226**	**0.3163**	**6.0**
WebThinker	0.203	0.166	0.340	0.077	0.1657	4.0
Open Deep Research	0.181	0.169	0.223	0.089	0.1543	4.0
Qwen3.7-Max						
Direct LLM	0.284	0.281	0.290	—	—	3.4
SuperWideSearch	**0.380**	**0.375**	**0.395**	**0.251**	**0.3409**	**5.7**
WebThinker	0.232	0.205	0.335	0.109	0.1950	3.7
Open Deep Research	0.207	0.192	0.266	0.108	0.1777	4.3
GLM-5.2						
Direct LLM	0.296	0.293	0.304	—	—	4.0
SuperWideSearch	**0.356**	**0.349**	**0.373**	**0.230**	**0.3177**	**6.0**
WebThinker	0.195	0.165	0.318	0.087	0.1628	3.4
Open Deep Research	0.155	0.141	0.212	0.075	0.1304	2.9
KIMI-K2.6						
Direct LLM	0.291	0.288	0.299	—	—	4.0
SuperWideSearch	**0.345**	**0.340**	**0.359**	**0.221**	**0.3083**	**6.3**
WebThinker	0.171	0.136	0.305	0.066	0.1399	3.2
Open Deep Research	0.251	0.243	0.272	0.136	0.2169	3.2
DeepSeek-V4-Pro						
Direct LLM	0.282	0.280	0.287	—	—	3.4
SuperWideSearch	**0.343**	**0.335**	**0.365**	**0.200**	**0.3007**	**5.7**
WebThinker	0.192	0.163	0.304	0.083	0.1603	4.0
Open Deep Research	0.177	0.162	0.238	0.085	0.1498	3.2

Table 6: Main results on DeepSearchDAG (348 instances), grouped by backbone model. NodeF1, Recall, and Precision are reported for all settings. PES and DSD-Score are reported only for tool-augmented search frameworks because they require recorded tool trajectories; Direct LLM entries are marked by dashes. SR denotes the Success Rate.

Method	NodeF1	Recall	Precision	PES	DSD-Score	SR (%)
GPT-5.6						
Direct LLM	0.293	0.284	0.310	—	—	2.4
SuperWideSearch	**0.322**	**0.306**	**0.353**	**0.180**	**0.2797**	**2.7**
WebThinker	0.282	0.275	0.301	0.165	0.2465	2.1
Open Deep Research	0.250	0.244	0.263	0.134	0.2154	2.1
Gemini 3.5 Flash						
Direct LLM	0.255	0.244	0.292	—	—	2.5
SuperWideSearch	**0.354**	**0.345**	**0.370**	**0.207**	**0.3099**	**3.3**
WebThinker	0.146	0.120	0.295	0.054	0.1187	1.8
Open Deep Research	0.227	0.211	0.289	0.112	0.1926	2.4
Gemini 3.1 Pro						
Direct LLM	0.306	0.304	0.313	—	—	**3.1**
SuperWideSearch	**0.359**	**0.352**	**0.376**	**0.203**	**0.3121**	2.1
WebThinker	0.162	0.151	0.204	0.076	0.1363	1.8
Open Deep Research	0.182	0.165	0.241	0.085	0.1526	2.7
Claude Opus 4.8						
Direct LLM	0.226	0.219	0.247	—	—	1.5
SuperWideSearch	**0.350**	**0.342**	**0.368**	**0.214**	**0.3089**	**3.9**
WebThinker	0.196	0.157	0.341	0.063	0.1562	2.4
Open Deep Research	0.173	0.161	0.217	0.076	0.1441	2.4
Qwen3.7-Max						
Direct LLM	0.284	0.281	0.290	—	—	2.1
SuperWideSearch	**0.376**	**0.371**	**0.392**	**0.241**	**0.3355**	**3.6**
WebThinker	0.224	0.195	0.332	0.094	0.1847	1.8
Open Deep Research	0.203	0.188	0.266	0.099	0.1718	3.0
GLM-5.2						
Direct LLM	0.294	0.291	0.302	—	—	2.4
SuperWideSearch	**0.354**	**0.347**	**0.373**	**0.221**	**0.3144**	**4.2**
WebThinker	0.185	0.152	0.314	0.070	0.1504	1.5
Open Deep Research	0.145	0.130	0.205	0.061	0.1197	1.2
KIMI-K2.6						
Direct LLM	0.291	0.288	0.300	—	—	2.7
SuperWideSearch	**0.337**	**0.331**	**0.352**	**0.206**	**0.2975**	**3.9**
WebThinker	0.168	0.131	0.309	0.058	0.1350	2.1
Open Deep Research	0.250	0.242	0.271	0.128	0.2132	1.8
DeepSeek-V4-Pro						
Direct LLM	0.285	0.283	0.291	—	—	2.4
SuperWideSearch	**0.343**	**0.335**	**0.367**	**0.193**	**0.2982**	**4.2**
WebThinker	0.184	0.154	0.302	0.070	0.1496	2.4
Open Deep Research	0.172	0.155	0.236	0.074	0.1426	1.8

Table 7: Main results on DeepSearchDAG *excluding single-claim instances* (330 instances). NodeF1, Recall, and Precision are reported for all settings; PES and DSD-Score are reported only for tool-augmented search frameworks with recorded trajectories. Direct LLM PES and DSD-Score entries are marked by dashes. SR denotes Success Rate.

Judge	Fleiss κ	Coverage flip (%)	Extraction agreement (%)	Mean inst. SD NodeF1	Mean inst. SD PES	Mean inst. SD DSD	SR agreement (%)
Qwen3.7-Max	0.892	8.1	99.6	0.0371	0.0418	0.0381	98.0
KIMI-K2.6	0.919	6.1	99.4	0.0275	0.0346	0.0293	99.0
DeepSeek-V4-Pro	0.909	6.7	98.1	0.0384	0.0469	0.0401	96.0

Table 8: Within-judge repeatability across three evaluations of the same 100 SuperWideSearch responses. "Mean inst. SD" is the mean, over instances, of the per-instance standard deviation across runs; it is not the standard deviation of the three corpus-level means.

to 0.0401, and SR agrees on 96–99% of instances. These results demonstrate strong temperature-0 repeatability while quantifying residual variability near semantic decision boundaries.

Cross-judge sensitivity. For each judge, Table 9 averages scores over its three runs. The reported metric ranges are the maximum minus minimum judge-level means. Cross-judge claim κ is computed after taking the majority covered/not-covered decision across the three runs of each judge.

SWS backbone	Qwen DSD	KIMI DSD	DeepSeek DSD	DSD range	NodeF1 range	PES range	SR range (pp)	Claim κ
Backbone 1	0.406	0.424	0.359	0.066	0.064	0.069	3.0	0.844

Table 9: Cross-judge sensitivity on the same fixed response set. Each judge value averages three complete runs; ranges are computed across the three judge models.

Majority claim decisions retain substantial cross-judge agreement ($\kappa = 0.844$). Mean DSD-Score ranges from 0.359 to 0.424, a range of 0.066; the corresponding NodeF1 and PES ranges are 0.064 and 0.069, and SR varies by 3.0 percentage points. We therefore use a fixed judge for the primary benchmark comparison and report this multi-judge experiment as a complementary sensitivity analysis.

D Quality Control Rules

The annotation process is governed by seven reproducible quality rules:

Rule 1: Decomposition Correctness. Sub-questions must faithfully preserve the original query's semantic intent. Modifier relationships (e.g., possessive structures) must not be incorrectly split. Each sub-question must be an atomic problem that can be independently verified, while dependencies between claims are allowed—for example, C_2 may depend on the answer of C_1.

Rule 2: Answer Uniqueness. If a query's key concept has multiple equally authoritative definitions (e.g., "tier-2 cities"), explicit authoritative source constraints are added. If no unique answer can be established even with constraints, the sub-question is deleted.

Rule 3: Temporal Consistency. All data/factual queries use "as of end of 2025" or a time cutoff appropriate for the business question as the temporal anchor. Queries with strong temporal words ("today," "recently," "currently") are deleted. Time-span queries that may change annually are decomposed into per-year enumerations.

Rule 4: Subjectivity Removal. Prediction-type questions ("predict," "forecast") are deleted. Subjective recommendation questions ("recommend," "best," "advice") are deleted. Subjective concepts with recognized quantitative substitutes are converted to objective metrics (e.g., "popularity" $\rightarrow$ "visitor count").

Rule 5: Data Scale Control. If estimated answer count exceeds 500, the query scope is progressively narrowed (geographic limitation, Top-N limitation, categorical stratification, temporal limitation) to 5–100 answers.

Rule 6: Comprehensive Checklist. After correction, each query is checked against: reference clarity, granularity, unique definition, temporal consistency, subjectivity, data volume, logical chain, and type classification.

Rule 7: Type Classification. Each sub-question is typed as enumeration ("what are" $\rightarrow$ entity list), numerical ("how much" $\rightarrow$ specific value), or factual description ("what is" $\rightarrow$ text description). Model-generated type labels undergo staged review and release-wide automated auditing.

E Annotation Workflow

Gold standard construction follows a **three-stage human annotation pipeline** applied to each claim after LLM initialization:

1. **Stage 1 — Annotation**: The primary annotator independently verifies the LLM-generated sub-question and answer. They may (i) revise the sub-question for clarity or accuracy, (ii) rewrite

the gold answer based on authoritative source lookup, (iii) record verified source URLs in the internal annotation record, (iv) add remarks, and (v) confirm or correct the claim type. Each annotator records their status (*completed/pending*).

2. **Stage 2 — Quality Control (QC)**: A second expert reviews the annotated result. The QC reviewer independently searches sources, may further revise the sub-question or answer, flags answer accuracy issues (e.g., *completeness*, *URL error*), and confirms the final claim type.
3. **Stage 3 — Re-check**: A senior reviewer audits the QC output, writes a detailed comment identifying any remaining issues, assigns a deduction score (0 = pass; -0.2 = minor issue; -0.4 = moderate; -1 = major error), and determines whether the claim is deliverable. The final gold answer is the QC answer unless the re-checker provides an explicit correction.

Claim-decomposition agreement. Before gold-answer population, two annotators independently reviewed each query's atomic-claim decomposition and assigned a claim type. Each proposed claim received one of three dispositions: *retain* when its boundary, wording, and type were appropriate; *revise* when its scope, atomicity, wording, or type required correction; and *remove* when it was duplicated, outside the query scope, or unsuitable as an independently verifiable claim. Annotators could also flag missing claims. Matching decisions were accepted, while disagreements over claim boundaries or types were resolved during quality control and, when necessary, senior re-check. This stage evaluates the reliability of claim decomposition and typing separately from gold-answer verification.

All annotator and reviewer identities are withheld for anonymity.

Milestone and dependency adjudication. Multiple annotators independently reviewed the milestone-construction rules and evaluated candidate milestone decompositions and dependency assignments using the same predefined rubric. The rubric checked milestone atomicity, claim ownership, dependency necessity, milestone type, and fan-in validity. Disagreements were resolved through adjudication, and the released skeleton was selected as the highest-scoring candidate under the rubric.

Source provenance and release scope. Verified source URLs are retained internally during annotation, quality control, and re-check to validate gold answers. The released artifact contains the queries, claims, gold answers, claim types, skeletons, and audit reports, while URL-level provenance records remain in the controlled annotation record.

Closed-loop Quality-Control Examples

The blue, green, and yellow blocks summarize the intermediate workflow state; each red block records the re-check finding together with the final correction, and the teal block verifies that the correction is reflected in the released artifact. "Artifact closure" means that the released sub-question, answer content, type, and machine-readable schema pass the stated release check.

Example 1 (dsd112/C2) — Scope and Answer Adequacy

Stage 0 — LLM Generation

Sub-question: What is the impact of AI on each occupational sub-category of "Information Security Engineering Technicians" in the 2022 *National Occupational Classification*?

Initial answer (excerpt): Generic Gartner/IDC statements about automating cybersecurity incident response.

Stage 1 — Annotation

Correction: The occupation is already a terminal category, so the unsupported "sub-category" scope was removed. The answer was rewritten to describe displaced routine work and the shift toward AI orchestration, model validation, adversarial analysis, and AI-security governance.

Type: Factual description.

Stage 2 — Quality Control (QC)

QC action: Adopted the corrected occupation-level scope and refined the answer wording. The factual claim type was retained.

Stage 3 — Re-check

Finding: The answer still did not operationalize "impact degree" and relied on generic references to AI rather than resolving the requested assessment.

Final correction: The claim was reframed as a replacement-likelihood assessment for the occupation. The final answer states that AI augments rather than completely replaces the role: routine operational tasks may be displaced, whereas unknown-threat analysis and innovative adversarial response remain human responsibilities.

Released-Artifact Closure

Closure check: The released C2 contains the corrected replacement-likelihood sub-question and final answer. Its factual type, occupation-level scope, and machine-readable record are consistent.

Status: Closed and included in the released artifact.

EXAMPLE 2 (DSD114/C5) — GEOGRAPHIC MISMATCH

Stage 0 — LLM Generation

Sub-question: List chairpersons of private undergraduate institutions across Hebei, Henan, Jiangsu, and Zhejiang.

Initial answer: Partial multi-province results.

Stage 1 — Annotation

Annotation action: Expanded the candidate institution list and retained enumeration type for subsequent scope review.

Stage 2 — Quality Control (QC)

QC action: Narrowed the sub-question to Hebei to make the claim atomic, but the revised answer inadvertently contained Henan institutions.

Stage 3 — Re-check

Finding: The answer was labelled Hebei but contained institutions from Henan, a critical geographic mismatch.

Final correction: The claim was rewritten as the atomic ordinal question "In the complete list of private undergraduate institutions in Hebei Province, what is the fifth institution?" The corrected gold answer is Hebei International Studies University (Shijiazhuang).

Released-Artifact Closure

Closure check: The released C5 uses the corrected Hebei-only scope, explicit ordinal, and gold answer. Its native enumeration array contains the same single institution.

Status: Closed and included in the released artifact.

EXAMPLE 3 (DSD118/C1) — ENUMERATION COMPLETENESS

Stage 0 — LLM Generation

Sub-question: Which Chinese companies have expanded into Thailand?
Initial answer: A broad regional description with only a few named companies.

Stage 1 — Annotation

Correction: Quantified the requested scope as 50 companies and constructed a 50-company Thailand list.
Type: Enumeration.

Stage 2 — Quality Control (QC)

QC action: Retained the 50-company list with formatting normalization; no answer-content or type issue was recorded.

Stage 3 — Re-check

Finding: The enumeration required an explicit cardinality and stable item boundaries for release-wide scoring.
Final correction: The final sub-question requests 50 companies in Thailand through December 2025. The gold answer was normalized to 50 named entries, beginning with BYD, Great Wall Motor, SAIC (MG), Changan, GAC Aion, NETA, Chery, Zhongce Rubber, Linglong Tire, and Prinx Chengshan.

Released-Artifact Closure

Closure check: The released C1 uses the corrected 50-company sub-question and enumeration type. Its native `gold_items` array contains exactly 50 entries, matching the requested cardinality.
Status: Closed and included in the released artifact.

F PROMPT TEMPLATES

The DSD-Score evaluation pipeline uses two core LLM prompts:

Extraction Prompt. Given a sub-question and the agent's full response, extract content relevant to the sub-question. If no relevant content exists, the claim is not processed further. The extractor preserves original wording and key facts without judging correctness.

Coverage Check Prompt. Given the sub-question, extracted answer, and gold answer, the check proceeds in two stages. Stage 1 verifies entity consistency and dimension consistency; claims failing either are classified as `not_covered`. Stage 2 branches by claim type: numerical claims use error-bound matching (relative error $\leq \tau_{\text{num}}$, exact match for rankings/years, absolute difference ≤ 5pp for percentages); an explicitly unavailable numerical gold requires abstention and rejects invented quantities. Enumeration claims use set-based $F_1^{\text{set}} \geq \tau_{\text{set}}$ with default $\tau_{\text{set}} = 0.60$; factual claims undergo fact consistency and information completeness verification. Only when all applicable checks pass is the claim classified as `covered`.

G BLIND EXTRACTION WITH ANAPHORA RESOLUTION

A key challenge in evaluating agent responses is that many claims' sub-questions contain *anaphoric references* to preceding claims (e.g., "the above enterprises," "C1," "the aforementioned model"). Extracting answers for such claims requires knowing what the preceding claims refer to. DSD-Score addresses this through a **blind two-phase extraction** process that resolves anaphora using

previously extracted answers from the agent's own response—never gold answers—thus preventing answer leakage during extraction.

G.1 Anaphora Detection

Before extraction begins, each claim's sub-question is classified by a deterministic regex-based module into one of three anaphora types (or `None` if no anaphora is present):

- **Explicit reference** (`explicit_CX`): The sub-question contains a direct claim ID reference such as "C1" or "C3," detected by the regex pattern `C\d+` with a negative lookbehind to avoid false positives from product codes like "NGC12V2304."
- **Chinese anaphoric word** (`cn_word`): The sub-question contains Chinese anaphoric expressions such as "shang shu" (the above), "qian shu" (the aforementioned), "qian wen" (the preceding text), "qian zhe" (the former), or "qi zhong" (among them). These are matched against a fixed keyword list.
- **Demonstrative "the said X"** (`gai_x`): The sub-question contains the pattern "gai X" (the said X, where "gai" is the Chinese demonstrative), where X is a domain noun (enterprise, model, product, platform, etc.), detected by a regex covering 30+ noun categories.

Claims classified as `None` (no anaphora) can be extracted independently; claims with any anaphora type require prior context from already-extracted answers.

G.2 Two-Phase Blind Extraction

Given an agent's response text and the ordered list of claims (sorted by claim ID), extraction proceeds in two phases:

Phase 1 — Batch extraction of non-anaphora claims. All claims without anaphora are sent to the LLM in a single batch prompt. By default, the agent response is truncated to 10,000 characters; the prompt then presents this text and the list of sub-questions, asking the LLM to extract the relevant content for each sub-question without judging correctness. If the agent's response does not mention a sub-question, the extractor returns `NOT_MENTIONED`. Crucially, no gold answers appear in this prompt.

Phase 2 — Sequential extraction of anaphora claims with context. Claims with anaphora are processed one by one in claim-ID order (ensuring that earlier claims, which may be referents, are extracted first). For each anaphoric claim, a *prior context* string is constructed from the already-extracted answers of preceding claims. This context is injected into the extraction prompt, enabling the LLM to first resolve the anaphoric reference (e.g., understanding that "the above enterprises" refers to the specific entities extracted in C1) and then locate the corresponding content in the agent's response.

G.3 Prior Context Construction

The construction of prior context differs by anaphora type to balance precision and coverage:

- **Explicit `CX` references:** Only the extracted answers of the specifically referenced claims (e.g., C1 and C3) are included. This targeted context minimizes noise.
- **Fuzzy anaphora (`cn_word`, `gai_x`):** All preceding extracted answers are included, starting from the most recent and proceeding backward. This broader context ensures that the referent—which may be any preceding claim—is captured.

G.4 Example (DSD2)

The query asks for the top two centralized long-stay apartment operators by management and opening scale, followed by their WeChat official accounts, mini-programs, websites, and business details. This yields five claims with a star-shaped dependency (C2–C5 all depend on C1):

- **C1** (enumeration, M1/identify): "Which two enterprises rank in the top two by centralized long-stay apartment management scale and opening scale?" — *no anaphora*.
- **C2** (factual, M2/lookup): "What are the WeChat official accounts of the *above top two* enterprises' brands?" — *anaphora: cn_word* ("above").
- **C5** (factual, M5/lookup): "What is the centralized long-stay apartment business information of the *above top two* enterprises?" — *anaphora: cn_word* ("above").

The extraction proceeds as follows:

Step 1 (Phase 1). C1 has no anaphora and is sent to the batch extraction prompt along with any other non-anaphora claims. The LLM extracts from the agent's response: e.g., "Vanke (Boyu) and Longfor (Guanyu)." This *extracted* answer (not the gold answer) is stored.

Step 2 (Phase 2, C2). C2 contains the anaphoric word "above." The prior context is built from C1's *extracted* answer: "C1: Vanke (Boyu) and Longfor (Guanyu).". This context is injected into the extraction prompt, allowing the LLM to understand that "the above top two enterprises" refers to Vanke and Longfor, and then search the agent's response for their WeChat official accounts.

Step 3 (Phase 2, C5). Similarly, C5's prior context includes C1's extracted answer (and C2–C4 if already extracted). The LLM resolves "the above top two enterprises" and extracts the business information from the agent's response.

Step 4 (Judgment). Only after *all* claims are extracted, the extracted answers are compared against gold answers using type-aware judgment: deterministic matching for numerical and enumeration claims, LLM-based four-condition coverage checking for factual claims.

This blind extraction design ensures that the extraction LLM never sees gold answers, eliminating the risk of answer leakage while still correctly resolving inter-claim dependencies through the agent's own extracted responses.

H CLAIM-TO-MILESTONE AGGREGATION

The gold skeleton is generated from atomic claims by a construction-time procedure. Below are the released core data structures and LLM prompt.

Data Structures.

```
@dataclass
class Milestone:
    milestone_id: str
    description: str
    information_type: str
        # "identify"/"lookup"/"verify"
    claim_ids: List[str]
    forkable: bool = False
    fork_count: int = 1
    fork_description: str = ""

@dataclass
class Dependency:
    source_id: str
    target_id: str
    reason: str
    dep_type: str = "sequential"
        # "sequential" | "fan_in"

@dataclass
class Skeleton:
    query: str = ""
    has_dependency_chain: bool = False
    milestones: List[Milestone]
    dependencies: List[Dependency]
```

LLM Prompt. The script sends the following prompt to an LLM to aggregate claims into a skeleton DAG (key rules highlighted in bold):

```
You are a DeepSearch search process
analysis expert. Given a complex query
and its decomposed atomic sub-problems
(claims), aggregate claims into a
"search skeleton" (Milestone nodes +
Dependency edges).

## Core Principles

### Milestone Definition
A milestone is a logical grouping of
claims, representing one information-
gathering stage. A milestone contains
zero or more related claims; an empty
milestone is a structural routing node
whose coverage is judged from its
description and the agent response.

### Milestone Types (only 3 allowed)
- "identify": Determine target entity
  (e.g., "find qualifying companies").
  Starting point, no prerequisites.
- "lookup": Find attributes of known
  entity. Must depend on identify.
- "verify": Cross-validation. Same
  question reached by multiple paths.
  Must have fan-in (2+ sources).

**"aggregate" type is FORBIDDEN**---
violates atomicity & verifiability.

### Correct Fan-in Definition
Fan-in = same objective question
independently reached by multiple
search paths -> converge on verify.
- OK: company district via Tianyancha
  + Qichacha -> one verify node
- OK: official spec + third-party test
  -> one verify node
- WRONG: different questions' answers
  summarized into new task (=aggregate)
- verify node's incoming edges all
  marked dep_type="fan_in"

### Dependency Edges
- Only mark necessary dependencies:
  "cannot build M_j without M_i result"
- If query lists all entities already,
  no dependency; milestones parallel
- Multi-hop queries: full dependency
  chain required

## Output (JSON)
{"has_dependency_chain": bool,
 "milestones": [{
   "milestone_id": "M1",
   "description": "...",
   "information_type": "identify/
     lookup/verify",
   "claim_ids": ["C1","C2"],
   "forkable": bool,
   "fork_count": 1,
   "fork_description": "..."
 }],
 "dependencies": [{
   "source_id": "M1",
   "target_id": "M2",
   "reason": "...",
   "dep_type": "sequential/fan_in"
 }]}

## Notes
1. Each claim -> exactly one milestone
2. Typically 3-10 milestones
3. forkable: can split into parallel
   sub-tasks (multiple entities)
4. fan-in: verify node needs 2+
```

```
   fan_in edges
5. type must be identify/lookup/verify
6. verify nodes: 2+ fan_in edges
```

The "3–10" note is a soft drafting heuristic for initial generation, not a dataset validity constraint. Human review may merge or split stages as required by the query, so the released skeleton uses a query-dependent milestone count. The resulting milestones and dependency edges form the gold skeleton DAG, which is then reviewed and corrected by human annotators.

H.1 Released Dataset Integrity Audit

The released supplement includes `validate_dataset.py`, which treats every condition in the integrity-audit table below as a hard failure and exits nonzero on any violation. The release audit checks unique claim ownership, milestone and claim identifiers, edge endpoints, edge uniqueness, acyclicity, dependency flags, and fan-in typing over every instance.

Hard-failure condition	**Released violations**
Claim assignment $\neq 1$	0
Dangling edge endpoint	0
Duplicate edge	0
Self-loop	0
Directed cycle	0
Dependency-flag mismatch	0
`verify` fan-in violation	0
Non-`verify` fan-in edge	0
Duplicate claim ID (within instance)	0
Duplicate milestone ID (within instance)	0
Instances audited	348
Gold claims audited	10,617

Table 10: *
Released integrity audit generated by `validate_dataset.py`. Every listed condition is a hard failure; the released audit report records `PASS`.

The companion `claim_quality_audit.py` scans all claims for normalized within-instance duplicates, identical text with conflicting gold answers, placeholders or fragment patterns, question/answer type mismatch, plural requests paired with a single parsed value, and URL/location/company semantic-type mismatch. These heuristic findings are exported as CSV review candidates.

I Concrete Query-to-Skeleton Examples

Below are four concrete DeepSearchDAG instances. In each, claims and their gold answers are highlighted in blue; milestones and the skeleton are highlighted in yellow. *Note: The original queries and claims are in Chinese; English translations are provided here for readability, and the original Chinese text is available in the released dataset.*

I.1 Example 1: A Simple Diamond DAG

Query. Did the CEO of NVIDIA sell more shares than the CEO of AMD in 2024?

Claims (sub-questions) and gold answers.

- **C1.** Who is the current CEO of NVIDIA? → Jensen Huang
- **C2.** Who is the current CEO of AMD? → Lisa Su
- **C3.** How many shares did Jensen Huang sell in 2024? → 120,000 shares
- **C4.** How many shares did Lisa Su sell in 2024? → 80,000 shares
- **C5.** Did Jensen Huang sell more shares than Lisa Su in 2024? → Yes

Milestones.

- **M1** (identify): Identify the CEO of NVIDIA. Claims: {C1}
- **M2** (identify): Identify the CEO of AMD. Claims: {C2}
- **M3** (lookup): Look up the 2024 share sales of NVIDIA's CEO. Claims: {C3}
- **M4** (lookup): Look up the 2024 share sales of AMD's CEO. Claims: {C4}
- **M5** (verify): Verify whether NVIDIA's CEO sold more shares than AMD's CEO. Claims: {C5}

Skeleton (dependency edges).

- $M_1 \rightarrow M_3$ (sequential): cannot look up NVIDIA CEO's sales without first identifying the CEO.
- $M_2 \rightarrow M_4$ (sequential): cannot look up AMD CEO's sales without first identifying the CEO.
- $M_3 \rightarrow M_5$ (fan_in): verify node converges results from both lookup branches.
- $M_4 \rightarrow M_5$ (fan_in): verify node converges results from both lookup branches.

This example exhibits a diamond-shaped DAG: two independent identify branches (M1, M2) feed into their respective lookup milestones (M3, M4), which then fan in to a single verify milestone (M5).

I.2 Example 2: A Complex Multi-Country Comparison DAG

This example demonstrates fine-to-coarse aggregation: 13 atomic claims are grouped into only 7 milestones, and the skeleton contains both fan-out and fan-in structures.

Query. For the three largest new energy vehicle (NEV) markets in 2024, what were their subsidy policies and market outcomes, and how do they compare?

Claims (sub-questions) and gold answers.

- **C1.** Which were the three largest NEV markets by sales volume in 2024? → China, United States, Germany
- **C2.** What was China's 2024 NEV subsidy policy? → Purchase-tax exemption extended
- **C3.** What was the US 2024 federal EV tax credit policy? → Up to $7,500 per eligible vehicle
- **C4.** What was Germany's 2024 EV subsidy policy? → Subsidies ended in December 2022
- **C5.** What was China's 2024 NEV sales volume? → 12.8 million units
- **C6.** What was the US 2024 EV sales volume? → 1.3 million units
- **C7.** What was Germany's 2024 EV sales volume? → 0.7 million units
- **C8.** What was China's 2024 NEV market share? → 40.9%
- **C9.** What was the US 2024 NEV market share? → 10%
- **C10.** What was Germany's 2024 NEV market share? → 18%
- **C11.** Which country had the highest 2024 NEV sales volume? → China
- **C12.** Which country had the highest 2024 NEV market share? → China
- **C13.** How did the three countries' 2024 NEV subsidy policies differ? → China extends incentives; US offers tax credits; Germany ended subsidies

Milestones.

- **M1** (identify): Identify the three largest NEV markets in 2024. Claims: {C1}
- **M2** (lookup): China's 2024 NEV policy and market. Claims: {C2, C5, C8}
- **M3** (lookup): US 2024 NEV policy and market. Claims: {C3, C6, C9}
- **M4** (lookup): Germany's 2024 NEV policy and market. Claims: {C4, C7, C10}
- **M5** (verify): Compare NEV sales volume across the three countries. Claims: {C11}
- **M6** (verify): Compare NEV market share across the three countries. Claims: {C12}
- **M7** (verify): Compare NEV subsidy policies across the three countries. Claims: {C13}

Skeleton (dependency edges).

- $M_1 \to M_2$, $M_1 \to M_3$, $M_1 \to M_4$ (fan-out): identifying the target countries enables parallel per-country lookups.
- $M_2 \to M_5$, $M_3 \to M_5$, $M_4 \to M_5$ (fan_in): sales-volume comparison requires data from all three countries.
- $M_2 \to M_6$, $M_3 \to M_6$, $M_4 \to M_6$ (fan_in): market-share comparison requires data from all three countries.
- $M_2 \to M_7$, $M_3 \to M_7$, $M_4 \to M_7$ (fan_in): policy comparison requires data from all three countries.

This example shows a richer DAG topology: one identify milestone fans out to three parallel lookup milestones, each aggregating multiple claims; the three lookup milestones then fan in to three distinct verify milestones. The resulting 13-edge skeleton cannot be captured by a simple tree or chain.

I.3 Example 3: A Case with Prerequisite Error (No Answer) — dsd38

We deliberately retain instances where the query's underlying premise is incorrect, making certain claims unanswerable via public search. Such cases test whether agents can recognize and report premise errors rather than hallucinating answers.

Query (dsd38). Compile comprehensive information about a consumer body fat scale and weight scale, including sensor types, accuracy, battery life, Bluetooth compatibility, and *medical device registration numbers, registrants, registration dates, and intended-use statements* (80 sub-questions total).

Selected claims and gold answers (medical-device-related).

- **C33.** What is the medical device registration number of the body fat scale S200? → No medical device registration number
- **C34.** What is the medical device registration number of the body fat scale S400? → No medical device registration number
- **C74.** Is the weight scale S200 a medical device? → This product is not a medical device.
- **C77.** Does the body fat scale S400 possess a valid medical device registration number? → No

Milestones and skeleton (excerpt).

- **M1** (identify): Confirm official product names and model mappings for both devices, and verify their medical-device status. Claims: {C33, C34, C74, C77, ...}
- **M8** (lookup): Look up S200 medical device compliance info: registration number, registrant, date, intended use. Claims: {C33, C35, C37, C39, C74} — depends on M1
- **M9** (lookup): Look up S400 medical device compliance info. Claims: {C34, C36, C38, C40, C77, C78} — depends on M1

Edges: $M_1 \to M_8$, $M_1 \to M_9$ (fan-out from identify); plus $M_1 \to M_2, \ldots, M_7, M_{10}, \ldots, M_{12}$ for non-medical claims (omitted for brevity).

Case analysis. The query assumes both scales are medical devices and asks for their registration numbers, registrants, dates, and intended-use statements (C33–C40). However, the gold answers reveal that *neither product is a registered medical device*—they are consumer wellness products. Claims C33–C40 and C74, C77 are therefore unanswerable: their premises are false. An agent that fabricates a plausible-looking registration number would be penalized under DSD-Score, while an agent that correctly reports "not a medical device" would be rewarded. Such cases are deliberately retained for three reasons: (1) they test whether agents can recognize incorrect premises rather than fabricating plausible-sounding answers; (2) they reflect real-world business search needs where users may hold incorrect assumptions; and (3) the ability to report "no answer found due to premise error" is a valuable agent capability that result-only metrics cannot evaluate. In DSD-Score, these claims are classified as *not-covered* if the agent fabricates an answer, and the dependency edges $M_1 \to M_8$ and $M_1 \to M_9$ are scored as 0 if M_1 is covered (agent correctly identifies the products) but M_8/M_9 are not (agent fails to report the premise error).

I.4 Example 4: Information Beyond Search Engine Reach — dsd2

This example illustrates a key advantage of human-constructed gold standards: certain information required by real-world business queries is not indexed by any public search engine, yet human annotators can locate it through multi-step navigation.

Query (dsd2). Identify the top two centralized long-stay apartment operators by management and opening scale, then compile their WeChat official accounts (account name, WeChat ID, certified entity, main columns and content), WeChat mini-programs (name, certified entity, main columns), and official website information.

Selected claims and gold answers.

- **C1.** Which two enterprises rank in the top two by centralized long-stay apartment management scale and opening scale? → Operator A, Operator B
- **C2.** What are the WeChat official accounts of the above top two enterprises' long-stay apartment brands? → *Brand A* (ID: account_a, certified: Company A); *Brand B* (ID: account_b, certified: Company B)
- **C3.** What are the WeChat mini-programs of the above top two enterprises' brands? → *Brand A* (certified: Tech Company A); *Brand B* (certified: Management Company B)

Milestones and skeleton.

- **M1** (identify): Identify the top two operators. Claims: {C1}
- **M2** (lookup): WeChat official accounts. Claims: {C2} — depends on M1
- **M3** (lookup): WeChat mini-programs. Claims: {C3} — depends on M1

Edges: $M_1 \to M_2$, $M_1 \to M_3$, $M_1 \to M_4$, $M_1 \to M_5$ (fan-out from identify).

Case analysis. WeChat official accounts and mini-programs reside inside the WeChat ecosystem and have *no browser-accessible external homepage links* that search engines can index. The annotator located this information by: (1) visiting each company's official website; (2) finding WeChat QR codes placed at the bottom of these pages; and (3) scanning the QR codes to access the accounts and record their names, IDs, certified entities, and column structures. No web search query—whether on Google, Bing, or Baidu—can directly return the WeChat ID or the mini-program certified entity. This case demonstrates that human-constructed gold standards capture information layers inaccessible to automated retrieval, ensuring that DSD evaluates agents on truly comprehensive search capability rather than mere search-engine proficiency. In DSD-Score, if an agent fails to find C2 and C3 (as expected for most current agents), these claims are classified as *not-covered*, and the dependency edges $M_1 \to M_2$ and $M_1 \to M_3$ become lucky-hit or cascade-fail edges if M_1 is covered.

J Node-Level Mapping from Existing Benchmarks to DSD

Representative search-agent benchmark paradigms can be situated in a common node-level view. Table 11 provides a topology-level mapping into DSD. This unification concerns the shared representation and evaluation interface; scores are not assumed to be numerically interchangeable with benchmark-specific metrics reported under different protocols.

BrowseComp → Single Factual Claim. A BrowseComp instance asks for one hidden entity (e.g., "find the person who did X"). In DSD, this maps to a single factual-description claim with no milestone and no dependency edge. DSD-Score degenerates to NodeF1, and since the claim type is factual, coverage is judged by LLM-based exact-match verification—functionally equivalent to BrowseComp's original exact-match metric.

DeepSearchQA → Single Enumeration Claim. A DeepSearchQA instance asks for a set of entities matching a criterion. In DSD, this maps to a single enumeration claim with no checked dependency edges. DSD-Score degenerates to NodeF1, and coverage is judged by set-based $F_1^{\text{set}} \geq \tau_{\text{set}}$ with default $\tau_{\text{set}} = 0.60$.

WideSearch → Single-Layer Parallel Enumeration. A WideSearch instance asks agents to systematically find and organize entities into a structured table. In DSD, each row or dimension maps to a parallel enumeration claim, all residing in a single milestone layer with no dependency edges. DSD-Score degenerates to the average NodeF1 across claims, mirroring WideSearch's row/item/column F1.

FIRE-Bench → Tree-Structured Claims. FIRE-Bench provides claim-level precision and recall but uses a tree structure. In DSD, this maps to multiple claims organized in a tree topology without cross-branch dependency edges. PES is not applicable (no edges), so DSD-Score degenerates to

Benchmark	Paradigm	DSD Mapping	Nodes	Edges	DSD-Score
BrowseComp	Single Answer	1 factual claim, no checked edges	1	0	NodeF1
DeepSearchQA	Set Answer	1 enumeration claim, no checked edges	1	0	NodeF1 (F_1^{set})
WideSearch	Set (wide)	Single-layer parallel enum. claims	≥ 1	0	NodeF1 (F_1^{set})
FIRE-Bench	Tree	Tree claims, no cross-branch edges	≥ 2	0	NodeF1
DeepWideSearch	Multi-hop+Set	Latent intermediate hops; final set only	1	0	NodeF1 (F_1^{set})
DeepSearchDAG	**DAG**	**Identify→lookup→verify, fan-in/out**	≥ 2	≥ 1	w_1N+w_2PES

Table 11: Unified mapping from existing benchmarks to DSD's node-level framework. Each paradigm corresponds to a restricted topology: BrowseComp reduces to a single factual-claim node; DeepSearchQA reduces to a single enumeration-claim node; WideSearch reduces to a single layer of parallel enumeration claims without dependency edges; FIRE-Bench reduces to a tree without cross-branch edges; DeepWideSearch reduces to a single checked node (the final set answer) with latent intermediate hops and no checked dependency edges. Only DeepSearchDAG exercises the DAG topology with checked nodes, checked edges, fan-out, and fan-in. "N"=NodeF1, "PES"=Prerequisite Edge Score.

NodeF1—subsuming FIRE-Bench's claim-level evaluation while extending to DAG topologies with cross-branch dependencies.

DeepWideSearch → Latent Multi-hop, Single Checked Node. A DeepWideSearch instance requires multi-hop reasoning before aggregating a set of entities. Although the reasoning chain contains multiple hops, the evaluation only verifies the final aggregated set—all intermediate hops remain latent (no ground truth). In DSD, this maps to a single checked enumeration-claim node with no checked dependency edges. DSD-Score degenerates to NodeF1 (F_1^{set}), same as DeepSearchQA, because PES requires checked edges between independently verified nodes.

DeepSearchDAG (DAG). Only DSD exercises the complete DAG topology: fan-out from identify to parallel lookups, fan-in convergence at verify nodes, and cross-branch dependencies—none of which can be expressed by the restricted topologies above.

K Answer–Dependency Coverage Analysis

A key motivation for PES is that final-answer coverage alone does not describe whether a search agent followed the prerequisite structure during tool use. PES measures trajectory adherence on milestone pairs linked by prerequisite edges. A naive summary statistic, the mean difference $\Delta = \overline{\text{NodeF1}} - \overline{\text{PES}}$, is an unreliable indicator of this structural pattern: positive and negative instance-level gaps cancel under averaging, and Δ conflates distinct edge-level coverage patterns. We therefore report two complementary conditional diagnostics.

Table 12 reports **answer coverage** (NodeF1) and **trajectory prerequisite coverage** (PES), together with a **potential lucky-hit signature** and the **downstream-without-upstream execution pattern**. The diagnostics are computed only for tool-augmented search frameworks with recorded trajectories.

Potential lucky-hit signature. This instance-level conditional probability generalizes the binary Success Rate (SR). Whereas SR is all-or-nothing—requiring full answer coverage, every sequential edge to be covered, and at least one incoming edge to be covered for each fan-in verify node—this diagnostic asks: *among instances that clear the answer-coverage bar, how many fall below the prerequisite-linked coverage bar?* Let $A_i = \mathbb{1}[\text{NodeF1}_i \geq \tau]$ denote an answer-coverage pass and $D_i = \mathbb{1}[\text{PES}_i \geq \tau]$ a prerequisite-linked coverage pass for instance i, where $\tau = \tau_m = 0.5$ and N

is the number of benchmark instances:

$$\text{Potential lucky-hit signature} = \frac{\sum_{i=1}^{N} A_i(1 - D_i)}{\sum_{i=1}^{N} A_i} = \Pr(D_i = 0 \mid A_i = 1).$$

The numerator counts instances that pass the answer-coverage criterion while falling below the prerequisite-linked coverage criterion; the denominator counts all answer-coverage passes. The statistic is undefined when the denominator is zero. At $\tau = 0.5$, it is more sensitive than SR because it does not require full coverage.

Downstream-without-upstream execution pattern. For each edge $M_u \rightarrow M_v$, the trajectory records whether the upstream and downstream tool actions occurred and whether the downstream action used the upstream result. The reported percentage is the proportion of edges with downstream tool activity but no observable use of the upstream result.

System	Answer coverage	Prerequisite-linked coverage	Potential lucky-hit signature	Downstream-without-upstream coverage pattern (%)
SuperWideSearch (Backbone 1)	0.380	0.251	0.464	13.6
SuperWideSearch (Backbone 2)	0.356	0.230	0.384	12.9
SuperWideSearch (Backbone 3)	0.345	0.221	0.475	13.1
SuperWideSearch (Backbone 4)	0.355	0.226	0.458	11.6
SuperWideSearch (Backbone 5)	0.334	0.200	0.445	9.7
SuperWideSearch (Backbone 6)	0.343	0.200	0.496	12.0
WebThinker (Backbone 1)	0.232	0.109	0.516	7.8
WebThinker (Backbone 5)	0.287	0.176	0.464	9.9
WebThinker (Backbone 2)	0.195	0.087	0.490	5.6
WebThinker (Backbone 6)	0.192	0.083	0.390	5.4
WebThinker (Backbone 4)	0.203	0.077	0.511	5.1
Open Deep Research (Backbone 3)	0.251	0.136	0.403	10.4
Open Deep Research (Backbone 5)	0.252	0.141	0.481	10.0
Open Deep Research (Backbone 1)	0.207	0.108	0.456	8.9
Open Deep Research (Backbone 2)	0.155	0.075	0.294	6.6
Open Deep Research (Backbone 6)	0.177	0.085	0.350	7.5
Open Deep Research (Backbone 4)	0.181	0.089	0.351	7.5

Table 12: Trajectory-based answer–dependency diagnostics for tool-augmented search frameworks. Answer coverage is NodeF1; trajectory prerequisite coverage is PES. The potential lucky-hit signature is the conditional fraction of instances with NodeF1 ≥ 0.5 but PES < 0.5. The downstream-without-upstream pattern is the fraction of dependency edges whose downstream tool activity lacks observable use of the upstream result. Direct LLM baselines are not included because they do not produce tool trajectories.

K.1 Configuration-Level NodeF1–PES Comparison

Because PES is undefined for the eight Direct LLM settings, we compute the association over the 24 tool-augmented configurations in Table 6. NodeF1 and PES are strongly correlated but not identical (Pearson $r = 0.988$, Spearman $\rho = 0.987$). For example, WebThinker with Claude Opus 4.8 and Open Deep Research with Qwen3.7-Max have similar NodeF1 values (0.203 vs. 0.207) but different PES values (0.077 vs. 0.108). Similarly, SuperWideSearch with KIMI-K2.6 and DeepSeek-V4-Pro have NodeF1 values of 0.345 and 0.343 but PES values of 0.221 and 0.200, respectively. These configuration-level results support using PES as a diagnostic process signal rather than as an independent model-ranking objective. Instance-level correlation requires the private per-instance trajectory records used for the reported framework runs.

Structural pattern. Consider an instance where the gold skeleton contains an edge $M_1 \rightarrow M_2$ (identify $\rightarrow$ lookup). If the agent performs a lookup action for M_2 without an observable upstream result from M_1 being used to construct that action, the edge receives zero PES credit even if the final answer later contains information about M_2. The downstream-without-upstream column counts such execution breaks; the potential lucky-hit signature lifts the same diagnostic to the instance level.

Why not the mean gap Δ. Across the tool-augmented configurations reported in Table 12, the mean $\Delta = \overline{\text{NodeF1}} - \overline{\text{PES}}$ is an incomplete summary because it mixes answer quality with process

adherence. The potential lucky-hit signature and downstream-without-upstream execution pattern expose distinct trajectory failures that Δ averages away.

Handling unsupported answers. NodeF1 evaluates whether the final response states correct information, while trajectory-aware PES evaluates whether the search process records prerequisite-linked tool behavior. The two signals are therefore reported separately: NodeF1 is available for LLM-only baselines, whereas PES and DSD-Score are defined for tool-augmented search agents with recorded trajectories.

L Error Analysis Examples

We provide detailed case studies of four representative agent errors:

(1) Lack of reflection. The agent encounters a tool call error, immediately concludes the task is unsolvable, and outputs an empty table without attempting alternative search strategies.

(2) Over-reliance on internal knowledge. The agent correctly identifies the core entity but generates the entire answer table from pre-trained knowledge without performing any web queries, resulting in outdated pricing information.

(3) Context overflow. After 15 search calls and 8 webpage visits, the agent's context window fills up, causing it to lose early search results and produce an incomplete answer.

(4) Insufficient retrieval. The agent identifies a relevant webpage through search but fails to extract key information during the visit step, as the LLM summarization omits critical details buried in the page.

M Experimental Setup Details

All agent systems utilize identical tools: (1) Google Search API; and (2) Webpage Visit tool. Since HTML webpages are often very lengthy, we use the same LLM in the agents to summarize HTML into concise text. The cost of this summarization is counted into efficiency metrics. We utilize the official API endpoints with default decoding parameters.

N Structural Dimension Formulas

This section provides the precise computation formulas for the four structural dimensions (Width, Depth, Members, CTH) used in the benchmark comparison figure (Figure 1 in the main paper). All metrics are computed per DeepSearchDAG instance and then averaged across the 348-instance dataset.

N.1 Notation

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ denote the set of n claims in one DSD instance. Each claim c_i has:

- text(c_i): the claim's question text.
- type$(c_i) \in \{\textit{enumeration}, \textit{numerical}, \textit{factual}\}$: the claim type.
- answer(c_i): the gold-standard answer string (for enumeration claims, a structured list of entities).
- id(c_i): the claim identifier (e.g., "C1").

Let $\mathcal{M} = \{m_1, \ldots, m_K\}$ denote the milestone set from the instance skeleton, where each milestone m_k has an ordered claim list claims$(m_k) \subseteq \mathcal{C}$. Let $\mathcal{D}_{\text{ms}}$ denote the set of milestone-level dependency edges $(m_j \to m_k)$ from the skeleton.

N.2 VOLUME

The Volume is the total number of claims in the instance:

$$V = |\mathcal{C}| \tag{6}$$

N.3 CLAIM TYPE HETEROGENEITY (CTH)

CTH measures the diversity of claim types using the Simpson diversity index. Let $p_t = |\{c \in \mathcal{C} : \text{type}(c) = t\}|/|\mathcal{C}|$ be the proportion of claims of type t:

$$\text{CTH} = 1 - \sum_t p_t^2 \tag{7}$$

CTH ranges from 0 (all claims share the same type, maximal homogeneity) to $1 - 1/T$ (claims uniformly distributed across T types, maximal heterogeneity). For the three DSD claim types, $\text{CTH} \in [0, \frac{2}{3}]$.

N.4 ENTITY COUNT

The entity count E captures the number of distinct entities investigated in the instance. It is computed by extracting the entity prefix from each claim text (the substring before the first occurrence of the Chinese possessive marker "de", filtering out anaphora and temporal phrases):

$$E = |\{\text{entity-prefix}(\text{text}(c)) \ : \ c \in \mathcal{C}\}| \tag{8}$$

N.5 MEMBERS (COMPOSITE)

Members is a composite metric that integrates scale, heterogeneity, and entity diversity:

$$M = V \times (1 + \text{CTH}) + E \tag{9}$$

The $(1 + \text{CTH})$ factor amplifies the contribution of heterogeneous instances, while E adds a bonus for entity breadth. This formulation ensures that two instances with the same claim count but different type distributions or entity counts receive different Members scores.

N.6 WIDTH

Width measures the total factual breadth by summing answer items across all enumeration-type claims:

$$W = \sum_{\substack{c \in \mathcal{C} \\ \text{type}(c) = \textit{enumeration}}} \text{count-items}(\text{answer}(c)) \tag{10}$$

where count-items$(\cdot)$ counts the number of entity entries in a gold answer (via numbered-line detection or comma-separated item splitting). If no enumeration claims exist, $W = V$ as a fallback.

N.7 DEPTH (EXPANDED CLAIM-LEVEL STRUCTURAL DEPTH)

Depth is the longest path in a deterministic expanded claim graph that integrates three types of edges. It is a workload descriptor for serialized claim structure; it is not the observed length of an agent trajectory, nor does every edge assert a manually annotated causal prerequisite:

(1) Within-milestone order edges. Claims within the same milestone are serialized in ID order:

$$\begin{aligned} &\forall m \in \mathcal{M}, \quad \forall i \in \{1, \ldots, |\text{claims}(m)| - 1\} : \\ &\qquad \text{claims}(m)[i] \to \text{claims}(m)[i + 1]. \end{aligned} \tag{11}$$

These adjacent order edges are a deterministic serialization convention used only for this expanded structural descriptor. They must not be interpreted as human-annotated causal edges or as evidence that an agent executed the claims in this order.

(2) Cross-milestone dependency edges. For each milestone dependency $(m_j \to m_k) \in \mathcal{D}_{\text{ms}}$, the last claim of m_j is linked to the first claim of m_k:

$$\text{last}(\text{claims}(m_j)) \;\to\; \text{first}(\text{claims}(m_k)) \tag{12}$$

(3) Text-based reference edges. Two patterns are detected from claim texts:

- **Explicit C-X references**: if $\text{text}(c_i)$ mentions "Cj" and $c_j \in \mathcal{C}$, add edge $c_j \to c_i$.
- **Anaphora**: if $\text{text}(c_i)$ contains anaphora keywords (e.g., Chinese phrases meaning "the above-mentioned" or "among them"), add edge $c_{i-1} \to c_i$ (dependency on the preceding claim).

Longest path computation. The combined directed graph $G = (\mathcal{C}, E_{\text{order}} \cup E_{\text{cross}} \cup E_{\text{text}})$ is processed via topological sort (Kahn's algorithm) followed by longest-path dynamic programming:

$$\text{dist}(v) = \max_{(u \to v) \in G} \big(\text{dist}(u) + 1\big), \quad D = \max_{v} \text{dist}(v) \tag{13}$$

N.8 Max Fan-in Intensity (MFI)

MFI measures the maximum number of incoming dependency edges at any single milestone node, reflecting the intensity of multi-evidence convergence:

$$\text{MFI} = \max_{m_k \in \mathcal{M}} |\{m_j : (m_j \to m_k) \in \mathcal{D}_{\text{ms}}\}| \tag{14}$$

N.9 Summary

Table 13 summarizes the dimensions and their computation sources.

Dimension	Formula	Source
Volume V	$\lvert\mathcal{C}\rvert$	Gold claims
CTH	$1 - \sum_t p_t^2$	Gold claims (type)
Entity E	distinct entity prefixes	Gold claims (text)
Members M	$V \times (1 + \text{CTH}) + E$	Gold claims
Width W	$\sum$ count-items(answer)	Gold claims (answer)
Depth D	longest path in expanded G	Skeleton + gold claims
MFI	max milestone fan-in	Skeleton

Table 13: Structural dimension formulas and data sources.

O Domain Distribution

Table 14 details the domain distribution of DeepSearchDAG across 19 mutually exclusive primary business domains. Every instance receives exactly one primary-domain label, and no residual "Others" bucket is used. The counts were regenerated from a deterministic primary-topic audit of the final released query and claim texts.

Domain	Count (%)
Enterprise & Industry	92 (26.4%)
Finance & Investment	39 (11.2%)
AI Products & Ecosystem	28 (8.0%)
E-commerce & Brands	27 (7.8%)
Education & Admissions	22 (6.3%)
Healthcare & Medicine	17 (4.9%)
Energy & Environment	16 (4.6%)
Academic Research	12 (3.4%)
Real Estate & Construction	11 (3.2%)
Entertainment & IP	11 (3.2%)
Media & Communications	10 (2.9%)
Agriculture & Food	9 (2.6%)
Society & Employment	9 (2.6%)
Transportation & Mobility	9 (2.6%)
Gov Policy & Public Admin	8 (2.3%)
Travel & Hospitality	8 (2.3%)
Legal & Compliance	8 (2.3%)
Geography & Regional	6 (1.7%)
Consumer Products & Services	6 (1.7%)
Total	**348 (100%)**

Table 14: Domain distribution of DeepSearchDAG. Each of the 348 instances is assigned to exactly one of 19 primary business domains.