

Article

CustomNerd: A Framework and Tool for Fast Deployment of Production-Ready Expertise-Based Question Answering Systems

Harsh Kashyap¹, Shela Wu², Daksh Agarwal³, Fengcheng Jiang², Euijae Kim², Elyse Herman⁴, Jacob Dickman⁵ and Dennis Shasha^{2,*}

¹ Apple; harsh_kashyap@apple.com

² New York University; shela.wu@nyu.edu; (S.W.); fj2080@nyu.edu (F.J.); ek3955@nyu.edu (E.K.)

³ Guru Gobind Singh Indraprastha University; daksh.10819051623@ipu.ac.in

⁴ McGill University; elyse.herman@desjardins.com

⁵ Rutgers University; jd1636@scarletmail.rutgers.edu

* Correspondence: shasha@cs.nyu.edu

Abstract

CUSTOMNERD is a modular framework to build domain-specific expertise systems using a common set of core components. The system is designed to support multiple domains and has already been deployed to instantiate expertise-driven systems in nutrition (DietNerd), news (NewsNerd), cloud technology (CloudNerd), and space science (SpaceNerd). CUSTOMNERD incorporates specialized knowledge from domain-specific data sources that a user provides and processes the results of searches on those sources. The system consists of five main components: (i) Query Conversion, which converts user input into structured queries for one or more application-specific databases of validated articles; (ii) Extraction, which uses the structured queries to extract relevant data from those external databases; (iii) Filtering and Classification, which identifies and prioritizes relevant documents among those retrieved; (iv) Reliability Assessment, which evaluates the quality of the documents (e.g., statistical validity, absence of conflict of interest); and (v) Response Generation, which produces text outputs based on data sources considered to be relevant and reliable. In addition, the system can (i) generate questionnaires based on a user query, (ii) allow users to upload their own documents, and (iii) work with a variety of Large Language Model (LLM) engines. This paper describes how the system can be used to build an expertise-driven application through a running example, explains the system's architecture, discusses related work, and evaluates CloudNerd on a 120-question Stack Overflow benchmark against OpenDeepResearch and AutoGPT using semantic textual similarity and RAGAS metrics with paired permutation tests. The results show that the RAG frameworks of CustomNerd and OpenDeepResearch both achieve better semantic fidelity than the agentic AutoGPT. Further, CustomNerd and OpenDeepResearch show complementary semantic advantages. Further, simply concatenating the two answers yields semantic benefits compared to the outputs of either one alone.

Keywords: modular AI; domain-specific expertise; knowledge modules; information retrieval



Academic Editor: Arkaitz Zubiaga

Received: 26 January 2026

Revised: 19 June 2026

Accepted: 22 June 2026

Published:

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article

distributed under the terms and

conditions of the [Creative Commons](#)

[Attribution \(CC BY\)](#) license.

1. Introduction

Creating hallucination-free and reliable expertise-driven systems requires high-quality data. For that reason, research and practice have followed the paradigm of grounding

generation in trustworthy external knowledge sources—such as peer-reviewed literature or community-governed technical repositories—as exemplified by retrieval-augmented generation systems [1–3].

However, data is not everything. Other important features of a high-quality, expertise-driven system include (i) determining the relevance of a source of information to a query, (ii) grading relevant sources based on their quality, and (iii) synthesizing the relevant and high-quality articles. In addition, a feature that is sometimes desirable, especially for novice querying users, is the ability to elicit information from the user in addition to the query. CUSTOMNERD provides a framework for all this with empirically evaluated CloudNerd results on a 120-question Stack Overflow benchmark (Section 8).

Existing frameworks for building LLM applications, such as LangChain [4] and LlamaIndex [5], require code-level changes to retrieval logic, prompt definitions, and deployment configuration when applying them to new domains. This entails a software development burden that CUSTOMNERD eliminates.

Furthermore, these frameworks provide no built-in mechanisms for (i) filtering out invalid or out-of-scope questions before retrieval begins, (ii) relevance classification that jointly considers topical fit, source quality, and domain-specific safety constraints, (iii) user-facing questionnaires that personalize query formulation based on individual context, or (iv) deterministic “saved state” packaging that bundles all configuration into a single reproducible deployment snapshot. CUSTOMNERD offers these functionalities.

The contributions of this paper are as follows:

- **Configuration-first framework architecture.** CUSTOMNERD enables domain experts with no programming background to deploy a fully functional expertise-driven RAG system in a matter of hours, entirely through a web interface, without modifying any core code. Domain-specific behavior—prompts, data sources, UI theming, and retrieval logic—is externalized into a declarative textual configuration.
- **A structured nine-stage pipeline with novel component combinations.** The pipeline integrates steps comprising question validity filtering (Section 4.2.3), relevance classification of articles, an analysis of source credibility, domain-specific safety constraints (Section 4.2.6), and an optional questionnaire-driven personalization layer that refines query formulation based on user-provided context (Section 3).
- **Reproducible saved-state deployment.** A saved-state mechanism bundles all configuration artifacts—prompts, environment variables, user interface settings, and retrieval code—into a single versioned snapshot that guarantees deterministic restoration across deployments and environments.
- **Empirical evaluation of CUSTOMNERD against state-of-the-art systems.** On a 120-question Stack Overflow benchmark with strict temporal constraints and RAGAS metrics, we compared CloudNerd (an instantiation of CUSTOMNERD for cloud-computing questions), OpenDeepResearch, and AutoGPT using RAGAS and STS scores. Three-way paired permutation tests on the same 120 question IDs show that CloudNerd leads on STS and answer similarity versus OpenDeepResearch, OpenDeepResearch leads on faithfulness and answer relevancy, and AutoGPT ranks last on all four metrics (Section 8, Table 6 ff). Interestingly, combining CloudNerd and OpenDeepResearch outputs improves over either system alone.

To demonstrate that the framework generalizes across domains, the paper also presents four working deployments—DietNerd, NewsNerd, CloudNerd, and SpaceNerd—as illustrative applications of the framework.

The paper is organized as follows. Section 2 reviews related work. Sections 3 and 7 serve as illustrative walkthroughs of setup and customization. Sections 4–8 present the core

research methods and evaluation. Section 9 reports a concat diagnostic on complementary CloudNerd and OpenDeepResearch outputs. Section 10 concludes.

2. Related Work

Four streams of prior work are relevant to CUSTOMNERD: foundational retrieval-augmented generation, general-purpose LLM orchestration frameworks, autonomous multi-agent research systems, and domain-specific expert QA systems. Each stream addresses part of the problem, but none provides the combination of zero-code domain adaptation, support for structured evidence quality control, and reproducible deployment that CUSTOMNERD provides.

2.1. Grounded and Retrieval-Augmented Generation

Ensuring reliability and reducing hallucinations in large language model outputs has motivated a broad line of work on grounding generation in external, trustworthy knowledge sources. Retrieval-augmented generation (RAG) systems integrate neural generation with information retrieval over curated corpora, allowing models to condition their outputs on retrieved documents rather than relying solely on parametric knowledge [1]. Subsequent work has explored scalable retrieval mechanisms and dense retrieval models that jointly learn retrieval and generation, demonstrating improved factual accuracy and robustness across downstream tasks [2,3]. These approaches exemplify a broader paradigm in which high-quality, community- or expert-governed data sources serve as anchors for reliable generation.

More recently, this paradigm has advanced to multi-agent collaborative reasoning for retrieval-augmented generation to handle long-context queries and multi-step reasoning tasks. MA-RAG, for example, defines a planner, a step definer, an extractor, and a QA agent within its pipeline and reports improved performance over previous RAG systems [6].

RAG-Critic [7] introduces a critic-guided agentic workflow that uses a data-driven hierarchical error system with more than 4000 fine-grained labels for error feedback. It uses a planning agent to dynamically orchestrate action functions and generate executable solution programs for automated error-driven self-correction.

2.2. General-Purpose LLM Orchestration Frameworks

RAG frameworks integrate retrieval with generation to ground outputs in external knowledge. Tooling ecosystems such as LangChain [4] and LlamaIndex [5] emphasize composable chains and vector-store integrations. CUSTOMNERD complements these by providing a fixed, end-to-end, audit-friendly pipeline with saved-state portability and SSE-first real-time UX. In 2025, LlamaIndex introduced Agentic Document Workflows (ADWs), which switch between retrieval strategies through a multistep workflow [8]. This development reflects a shift in LLM tooling from static retrieval pipelines toward workflow-based retrieval systems, in which document selection, retrieval strategy choice, and synthesis can be controlled through explicit agentic steps. However, these workflow abstractions still assume that the developer defines retrieval logic, prompt structure, and deployment behavior in code. CustomNerd differs by exposing these decisions as saved-state configuration, so that a domain owner can instantiate or revise a domain-specific assistant without modifying the underlying framework.

Both frameworks expose composable abstractions—retrievers, prompt templates, memory stores, and agent executors—that can be assembled into arbitrary workflows. However, adapting either framework to a new domain requires writing Python code to configure retrievers, redefine prompt logic, connect vector stores, and manage deploy-

ment infrastructure, placing the engineering burden on software developers rather than domain experts.

Neither framework provides the following: (i) a web-based administration console that allows non-programmers to reconfigure the full system without touching source code; (ii) LLM-based question validity gating as a built-in first-class pipeline stage; (iii) multi-signal relevance classification that jointly evaluates topical fit and domain-specific safety constraints (e.g., filtering animal-only studies from medical results, or filtering non-cloud content from technical results); (iv) questionnaire-driven query personalization before retrieval begins; or (v) a saved-state packaging mechanism that captures all prompts, retrieval code, UI configuration, and environment variables into a single, version-controlled, redeployable snapshot.

To mitigate these rigid pipeline limitations, emerging tooling ecosystems have introduced dynamic structural extensions. A notable early paradigm is LlamaIndex's multi-document architecture [9], which establishes a decentralized hierarchy of specialized sub-agents routed by a high-level meta-agent. This abstraction has been adapted in 2025 for domain-constrained workflows; for example, ARGUS [10] implements a hybrid Government Question-Answering (GQA) architecture on top of LlamaIndex, combining prompt-driven automated knowledge graph construction with explicit graph-path guidance and chain-of-thought style fine-tuning. However, these extensions still require developers to implement low-level retrieval routing and state-machine behavior in code.

2.3. Domain-Specific Expert QA Systems

A fourth line of work builds RAG systems tailored to specific domains—medical, legal, or scientific—by curating domain corpora and fine-tuning retrieval or generation for domain-specific quality standards. The DietNerd system [11], a direct predecessor of CUSTOMNERD, demonstrated that grounding LLM-generated nutrition answers in PubMed-retrieved, peer-reviewed evidence substantially reduces hallucinations compared with ungrounded generation. However, DietNerd was a single-domain system requiring code changes to deploy in a new field. Recent domain-specific systems, such as self-correcting agentic graph RAG models for clinical decision support in hepatology [12] and adaptive iterative self-query retrieval systems [13], further demonstrate the importance of specialized architectures for high-stakes fields.

CUSTOMNERD generalizes this approach into a multi-domain **generator**: the same pipeline serves medical (DietNerd), news (NewsNerd), cloud technology (CloudNerd), and space science (SpaceNerd) through configuration alone, with all domain-specific behavior—prompts, retrieval code, UI branding, and environment settings—editable through a web interface and exportable as a named saved state. This generality distinguishes CUSTOMNERD from both single-domain expert systems and from general-purpose frameworks that provide no domain-quality defaults.

Domain-specific RAG remains an active research direction across multiple high-stakes domains. In law, LexRAG [14] introduces a benchmark and toolkit for retrieval-augmented generation in multi-turn legal consultation, emphasizing the need to ground generated responses in legal articles and the consultation context. In medicine, MedRAG [15] combines retrieved medical evidence with generated background documents to compensate for incomplete or noisy retrieval in medical question answering. More generally, DSRAG proposes a domain-specific RAG framework based on document-derived multimodal knowledge graphs, combining structured subgraph retrieval with vector retrieval for specialized question answering [16]. These systems reinforce the importance of domain-tailored corpora, retrieval strategies, and evaluation criteria, but they remain specialized systems or benchmarks. CustomNerd instead provides a reusable configuration-first frame-

work for instantiating such domain-specific assistants across domains without rewriting the core system.

2.4. Autonomous Multi-Agent Research Systems

AutoGPT [17] and CrewAI [18] represent a distinct paradigm: autonomous agents that iteratively decompose open-ended tasks, issue web searches or tool calls, and synthesize reports with minimal human guidance. LangChain's Open Deep Research—built on LangGraph's supervisor–researcher–writer–critic architecture—is a concrete instance of this pattern [4,19]. In 2025, researchers proposed systems with more complex behaviors; for instance, Puppeteer switches between agents during reasoning and prunes low-contribution agents [20].

Recent 2025 work has also begun to treat multi-agent orchestration as a research problem in its own right. Zhang et al. propose OSC [21], a cognitive orchestration framework for multi-agent LLM collaboration that introduces Collaborator Knowledge Models to let agents dynamically estimate collaborator knowledge states and adjust communication behavior during reasoning. This line of work shows that multi-agent systems are moving beyond fixed agent teams toward adaptive coordination policies.

These systems excel at exploratory, open-domain tasks. However, they are not designed for domain-constrained, hallucination-minimizing expertise systems: they retrieve from the open web rather than curated, domain-appropriate databases; they apply no quality filters on source credibility or regulatory compliance; and they offer no mechanism for a domain owner to lock in specific data sources, safety rules, or prompt behavior without modifying code.

CUSTOMNERD combines RAG with a structured quality-control pipeline: retrieved documents are filtered by an LLM-based relevance classifier before synthesis, and only evidence meeting domain-specific safety criteria proceeds to the final response. Standard ranking (BM25), diversification (MMR), an agentic component, and neural reranking are compatible with this design and can be introduced as drop-in components via the `user_search_apis.py` interface. This paper, however, focuses on the RAG-only configuration of CUSTOMNERD, because we think that configuration is the most controllable and therefore the safest method for expertise-based systems.

We compare OpenDeepResearch and AutoGPT against the RAG configuration of CloudNerd on a 120-question benchmark (Section 8.3; Tables 9–12), with development-set comparisons in Section 8.5.2 and Appendix M.

Critical Gap Analysis

Table 1 maps each stream of prior work to its critical limitation and to the specific design decision in CUSTOMNERD that addresses it.

Taken together, these gaps motivate a new class of system—a *framework generator*—whose primary output is not answers but fully configured, production-ready domain applications. CUSTOMNERD is designed to occupy this position: it provides a structured, multi-stage evidence pipeline with built-in validity gating, multi-signal relevance classification, and questionnaire personalization, while simultaneously offering a zero-code configuration layer that allows domain experts to instantiate new expert systems in hours and export them as self-contained deployment snapshots.

Table 2 contrasts the configuration-first CUSTOMNERD design with LangGraph-style agentic workflows. Empirical scores for a representative LangGraph implementation (OpenDeepResearch) appear in Section 8; the table summarizes architectural differences that motivate the split win pattern across RAGAS metrics.

Table 1. Critical gaps in prior work and CUSTOMNERD’s targeted responses.

Prior Work	Critical Limitation	CUSTOMNERD’s Response
Foundational RAG [1–3,6,7,22]	<ul style="list-style-type: none"> Any retrieved document proceeds to synthesis regardless of domain safety, regulatory compliance, or source quality. No mechanism to reject out-of-scope queries before retrieval begins. No zero-code path for adapting the pipeline to a new domain. 	<ul style="list-style-type: none"> LLM-based question validity gate rejects out-of-scope queries before any retrieval occurs. Multi-signal relevance classifier enforces domain safety constraints on every retrieved document. Web-based configuration enables domain adaptation without modifying source code.
General-Purpose Frameworks [4,5,8,9]	<ul style="list-style-type: none"> Adapting to a new domain requires writing retrieval code, re-defining prompt logic, and managing deployment infrastructure. No built-in administration console for non-programmer domain experts. No saved-state portability for deterministic deployment across environments. No SSE-based real-time pipeline feedback to end users. 	<ul style="list-style-type: none"> Zero-code web interface exposes all configuration (prompts, retrieval, UI, environment) without touching source code. Saved-state export bundles all configuration into a single redeployable snapshot. SSE streaming communicates pipeline stage progress to the user in real time.
Autonomous Multi-Agent Systems [17–21]	<ul style="list-style-type: none"> Retrieve from the open web with no quality filters on source credibility, domain appropriateness, or regulatory compliance. No mechanism for a domain owner to lock in curated data sources or safety rules without modifying code. Not designed for expert-governed, hallucination-minimizing applications. 	<ul style="list-style-type: none"> Domain-specific data sources (PubMed, Stack Overflow, arXiv, news APIs) are configured explicitly, not inferred at runtime. Relevance classifier applies domain safety constraints (e.g., exclude animal-only studies in medical domains). All safety rules and data sources are locked in via saved state.
Domain-Specific Expert QA [10–16]	<ul style="list-style-type: none"> Achieve high answer quality within one domain but require code-level changes to deploy in any other field. Domain-specific behavior is hard-coded rather than configurable. Cannot be repurposed by non-programmers or domain experts without developer involvement. 	<ul style="list-style-type: none"> Any domain is instantiated through configuration alone—prompts, retrieval code, UI branding, and environment variables—without code changes. All domain behavior is exportable as a named saved state and loadable on any CUSTOMNERD instance by non-programmers.

Table 2. Design comparison: CUSTOMNERD versus LangGraph-style agentic workflows (OpenDeep-Research in Section 8).

Dimension	CUSTOMNERD	LangGraph Agentic Workflow
Architecture	Configuration-first, domain-specialized nine-stage pipeline	General-purpose supervisor–researcher–writer–critic graph
Control model	Deterministic saved-state bundle	Dynamic graph traversal; critic may trigger revision loops
Early filtering	Relevance classifier before synthesis	Full retrieved set passed to writer node
Concurrency	Parallel relevance scoring and batch summarization	Mostly sequential node execution
LLM calls per query	Bounded, versioned prompts	Variable (extra rounds when critic rejects)

Table 2. Cont.

Dimension	CUSTOMNERD	LangGraph Agentic Workflow
Prompt growth	Fixed, versioned prompts per stage	Grows with retrieved context passed to writer
Token efficiency	Lower (pre-synthesis pruning)	Higher (full retrieval set in synthesis)
Cost predictability	High (fixed pipeline)	Moderate (depends on critic revision loops)
Evidence access	Curated JSONL + Stack Overflow API; temporal cutoff enforced	Same curated JSONL + Stack Overflow API; same temporal cutoff (Appendix M)
Deployment	Web UI + saved state; no graph coding	Requires graph construction and node orchestration in code
Primary optimization	Bounded cost, scorable rate, domain safety gates	Flexible multi-step reasoning and grounding depth

3. CUSTOMNERD in Action

To show how one can set up a new application, we first show how to install the base system, and then we show how to customize the system to a new application.

3.1. Setup

The CUSTOMNERD code is available at <https://github.com/Harsh23Kashyap/Custom-Nerd>. CloudNerd evaluation artifacts, saved-state bundles, and run logs referenced in Section 8 and Appendix R are documented in that repository. Full installation instructions for Windows (WSL), macOS, and Linux are provided in Appendix A.

Switching Between Nerd Variants

Once the site is running, a new NerdEngine Developer (hereafter D) can either start from a blank configuration or reuse one of our preconfigured NerdEngine variants. Click the **Configurations** button in the navigation bar and select **Load Saved States**. This opens a panel listing the saved states CUSTOMNERD ships by default: **DietNerd**, **SpaceNerd**, **CloudNerd**, **SciNerd**, and **NewsNerd** (visual examples of each variant's home page are shown in Appendix B.3). D can load any of these as-is, or treat them as templates that D customizes and saves under a new name. After choosing a state, D can click **Load**, refresh the page, and then return to the **Home** tab—the selected Nerd will now be the default behavior for the site. Appendix B.4 provides a detailed visual walkthrough of the saved state management interface.

Note that D must provide the API keys, when necessary, for each application's data sources (e.g., OpenAI, Gemini, or Claude keys for model access, PubMed or news APIs where applicable) in the environment configuration (see Appendix B.7). **Ollama**, requires no API key and runs entirely on a local machine. Selecting LLM=Ollama instead triggers an automated local setup workflow (see Section 6).

3.2. Customization—An Example

To illustrate how CUSTOMNERD can be deployed on a new application through configuration alone, we present a detailed walkthrough of **CloudNerd**, a specialized assistant for cloud infrastructure troubleshooting. This case study demonstrates the complete configuration process, from initial requirements to production deployment, achieved entirely through the web interface *without code modifications to the core system*. A comprehensive visual guide to the web interface is provided in Appendix B.

3.2.1. Domain Requirements and Design Goals

The CloudNerd example application of CUSTOMNERD aims to serve as a technical troubleshooting assistant for cloud engineers and DevOps practitioners. The target domain spans cloud computing platforms (AWS, Azure, GCP, Oracle Cloud, DigitalOcean), containerization technologies (Docker, Kubernetes), infrastructure-as-code tools (Terraform, Pulumi, ARM templates, CloudFormation), serverless computing (Lambda, Azure Functions, Cloud Run), cloud networking (VPC, load balancers, CDN, DNS), and security practices (IAM, RBAC, compliance, encryption).

CloudNerd takes its expertise primarily from community-validated knowledge on Stack Overflow, where accepted answers, high vote counts, and author reputation serve as quality signals. The design prioritizes practical, reproducible solutions with code snippets and configuration examples over theoretical discussions. The extended saved state `saved_states/CloudNerd Advanced/` augments this with **additional cloud-native sources**: the AWS official blog RSS feed (plus optional AWS site search and topic-scoped resource pages for services such as Lambda and container platforms), the Microsoft Azure blog RSS feed together with HTML search-result link harvesting, and the CloudFest industry conference RSS feed. Candidate URLs are deduplicated, non-article listing hubs are skipped, and article pages are fetched in parallel; visible text is normalized into the same record shape used for Stack Overflow answers so downstream relevance classification, extraction, and synthesis need no structural changes. Shared `helper_functions.py` logic (enabled by default, configurable via `RATE_LIMIT_CURL_RESILIENCE`) can re-fetch rate-limited HTTP URLs with `curl`; the Advanced `user_search_apis.py` adds in-module retries, `Retry-After` handling, API backoff compliance, and defensive JSON parsing for Stack Exchange responses.

3.2.2. Frontend Configuration: Visual Identity

The visual identity (site name, icon, colors, disclaimer) was set via the Frontend Configuration tab; the full CloudNerd branding settings are provided in Appendix I.2.

3.2.3. Backend Configuration: Comprehensive Prompt Engineering

All five prompt categories were customized for technical troubleshooting through the Backend Configuration tab (see Appendix B.6). Table 3 outlines the prompt categories and their purposes.

Table 3. AI Prompt Configuration Categories

Prompt Category	Purpose and Configuration Options
Question Validation	Classify whether questions fall within the domain's scope and capabilities. Filters out inappropriate queries before processing begins.
Query Generation	Transform user questions into optimized search queries. Options: simple query generation, query expansion (multiple variations), contention-based queries for academic domains.
Relevance Classification	Evaluate articles for topical fit (relevance to question) and safety value (domain appropriateness). Ensures that only pertinent and high-quality content proceeds to synthesis.
Article Processing	Distinguish source types (questions vs. answers, studies vs. reviews). Extract structured information: problem context, solutions, risks, benefits, tools, metrics. Generate type-appropriate summaries.
Response Synthesis	Compose final answers with proper citations, domain-specific disclaimers, and domain-appropriate structure. Governs tone, formality, and communication style.

The detailed configuration for each prompt category, including domain topic lists, rejection rules, example classifications, and extraction fields, is provided in Appendix I.2.

3.2.4. Environment Configuration: API Keys and Provider Selection

CloudNerd uses OpenAI as the LLM provider (LLM="OpenAI") and requires OPENAI_API_KEY plus an optional STACK_API_KEY that raises the Stack Exchange rate limit from 300 to 10,000 requests per day; see Appendix B.7. The Advanced retrieval bundle does not require separate API keys for the vendor blog RSS or CloudFest feed (public HTTP/RSS); optional STACK_API_KEY remains important because Stack Overflow evidence is still retrieved first and capped by answer_counter, while cloud feed breadth is tuned by cloud_links_per_source and an internal cap on unique cloud URLs.

3.2.5. User Flow Configuration: Retrieval Implementation

The User Flow Configuration tab (see Appendix B.8) provides toggle controls for enabling various search features and interaction capabilities. Each option includes an inline SVG diagram (from user-flow-diagrams.js) comparing toggle-off versus toggle-on behavior on the question page or in the retrieval pipeline.

When **Deep Search** is enabled, article collection is routed through tiered_collect_articles() in helper_functions.py rather than a single collect_articles() call. The active saved state selects a **retrieval profile** (RETRIEVAL_NERD_PROFILES); CloudNerd and DietNerd register mode: "cascade" profiles, while other shipped Nerds default to legacy single-pass retrieval. The user-flow toggle (cascade_retrieval.visible in user_env.js) must be true for cascade mode to activate—otherwise the pipeline stays on legacy search even when a cascade profile is loaded.

Tiered retrieval runs up to five **planes** in order, merging and deduplicating results until quality gates pass or all enabled planes complete: **P1** precision queries from clean_query(); **P2** relaxed title/keyword variants; **P3** domain-facet queries (CloudNerd lexicon or DietNerd [Humans] suffix); **P4** broad LLM-expanded queries; **P5** gap-fill for missing key terms. On Stack Overflow deployments, P1 maps to strict search, P2 to title-only search, and P4 to extended-keyword expansion; P3 adds a domain facet before broad expansion. The Deep Search diagram shows a simplified three-step ladder (Strict → Title → Extended); plane budgets and short-circuit behavior are code-defined, not set in variables.env (Appendix J).

The retrieval layer in CloudNerd Advanced combines (i) a **two-stage Stack Exchange** pipeline optimized for Stack Overflow's data model—search for candidate questions per expanded query, then fetch top-voted answers with the withbody filter—with (ii) a **parallel cloud RSS and page-fetch** stage that queries AWS, Azure, and CloudFest feeds (and related search or resource-hub links where configured), deduplicates URLs, and extracts article-visible text into Stack-compatible records. Stack results are returned first, followed by up to a capped number of cloud-sourced items derived from all expanded queries.

The number of Stack Overflow answers is configurable via answer_counter; cloud breadth per provider uses cloud_links_per_source, with an additional cap on total unique cloud URLs merged into the batch.

The complete retrieval implementation, including RSS matching with query-token expansion, API interaction, deduplication, rate limiting and resilience, HTML normalization, and metadata extraction, is provided in Appendix I.3.

3.2.6. User Interface Configuration for Search Strategies on CloudNerd

PDF upload and identifier-based search were disabled; Stack Exchange article search was enabled by default with references, chat history, query cleaning, and Deep Search (cascade_retrieval.visible) active (see Appendix I.2).

3.2.7. Query Refinement Pipeline

Query expansion outputs are post-processed by `clean_query(query_list, max_queries=None)` in `clean_query.py`, which parses JSON `expanded_queries` (or plain nested lists) into search strings suitable for the Stack Exchange API.

The preprocessing function used to parse and sanitize expanded queries is provided in Appendix I.4.

3.2.8. Saved State: Complete Configuration Bundle

After testing, the complete configuration was exported as a named saved state. The repository includes a baseline `saved_states/CloudNerd/ bundle` and an extended `saved_states/CloudNerd Advanced/ bundle` (multi-source retrieval as above); each bundles prompts, UI settings, retrieval code, environment variables, and logo. The full file listing for the CloudNerd configuration walkthrough is provided in Appendix I.2; loading a chosen state on any `CUSTOMNERD` instance deterministically restores that CloudNerd deployment variant.

The evaluated CloudNerd configuration used in Section 8 replaces advanced multi-source RSS retrieval with multi-plane tiered Stack Overflow retrieval (`tiered_collect_articles`: planes P1–P5; benchmark runs map P1 to strict search, P2 to title-only, and P4 to extended-keyword expansion), activated by the CloudNerd code profile plus the User Flow `Deep Search` toggle, post-matcher TF-IDF citation recovery, supplemental gap-fill when coverage is thin, and an evidence-constrained synthesis prompt (Appendix J).

3.2.9. CloudNerd Question-Answering

To demonstrate CloudNerd in action, consider how the system processes the question “How to secure a Kubernetes cluster?”:

Step 1: Question Validation

- Input: “How to secure a Kubernetes cluster?”
- Prompt: `DETERMINE_QUESTION_VALIDITY_PROMPT` (Appendix C.1.1.6);
- Output: “True” (Kubernetes is a recognized cloud technology);
- Action: Proceed to query generation.

Step 2: Query Expansion

- Input: “How to secure a Kubernetes cluster?”
- Prompt: `GENERAL_QUERY_PROMPT` (Appendix C.1.1.10);
- Output: Twelve JSON-formatted variations covering RBAC, network policies, pod security, TLS certificates, API server security, authentication, authorization, secrets management, and monitoring;
- Action: Pass to query refinement.

Step 3: Query Refinement

- Input: JSON object with `expanded_queries` array;
- Function: `clean_query(query_list, max_queries=None)` in `clean_query.py`;
- Output: List of 12 plain-text queries (feeds plane P1 when Deep Search is on);
- Action: Execute retrieval—CloudNerd Advanced uses Stack Overflow plus cloud RSS/-page fetch per query; the 120-question benchmark uses tiered Stack Overflow retrieval instead (Appendix J).

Step 4: Article Retrieval

- Input: Twelve query variations (and, when Deep Search is enabled, additional plane P2–P5 query batches);
- Function: `collect_articles` in `user_search_apis.py`, orchestrated by `tiered_collect_articles()` when `cascade_retrieval.visible` is true;

- Process (advanced saved state): For each query, search Stack Overflow for up to 10 questions, fetch up to five answers per question (votes-sorted), deduplicate by question ID, clean HTML while preserving code blocks; then gather AWS blog RSS, Azure blog RSS (and blog search links where applicable), and CloudFest RSS hits with token-based query expansion, merge and deduplicate URLs, skip search-hub pages, and parallel-fetch article HTML into plain text capped for token budget;
- Process (benchmark eval): Planes P1–P2 run strict and title-only Stack Overflow searches; plane P3 adds domain-facet queries; planes P4–P5 add broad expansion and term gap-fill when quality gates do not short-circuit;
- Output: Up to `answer_counter` Stack Overflow-shaped records, followed in the Advanced bundle by a capped set of cloud-sourced records (same schema: title, URLs, body text) for unified downstream processing.

Step 5: Relevance Classification

- Input: Merged evidence batch (Stack Overflow answers plus optional vendor or industry feed excerpts);
- Prompt: `RELEVANCE_CLASSIFIER_PROMPT` (Appendix C.1.1.14; executed in parallel)
- Output: Binary classifications (yes/no) for each item;
- Action: Filter to retain only “yes” items (typically dozens remain depending on caps).

Step 6: Information Extraction

- Input: Relevant evidence items (Q&A or blog-style bodies);
- Prompts: `ARTICLE_TYPE_PROMPT` (classify as question/answer), `ABSTRACT_EXTRACTION_PROMPT` (extract 13 structured fields);
- Output: Structured summaries, including problem context, proposed solutions, accepted fixes, risks, benefits, cloud services, tools, configurations, performance metrics, errors, security considerations, alternatives, and community signals (where applicable);
- Action: Pass structured data to synthesis.

Step 7: Answer Synthesis

- Input: Structured summaries from the retained evidence set;
- Prompt: `FINAL_RESPONSE_PROMPT` (Appendix C.1.2.13);
- Process: Select 8–20 highest-quality items (prioritizing accepted Stack Overflow answers and high scores where present), organize evidence into thematic sections (e.g., Access Control and Identity Management, Network Policies and Encryption, Audit and Monitoring), explain RBAC setup, network policy configuration, TLS encryption, and monitoring tools (Falco, kube-bench) when supported by sources, include security warnings about misconfiguration risks, format citations with URLs (Stack Overflow and vendor or feed pages as appropriate);
- Output: Comprehensive technical answer with inline citations.

The complete workflow executes in approximately 30–60 s, balancing thorough evidence collection with acceptable latency for technical users.

3.2.10. Configuration Time Investment

The entire CloudNerd configuration was completed in approximately 2.5 h using the web interface configuration tabs (see Appendix B):

- **Frontend (15 min):** Set branding elements, colors, disclaimer (see Appendix B.5);
- **Environment (5 min):** Add API keys, select OpenAI provider (see Appendix B.7);
- **User Flow (45 min):** Write `collect_articles` function, configure HTML cleaning, test Stack Exchange API integration (see Appendix B.8);

- **Prompts (80 min):** Customize all five prompts with cloud-specific rules, examples, and extraction fields (see Appendix B.6);
- **Testing (15 min):** Validate with sample questions (Kubernetes security, EC2 selection, database migration);
- **State Saving (2 min):** Export complete configuration bundle (see Appendix B.4.2).

This 2.5-h investment produced a production-ready cloud troubleshooting assistant without any modifications to CUSTOMNERD's core codebase, demonstrating the framework's configuration-first design philosophy and the power of prompt-driven domain specialization.

4. Materials and Methods

CUSTOMNERD is designed to be a generator for expertise-based retrieval-augmented generation (RAG) applications. Because CUSTOMNERD externalizes domain-specific behavior into metadata and prompt definitions, it can be tailored to support any application that for which there is a reliable expert literature. This paper illustrates this generalizability with four distinct examples: a medical research assistant, a news summarizer, a cloud-technology advisor, and a space-science explainer.

The CUSTOMNERD framework has the following three core features as goals:

1. **Consistency**—similar developer experience across domains.
2. **Adaptability**—reconfiguration without code changes.
3. **Domain Fidelity**—answers that reflect the evidence, tone, and regulatory needs of the domain.

These features enable new verticals to be deployed in hours rather than months.

4.1. Algorithm Overview

While the details of an application assistant depend on specific prompts and database accesses, the framework consists of the basic steps shown diagrammatically in Figure 1. Algorithm 1 shows the method in pseudo-code, along with references to the corresponding sections for a detailed description of those steps. The pipeline generalizes the DietNerd methodology [11] to multiple domains.

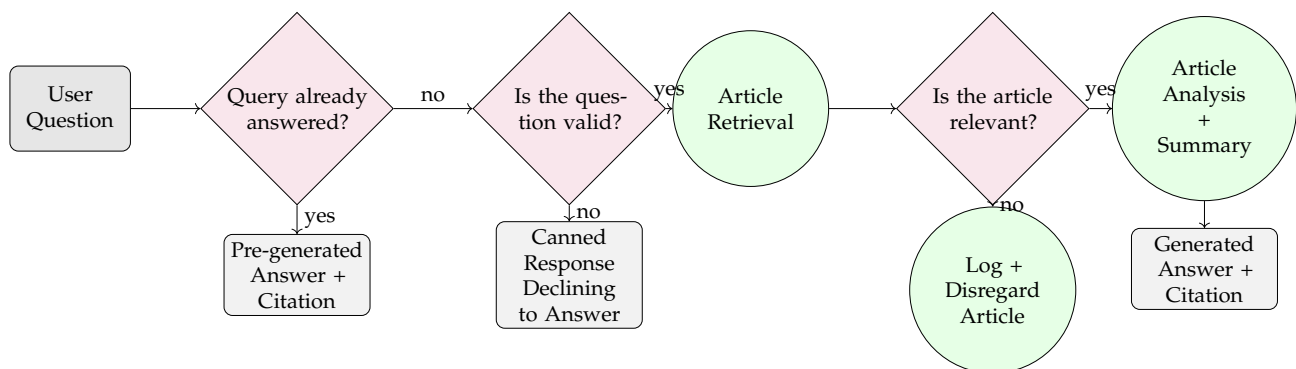


Figure 1. CUSTOMNERD Workflow Diagram: A workflow showing the dynamics of CUSTOMNERD starting with a user question. Purple diamonds are decision points. Green circles are actions. Red squares are outputs. This diagram is expressed in text in Algorithm 1.

4.2. System Design and Architecture

Chain-of-LLMs and Data Integration. CUSTOMNERD orchestrates multiple LLM prompts to validate questions, generate search queries, evaluate relevance, normalize records, and synthesize final answers with citations. The LLM backend—**OpenAI** (GPT series), **Gemini** (Google), **Claude** (Anthropic), or **Ollama** (local, open-weight models)—is

selected via a single LLM configuration variable, with prompts sourced from the prompt configuration file and helper utilities. Ollama enables fully offline, privacy-preserving operation with no API billing; the system automatically handles installation, model downloading, and server lifecycle management. Built-in Application Programmer Interfaces (APIs) include, for example, PubMed and publisher APIs for academic domains, news APIs for current events, Stack Overflow and documentation APIs for cloud, and arXiv/NASA for space. Other interfaces can be added by implementing the standard retrieval interface in the `user_search_apis.py` file. This module exposes a `collect_articles(query_list)` function that accepts a list of search queries and returns a list of standardized article dictionaries, allowing seamless integration of any new data source into the pipeline without modifying core logic.

Algorithm 1: CUSTOMNERD Algorithm Text: Start from a user question Q , as also illustrated in Figure 1. The condition on previous question–answer pairs requires database access. The de novo generation of an answer entails several prompts, as illustrated for our four application areas in Sections 4.2.3, 4.2.4, 4.2.6 and 4.2.9–4.2.11 and their corresponding appendices.

```

if user determines question  $Q$  is similar to some previously answered question  $Q'$  then
  User selects  $Q'$ 
  System returns the previously generated answer for  $Q'$ 
else
  if question  $Q$  is valid (Sections 4.2.3 and 4.2.4) then
    Parse the user question into keywords (Section 4.2.4)
    Perform article retrieval using configured providers
    for Each returned article  $A$  do
      if  $A$  is relevant to  $Q$  or includes safety considerations (Section 4.2.6) then
        Summarize the article in a few sentences
        Perform an article analysis and grade it (Sections 4.2.10 and 4.2.9)
      end
    end
    Only the strongest (highest grade) articles are used to answer the
    question (Section 4.2.11).
  end
end
end

```

Nucleus sampling and temperature settings. Our framework for configuring sampling parameters based on the task extends the methodology described in the DietNerd paper (Section 3.2, [11]) to support a variety of domain deployments. Our parameter selection strategy is informed by a careful consideration of the trade-offs between sampling diversity and output determinism; the strategy explicitly models how the interplay between temperature and `top-p` governs both coherence and creativity. For example, a low temperature in conjunction with a high `top-p` ($T \in [0.0, 0.3]$, $p \in [0.9, 1.0]$) produces highly deterministic outputs while maintaining lexical diversity [23,24]. In contrast, a high temperature with low `top-p` ($T > 0.7$, $p < 0.7$) can result in more unpredictable token sequences that may lack thematic consistency [25,26].

With this in mind, we propose the following parameter-setting guidelines based on domain requirements and task type:

- **High-precision tasks** (e.g., legal, medical, financial, structured extraction): $T \in [0.0, 0.1]$, $p \in [0.9, 1.0]$ to minimize hallucinations and maximize reproducibility [23,27].

- **Analytical tasks** (e.g., literature reviews, summarization, reasoning): $T \in [0.2, 0.4]$, $p \in [0.8, 1.0]$ to balance grounded reasoning with flexible phrasing [28,29].
- **Classification tasks** (e.g., validity, relevance, categorical mapping): $T \in [0.0, 0.2]$, $p \in [0.5, 0.9]$ to reduce output variance and improve inter-run reliability. [30]
- **Creative synthesis** (e.g., rewriting, narrative, marketing): $T \in [0.5, 0.8]$, $p \in [0.8, 1.0]$ to enable diverse word choice while preventing improbable token sampling [23,31].

These defaults reflect the entropy requirements of each task. Lower temperatures are used for deterministic classification and structured extraction tasks to reduce output variance, while higher temperatures are used in synthesis tasks to enable lexical diversity without sacrificing coherence through nucleus truncation. Using this information, our system implements the following defaults with GPT-4-turbo and Gemini-1.5-Pro. Question validity checks use low-entropy decoding ($T = 0.2$, $p = 0.7$) to ensure consistent binary judgments. Section mapping uses near-deterministic decoding ($T \in [0.0, 0.2]$, $p = 0.9$) because the task requires selecting the most probable structural mapping rather than generating novel text. Relevance judgments allow slightly higher stochasticity ($T \in [0.3, 0.5]$, $p = 0.8$) to capture alternative phrasing patterns that may signal semantic similarity. For CloudNerd evaluation, final synthesis uses low temperature ($T = 0.2$, $p = 1.0$, FINAL_RESPONSE_TEMPERATURE=0.2) to prioritize grounded, reproducible answers over lexical diversity. Other CUSTOMNERD domains may use higher T for creative synthesis per the task-type guidelines above; temperatures above 0.8 are not recommended except for specialized creative tasks due to the increased risk of semantic drift [32,33].

4.2.1. Similar Question Matching

CUSTOMNERD includes a question matching subsystem that identifies similar past questions from the conversation history, enabling users to obtain previously generated answers without regenerating responses. This feature reduces computational costs, improves response time, and enhances user experience by surfacing relevant historical answers.

When enabled, the system automatically maintains a searchable database of all questions and their corresponding answers. As users interact with the system, each question-answer pair is stored with metadata, including timestamps and session information. When a new question is entered, the system searches this database to find semantically similar past questions.

Hybrid Similarity Matching: The system employs multiple complementary methods to identify similar questions, combining four techniques to maximize accuracy:

1. **Phrase-level matching:** Identifies questions with similar word combinations and phrases, capturing semantic relationships even when wording differs slightly.
2. **Character-level matching:** Ignores typographical variations, word order differences, and morphological variations that might otherwise prevent matches.
3. **Keyword overlap analysis:** Finds questions with shared important terms, even when the overall structure differs.
4. **Sequence alignment:** Compares question structure and word order to identify questions that express the same intent through different phrasing.

The system combines these methods using a max-pooling strategy, selecting the highest similarity score across all approaches ($S_{final} = \max(S_{tfidf}, S_{fuzzy}, S_{jaccard}, S_{sequence})$). This approach prioritizes recall, ensuring that if a question matches strongly on *any* single dimension (e.g., high keyword overlap despite different word order, or high character similarity despite typos), it is flagged as a potential match, regardless of how users phrase their questions.

Intelligent Text Processing: Before matching, questions undergo normalization to handle natural language variations. The system recognizes synonyms (e.g., “pros” and

“advantages”, “cons” and “disadvantages”), standardizes terminology (e.g., “B12” and “vitamin B12”), and normalizes formatting differences. This is achieved through a lightweight Natural Language Processing (NLP) pipeline that applies tokenization, stemming (reducing words to their root form), and a domain-specific dictionary lookup to map variations to canonical terms. This preprocessing ensures that questions with equivalent meaning but different wording may be similar.

User Experience: When a user enters a question, the system automatically searches their entire personal conversation history, which is stored in a file named `historical_answer.json` on their device. The system displays similar past questions as clickable suggestions, so users can easily see if they have already asked the same or a related question before and directly access the previous answer if desired. Users can either select a previous question whose answer will be immediately viewable or request the system to generate a new response.

In addition, users can browse their interaction timeline in chronological order, independent of similarity matching, making it simple to review the sequence of previous questions and answers. Since this is each user’s own history file, there is no login mechanism required and no sharing between users; all questions and answers are private to the individual. At any time, a user may choose to delete their `historical_answer.json` file to erase their history.

Examples: The system works across diverse domains, recognizing semantic equivalence regardless of domain-specific terminology:

- **Medical domain:** As a consequence of synonym detection, suppose a user asks, “What are the pros and cons of vitamin B12 supplementation?” The system recognizes this as similar to a previous question, “What are the advantages and disadvantages of taking vitamin B12?” despite different wording (as noted above, “pros” is recognized as a synonym of “advantages”, and “cons” with “disadvantages”). Thus, the prior question immediately surfaces as a suggestion.
- **Cloud technology domain:** A user queries, “How do I secure a Kubernetes cluster?” The system matches this to a previous question, “What’s the best way to secure Kubernetes?” recognizing that both seek security best practices for Kubernetes, even with different phrasing. The existing answer with security recommendations and references is instantly available.
- **Space science domain:** A question, “What causes black holes to form?” is matched to “How are black holes created?” The system understands that “causes to form” and “created” express the same scientific inquiry, surfacing the previous detailed explanation with academic citations.

In each case, the system identifies semantic equivalence across different phrasings, allowing users to access complete answers with citations immediately, saving both time and computational resources.

4.2.2. Enhancing the Specificity of Responses Through Questionnaires

If the user asks CUSTOMNERD to generate an answer as opposed to returning a cached response, application developers may invoke a questionnaire feature to allow users to receive more targeted responses than they would without the questionnaire. This section shows how this works for a medical application.

The runtime interface to this module operates on a strict “query-in, questionnaire-out” execution pipeline. When a user submits a natural language symptom inquiry (e.g., “severe headache”), the system utilizes a lightweight `gemini-2.5-flash-lite` model to extract canonical clinical keywords, which are subsequently vectorized using Google’s `text-embedding-004`. The pipeline then computes the cosine similarity via `sklearn` to

map these embedded keywords against our precomputed Mayo Clinic data repository, expanding the single input into a cluster of semantically related differential conditions (e.g., tension headache, migraine, cluster headache, and sinus headache).

To aggregate and refine the final question set, the top-ranked condition provides the foundational base questions, while the overlapping features from the remaining related conditions form a collaborative question pool. The system normalizes heterogeneous question nomenclature (e.g., merging “smoking status” and “smoker”) and applies a max-pooling voting ensemble across the candidate pool, dynamically selecting a highly targeted matrix of 5 to 7 queries with the highest frequency of votes. Furthermore, strict demographic gates are enforced during this selection loop—for instance, automatically excluding neonatal jaundice questionnaires if the verified age profile is adult—effectively adjusting the downstream retrieval scope based on the user’s contextual variance.

For example, for health applications, we have built a database of diseases, making use of the public Mayo Clinic website, whose landing page is <https://www.mayoclinic.org/diseases-conditions>. Specifically, CUSTOMNERD uses the selenium library in Python to read the 1117 cataloged diseases in the “Diseases and Conditions” page, focusing on potential risk factors. CUSTOMNERD uses OpenAI’s GPT-4o model to convert the risk factors for each disease into a questionnaire of five questions. Our prompt—shown in Appendix K.1—asks the LLM to focus on the most common and most risky factors. The resulting questions are stored in a JSON file and presented to the user.

Given a user’s question, the most related symptom is selected through vector similarity. Embeddings for diseases in the database are created using Google’s models/text-embedding-004 model. Receiving an input from the user, CUSTOMNERD employs a gemini-2.5-flash-lite model to extract keywords from the input. These keywords are then embedded with Google’s models/text-embedding-004. We use the cosine similarity function in the sklearn library to choose the most related disease in the database, along with the pre-generated questions.

Sometimes a symptom can be sex- or age-restricted. For example, males do not suffer from ovarian cancer, and only infants will have neonatal jaundice. Neonatal jaundice questions will then be excluded, for example, if the age of interest is not very young.

This entire procedure enables a final questionnaire tailored to the user’s query. For example, in contrast to the questionnaire for the query “what are diets recommended if I have a headache”, the query “what are diets recommended if I’m light-headed” yields a questionnaire asking about age, birth sex, history of dizziness, family history of balance problems, and whether the querier takes certain medicines.

As an example of how different answers to the same questionnaire can affect the system’s answer, consider the question “Risk Factors for Miscarriage”. For User 1, who is 40–45 years old, smokes, and has long-term conditions, the analysis focuses on age and smoking. The advice consequently centers on the statistical risks of advanced maternal age and the known impacts of smoking. The “long-term conditions” are treated as a secondary but relevant factor. See Appendix K.2 for the detailed response.

In contrast, for User 2, who is under 35 with a past miscarriage and uterine problems, the analysis identifies “under 35” as a lower-risk category. Instead, the questionnaire focuses on “Past Miscarriages: yes” and “Uterine Or Cervical Problems: yes.” The advice focuses on managing physiological and historical medical risks, highlighting the significance of prior pregnancy outcomes and structural health issues. Please find this response also in Appendix K.2.

4.2.3. Question Validity

Before processing a question, CUSTOMNERD employs an AI-powered validation step to ensure the query falls within the domain's scope and capabilities. This gate filters out questions that are outside the system's intended purpose in order to avoid hallucinations because of insufficient information.

The validation system is domain-configurable, allowing each deployment to define what constitutes a valid question based on its specific focus. The goal is to reject invalid questions with clear explanations, redirecting users to appropriate resources when needed.

Examples across domains:

- **Medical domain:** Questions about meal planning or recipe requests are flagged as outside scope, as the system focuses on evidence-based nutritional research rather than meal preparation. Questions about animal nutrition are similarly filtered out, because the literature source PubMed focuses on human health.
- **Cloud technology domain:** Questions about general programming concepts unrelated to cloud infrastructure, or requests for specific code implementations, are identified as outside the domain's scope. The system focuses on cloud architecture, security, and best practices rather than general software development.
- **Space science domain:** Questions about astrology, science fiction scenarios, or speculative theories without a scientific basis are filtered out. The system maintains focus on peer-reviewed astronomical research and established scientific knowledge.
- **News domain:** Questions about historical events beyond recent news cycles, or requests for opinion pieces rather than factual reporting, are identified as outside the current events focus.

Complete prompt definitions for the domain-specific validation guards are provided in Appendix C.1

4.2.4. Query Generation for Safe Responses

After validation for applicability, the system transforms the user's natural language question into search queries suitable for domain-specific databases. The prompt requests a primary search query that captures the core intent of the question, plus optional alternative queries that explore different aspects or potential controversies. This multi-query approach captures both mainstream perspectives and alternative viewpoints when relevant.

Domain-specific prompt behavior. Each saved state defines its own generation strategy:

- **CloudNerd (Stack Overflow).** The 'GENERAL_QUERY_PROMPT' (Appendix C.1.1.10) requires 8–12 JSON-formatted variations under the key 'expanded_queries'. For example, a user question such as "How do I secure a Kubernetes cluster?" automatically expands into entries like:
 - "Kubernetes cluster security best practices";
 - "Kubernetes role-based access control setup";
 - "Kubernetes pod security policies";
 - "TLS certificates in Kubernetes cluster";
 - "How to monitor Kubernetes security".

These variations deliberately cover Role-Based Access Control (RBAC), network segmentation, certificate management, and observability so that the downstream Stack Overflow collector hits question threads about Identity and Access Management (IAM), NetworkPolicy YAML, Transport Layer Security (TLS) termination, Falco/kube-bench monitoring, etc., without the end-user needing to enumerate each vector.

- **SpaceNerd (NASA/Wikipedia).** Its 'GENERAL_QUERY_PROMPT' emits newline-separated keywords rather than JSON, because the NASA Image/Wikipedia APIs

favor plain keyword concatenation. For example, the request “Show me images of the Andromeda Galaxy” produces the following:

- “Andromeda Galaxy”;
- “Andromeda Galaxy photo”;
- “Andromeda Galaxy visualization”.

The first line ensures Wikipedia/ADS lookups, while the subsequent variants target NASA’s media endpoints for photography versus rendered visualizations.

These auto-generated lists are logged alongside the user prompt, enabling transparency (users can see exactly which variants were issued) and reproducibility (saved states guarantee the same set of expansions for the same question).

For domains requiring specialized terminology or query formatting, an optional query refinement step can perform preprocessing, e.g., normalizing technical terms, expanding abbreviations, and adapting queries to match the conventions of specific databases (e.g., medical terminology for PubMed, technical keywords for cloud documentation). This refinement improves retrieval accuracy, especially for domains with complex or domain-specific vocabularies.

4.2.5. Article Retrieval

The system executes parallel searches across one or more data sources configured for the domain. Each source is queried with the generated search terms, and results are collected with metadata, including titles, authors, publication dates, identifiers, and access information. The system automatically de-duplicates articles that come from multiple sources, ensuring that each unique article is processed only once while preserving all source attributions.

If users provide specific article identifiers (such as DOI, PMID, or arXiv IDs), the system retrieves those articles in addition to any other search results. This allows users to request analyses of particular papers or documents they already know about. Throughout the retrieval process, progress updates are streamed to the user interface, providing real-time feedback on how many articles have been collected from each source.

4.2.6. Relevance Classifier and Safety Considerations

Not all retrieved articles are relevant to the user’s question. The system employs an AI-powered relevance classifier that evaluates each article on two dimensions: topical fit (whether the content addresses the question) and safety value (whether the content is appropriate and reliable for the domain). Articles are scored and categorized into relevant and irrelevant groups, with domain-specific filters applied (e.g., excluding animal-only studies in human health domains, filtering out opinion pieces in evidence-based domains).

Relevance scoring uses a hybrid approach that combines multiple signals to balance precision (keeping results on-topic) and recall (ensuring important articles are included):

- **Topical fit:** Semantic similarity between the article and the user’s question.
- **Source quality:** Publication credibility, peer-review status, and publisher reputation.
- **Recency:** Freshness of the material for domains where timeliness matters (e.g., news, cloud).
- **Community signals:** For community-driven sources (e.g., Stack Overflow), upvotes, accepted answers, and author reputation.
- **Safety constraints:** Domain-specific guardrails evaluate articles to filter out unsafe content, as we can see in Section 4.2.7 below.

Articles are classified in parallel to save time. Then, the system filters them in multiple steps: an initial broad filter keeps articles that might be relevant, and later filters use

domain-specific rules (for example, removing duplicate news stories or skipping animal-only studies in medical topics). Only articles that meet both relevance and safety standards are kept and moved forward.

4.2.7. Evaluation Pipelines

Evaluation pipelines apply domain-specific filters and validators to ensure that only trustworthy, appropriate content proceeds to synthesis. These pipelines transform raw retrieval results into curated evidence that meets each domain's quality and safety standards.

Safety and Credibility Filters: Each domain applies filters tailored to its requirements:

- **Medical domains:** Enforce peer-review requirements, exclude animal-only studies when human evidence is needed, filter out recipe requests or non-research queries, and prioritize studies with appropriate sample sizes and statistical rigor.
- **News domains:** Filter out opinion-heavy sources, deduplicate articles covering the same event, prioritize multiple-source verification, and exclude clearly biased or unreliable publications.
- **Technical domains:** Highlight security risks and vulnerabilities, prioritize solutions with security best practices, flag deprecated or outdated approaches, and emphasize cost implications and scalability considerations.
- **Academic domains:** Ensure identifier integrity (e.g., arXiv IDs, DOI validation), verify peer-review status when required, and prioritize sources with complete metadata and proper attribution.

Regulatory Compliance: For domains operating in regulated fields (medicine, finance, law), evaluation pipelines include additional compliance checks such as:

- **Disclosure requirements:** Ensure funding sources and conflicts of interest are identified when available.
- **Evidence hierarchy:** Prioritize higher-quality evidence types (systematic reviews over individual studies, peer-reviewed over preprints).
- **Risk communication:** Flag content that requires specific disclaimers or warnings based on domain regulations.

4.2.8. Article Match Against Article Analysis Database

For the sake of efficiency, the system maintains a cache of previously processed articles. When an article has been analyzed in a previous session, the system reuses the existing summary and analysis rather than reprocessing it from scratch.

4.2.9. Text Parsing

CUSTOMNERD extracts and processes content in multiple formats. For open-access articles, the full text is retrieved, cleansed of formatting artifacts, and structured into key sections (abstract, introduction, methods, results, discussion). When full text is not available, the system works with abstracts and metadata, attempting to extract maximum value from available information.

Users can also upload documents directly (PDF formats), which are processed in the same way as retrieved articles. This capability allows users to include proprietary research, internal documents, or papers not available through public databases in their queries.

4.2.10. Article Summarization

The system distinguishes between different types of sources and applies type-appropriate analysis strategies based on the domain. For example, in medical domains, primary research studies are analyzed for methodology, sample size, and specific findings, while review articles are evaluated for completeness and synthesis quality. In technical do-

mains, questions are distinguished from answers, with accepted solutions prioritized over alternative approaches. In news domains, breaking news articles are distinguished from analysis pieces, with recency and source credibility factored into evaluation. In academic domains, peer-reviewed publications are distinguished from preprints, with publication venue and citation metrics considered. This type-aware processing ensures that the strength and nature of evidence are properly contextualized according to domain-specific standards.

For each article, the system extracts and structures key metadata (title, authors, publication source, publication date, identifiers) and generates a query-tailored summary that highlights aspects most relevant to the user's question. The goal of this targeted summarization is to preserve the most pertinent information from each source for the final synthesis, rather than generic summaries that may miss question-specific nuances.

4.2.11. Safety-Conscious Output Synthesis

The final answer is synthesized from the processed articles, with the prompts prioritizing high-quality evidence and explicitly highlighting risks, limitations, and areas of uncertainty. The synthesis process tries to balance comprehensiveness with accuracy, ensuring that strong claims are supported by robust evidence, while weaker or contested findings are appropriately qualified.

The final output includes inline citations that map back to source articles, using identifiers (such as DOI) when available, with fallback mechanisms to ensure all claims are properly attributed even when metadata is incomplete.

4.2.12. Advanced Features

The configuration system includes several advanced features that accelerate domain implementation and improve user experience:

- **AI-Assisted Code Generation:** The code generator accepts API documentation, code snippets, or natural language descriptions and automatically generates integration functions that match the system's required interfaces.
- **AI-Assisted Prompt Generation:** The prompt generator uses the current prompt as context to suggest improvements, helping domain experts refine their prompts more efficiently.
- **Real-Time Validation:** The interface provides immediate feedback on configuration inputs: Color inputs validate format, file uploads check size and type constraints, and API key inputs indicate required versus optional fields.
- **Adaptive Theming:** The system automatically detects whether the selected color scheme represents a light or dark theme and adapts interface elements accordingly.
- **Productivity Shortcuts:** Keyboard shortcuts enable rapid configuration workflows, allowing experienced users to save time when switching between configuration tabs.

5. Website Interface Architecture

The website architecture consists of a FastAPI backend and a frontend built with HTML, CSS, and JavaScript. The frontend listens to Server-Sent Events (SSEs) from the backend to display live progress and final responses with references. Figure 2 shows the overall data flow. A comprehensive visual walkthrough of the user-facing interface and configuration options is provided in Appendix B.

Component Responsibilities:

- **Frontend (HTML, CSS, JS):** Handles user input/output, streams live updates via SSE, and supports exporting results as PDF. The user-facing interface and configuration options are detailed in Appendix B.

- **Backend (FastAPI):** Orchestrates the RAG pipeline, interacts with AI models, and exposes REST and SSE endpoints.
- **Storage (JSON State/History):** Persists cached history, manages configuration bundles, and enables quick retrieval of session data. Saved state management is illustrated in Appendix B.4.

The complete API endpoint reference, background task orchestration, front-end implementation details, and endpoint-to-function mapping are provided in Appendix F.

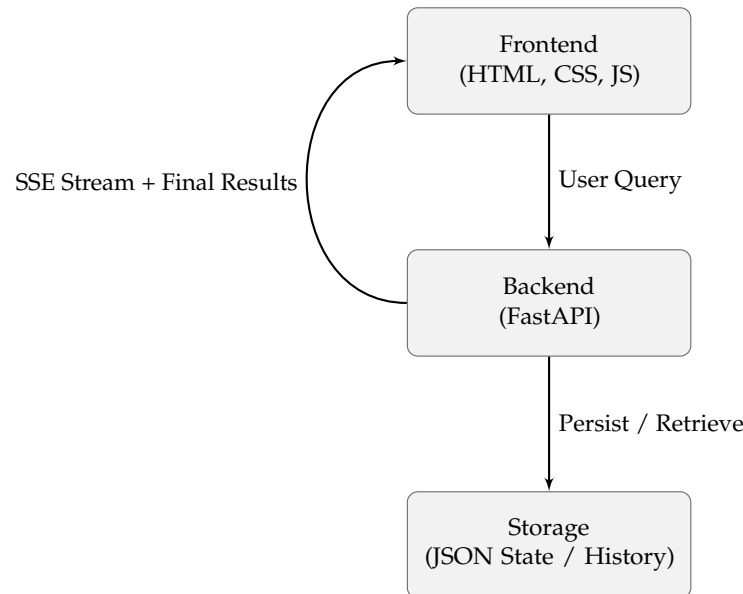


Figure 2. System architecture illustrating data flow between frontend, backend, and storage.

6. Deploying Alternative Large Language Models

CUSTOMNERD supports four LLM backends: **OpenAI**, **Gemini**, **Claude**, and **Ollama**. Switching providers requires only updating the LLM variable in `variables.env` and re-testing existing prompts, with migration guidance provided in Appendix G.4.

6.1. Cloud Providers (OpenAI, Gemini, Claude)

For cloud-hosted backends, the operator sets `LLM` to the desired provider name, supplies the corresponding API key (`OPENAI_API_KEY`, `GEMINI_API_KEY`, or `ANTHROPIC_API_KEY`), and optionally verifies connectivity using the **Test** button in the Environment Configuration tab, which performs a lightweight API call and reports success or failure inline. All subsequent pipeline stages—validity gating, query expansion, relevance classification, and response synthesis—proceed without further changes to prompts or retrieval logic.

6.2. Ollama: Zero-Cost Fully Local Inference

CUSTOMNERD supports **Ollama** (<https://ollama.com>), an open-source runtime that serves open-weight language models entirely on the operator's own hardware. This means the system can be run **completely locally**: No API key is required, there is no per-token billing, and no user query or document content is transmitted to any external server. This makes CUSTOMNERD viable in privacy-sensitive, offline, or resource-constrained deployments where subscription-based cloud inference is not feasible.

For example, models such as **LLaMA** (Large Language Model Meta AI) developed by Meta are supported **via Ollama** (<https://ai.meta.com/llama/>).

6.2.1. Why Local Inference Matters

Cloud LLM providers impose per-token costs that can accumulate rapidly for high-volume or long-context research queries. They also require data to leave the operator’s infrastructure, which may conflict with institutional data-governance policies. Ollama eliminates both constraints: inference executes on the local CPU or GPU, and the operator retains full custody of all inputs and outputs. On Apple Silicon hardware (M1 and later) and modern x86 workstations, 7–8 billion parameter models run at conversational speeds without specialist accelerators.

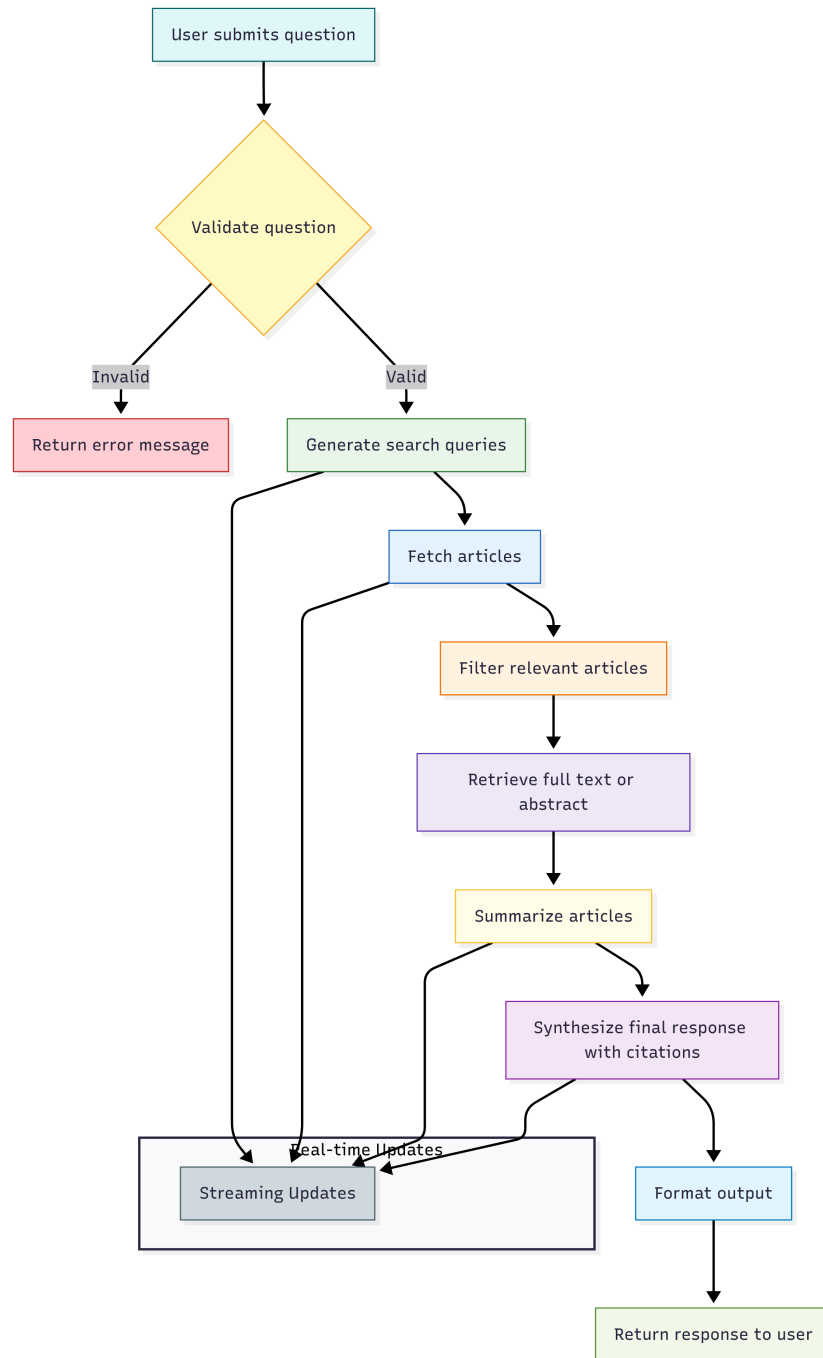


Figure 3. CUSTOMNERD workflow: User request triggers the multi-stage pipeline (query generation, article collection, organization, filtering, synthesis, citation mapping) with stages communicated via SSE to the frontend.

6.2.2. Pre-Flight Status Check

When the Ollama tab is opened in the web interface, the system immediately queries the local Ollama daemon (`GET/api/tags`) and displays three live status indicators: whether Ollama is installed, whether the server is running on port 11434, and whether the selected model has already been downloaded. This pre-check prevents redundant setup steps and gives the operator immediate situational awareness before triggering any installation.

6.2.3. Model Selection and the Model Guide

The operator selects a model from a categorised dropdown (Recommended, Small, High Quality, Coding, Cloud) or enters a custom model name directly. A built-in **Model Guide** pop-up assists in this choice: it presents a filterable table of curated models with per-model parameter count, estimated disk footprint, minimum RAM requirement, relative inference speed, and quality rating. The guide offers two view modes toggled from the header:

- **Basic mode** —shows a plain-language compatibility summary (e.g., “*MacBookM1 or later · Windows/Linux PC with 8 GB RAM*”), suitable for non-technical users selecting hardware-appropriate models.
- **Advanced mode** —reveals the full technical breakdown: parameter count, disk size, minimum RAM, speed badge, and quality badge.

If models are already installed on the local machine, they are detected automatically via the `/ollama_status` endpoint when the guide opens, and an **Installed** chip is displayed next to the relevant rows. Models found locally but absent from the curated list appear in a dedicated **Installed on your machine** section at the top of the table, and are simultaneously added to the model dropdown.

6.2.4. Automated Setup with Multi-Method Installation Fallback

When the configuration is saved, the backend `/ollama_setup` endpoint orchestrates four sequential steps, streaming real-time progress to the browser via Server-Sent Events so the operator can observe each stage without reloading the page:

1. **Installation (with fallback chain).** The system checks whether Ollama is present on the host. If not, it attempts installation using a prioritised sequence of methods tailored to the following detected operating system:
 - **macOS:** (1) official `curl` install script; (2) Homebrew (`brew install ollama`) if Homebrew is available; (3) direct binary download from the GitHub Releases page to `/usr/local/bin`.
 - **Linux:** (1) official `curl` install script, which also registers and starts a `systemd` service automatically; (2) architecture-aware binary download (`amd64` or `arm64`) to `~/local/bin`.
 - **Windows:** (1) official PowerShell install script (`irm https://ollama.com/install.ps1 | iex`); (2) `winget install Ollama.Ollama` if the Windows Package Manager is available.

Each method is attempted in order; the chain halts at the first success. If all methods fail, a `fatal` event is streamed to the browser with platform-specific manual instructions.

2. **Server startup.** The backend starts the `ollama serve` daemon as a background process and polls `/api/tags` for up to 15 s, waiting for readiness before proceeding.
3. **Model download.** `ollama pull <model>` is executed and its output is streamed line-by-line to the browser, giving the operator live download progress.

4. **Verification.** The system confirms that the selected model appears in `/api/tags` and emits a complete event. If the model is not immediately visible (rare on first install), a warning message prompts a backend restart.

6.2.5. Inference Routing

At query time, the existing OpenAI Python SDK is reused with its `base_url` redirected to `http://localhost:11434/v1`. All prompts—validity gating, query expansion, relevance classification, and final response synthesis—execute unchanged against the local model, with no modification to the pipeline. A `_safe_create()` wrapper intercepts `NotFoundError` (model not pulled) and `APIConnectionError` (server not running) and surfaces them as actionable user-facing messages rather than opaque 500 errors.

6.2.6. Hardware Requirements and Model Selection Guidance

Table 4 summarises the supported models with their resource profiles. Lightweight models (`llama3.2`, `phi3:mini`, `llama3.2:1b`) run on any modern laptop with 4–8 GB of RAM and are recommended for first-time deployments. Mid-range models (`llama3.1:8b`, `qwen3:8b`, `mistral`) require 8–16 GB RAM and deliver substantially higher response quality, suitable for Apple Silicon M1 Pro and above or x86 workstations. A cloud-routed model (`kimi-k2.5:cloud`) is also available for operators who prefer a hosted fallback without switching providers in the environment configuration.

Table 4. Ollama model profiles available in CUSTOMNERD. RAM figures indicate the minimum recommended system memory for comfortable operation.

Model	Params	Disk	Min RAM	Speed	Best Suited for
<code>llama3.2</code>	3.2B	~2 GB	4 GB	Fast	General use, first deployment
<code>llama3.1:8b</code>	8B	~4.7 GB	8 GB	Medium	Better reasoning, 128 K context
<code>llama3.2:1b</code>	1B	~1.3 GB	3 GB	Fastest	Very low-RAM devices
<code>phi3:mini</code>	3.8B	~2.3 GB	4 GB	Fast	Compact, punches above weight
<code>qwen3:8b</code>	8B	~5 GB	8 GB	Medium	Best reasoning at 8B class
<code>mistral</code>	7B	~4.1 GB	8 GB	Medium	Versatile general purpose
<code>code llama:7b</code>	7B	~3.8 GB	8 GB	Medium	Code generation tasks
<code>kimi-k2.5:cloud</code>	—	None	None	Cloud	Cloud-routed, no local download

The full model catalogue with Basic/Advanced hardware compatibility notes is available interactively through the Model Guide in the web interface, and a static reference is provided in Appendix G.4.

When the number of retrieved articles exceeds ten, CUSTOMNERD partitions them into parallel batches of ten—empirically, the maximum that Gemini 1.5 Pro processes without hallucinations—and synthesizes a bulleted summary for each batch independently. The batch summaries are then concatenated and deduplicated: section titles across summaries are compared using cosine similarity, and sections with a similarity score above 0.7 are merged, with their inline citations combined. A final LLM call condenses each merged section, preserving all source attributions while eliminating redundancy. The full synthesis and summarization prompts are provided in Appendix L.1 and Appendix L.2, respectively. For example, for the query, “What are the effects of the keto diet?”, Table A6 shows the concatenated result alongside the deduplicated output.

7. Example Use Case Applications of CUSTOMNERD

We demonstrate the flexibility of CUSTOMNERD by implementing four case studies: DietNerd, NewsNerd, CloudNerd, and SpaceNerd. Visual examples of each variant’s home page interface are shown in Appendix B.3. You can find the main prompts for these applications in Appendix C.1.

7.0.1. DietNerd (Medical/Nutrition Research)

DietNerd specializes in evidence-based medical and nutrition research, prioritizing peer-reviewed sources and human evidence. The system collects explicit health parameters through structured questionnaires, ensuring all clinical context is captured before retrieval begins.

7.0.2. NewsNerd (Current Events Research)

NewsNerd focuses on current events with an emphasis on speed and recency. The minimal questionnaire approach prioritizes user convenience, while the evaluation pipeline tests source credibility and performs de-duplication.

7.0.3. CloudNerd (Cloud Computing Research)

CloudNerd makes use of community-driven technical content from Stack Exchange, using query expansion to generate comprehensive search variations. The evaluation pipeline prioritizes accepted answers and solutions with reproducible configurations, while synthesis emphasizes security considerations and cost implications.

7.0.4. SpaceNerd (Astronomy/Space Research)

SpaceNerd aggregates content from academic sources (arXiv, NASA). The synthesis approach highlights areas of uncertainty and ongoing debates, reflecting the exploratory nature of space science.

Table 5 summarizes the design goals and trade-offs of these four applications.

The goals in Table 5 illustrate the usefulness of a tool that allows a design team to decide in application-specific tradeoffs, e.g., between precision and understandability. The configuration-first approach presents a consistent user interface layer to apply to these four applications seamlessly and thus will generalize to new applications.

Table 5. Domain-Specific Design Trade-offs.

Trade-Off Dimension	DietNerd	NewsNerd	CloudNerd	SpaceNerd
Latency vs. Reliability	Tolerates longer retrievals in favor of rigor and evidence quality	Must deliver within seconds; prioritizes speed	Balances speed with technical depth; community validation provides quick quality signals	Moderate latency acceptable for comprehensive academic coverage
Information Provenance vs. Understandability	Shows full references (AMA style); prioritizes provenance and rigor	Hides references by default; prioritizes understandability	Shows Stack Overflow links inline, favoring provenance	Provides both provenance (arXiv, ADS handles) with plain-language understandability
Tone and Formality	Formal scientific language; accessible to lay readers	Journalistic brevity; neutral tone	Technical, snippet-heavy; preserves code precision	Explanatory yet precise; highlights uncertainties

Table 5. Cont.

Trade-Off	Dimension	DietNerd	NewsNerd	CloudNerd	SpaceNerd
Scope of Filtering		Emphasises exclusion: animal studies, recipes, non-research queries	Emphasises deduplication: same events from multiple sources	Emphasises technical depth: security, best practices, reproducibility	Emphasises identifier consistency: arXiv IDs, DOI validation, metadata completeness

The full installation guide and a step-by-step CloudNerd customization walkthrough are provided in Appendix A with an overview in Section 3.

8. CloudNerd Evaluation

We evaluate CloudNerd on cloud-related Stack Overflow questions using semantic textual similarity (STS) and RAGAS metrics [34]. CloudNerd performs multi-plane tiered retrieval via `tiered_collect_articles()` (CloudNerd code profile plus User Flow **Deep Search** toggle): planes P1–P5 in `helper_functions.py` bind to Stack Overflow via `user_search_apis.py`, merge results across planes, and pass `retrieval_confidence` into synthesis (Appendix J). CloudNerd aims to avoid hallucinations by requiring the synthesis to be evidence-constrained (Appendix J). Benchmark construction and per-question scores appear in Appendices N and Q.

This section reports the full evaluation program: a 20-question training set used for pipeline tuning and component ablations, and a disjoint 120-question test benchmark for three-way comparison with OpenDeepResearch and AutoGPT.

8.1. Comparison Metrics

Semantic textual similarity (STS) has been widely studied as the task of quantifying the degree of semantic equivalence between pairs of texts, typically framed as a continuous similarity score rather than a binary judgment [35]. Recent approaches to STS commonly rely on transformer-based sentence embedding models, which encode sentences into a shared vector space and compute similarity using cosine distance [36]. This embedding-based evaluation paradigm has become standard for comparing generated responses against reference answers in natural language generation and question-answering settings.

Adopting STS and RAGAS [34], we compare CloudNerd with OpenDeepResearch and AutoGPT on a 120-question benchmark, with training set comparisons in Appendix R. Alternative evaluation frameworks such as FActScore [37] have emerged. FActScore provides atomic scoring of factual precision in long-form text, but we prioritize RAGAS here because its criteria (answer relevancy, faithfulness to retrieved context, and answer similarity) align more directly with our cloud troubleshooting use case.

Statistical Significance Testing in Empirical Evaluation

Empirical comparisons between systems require statistical testing to assess the robustness of observed performance differences. Non-parametric testing provides a distribution-free method for estimating statistical significance by repeatedly resampling observed outcomes, making it well-suited for settings with limited sample sizes or unknown score distributions [38]. Such significance tests are commonly paired with confidence interval estimation to quantify the magnitude of performance differences between competing systems. We apply this non-parametric methodology in our evaluations here.

8.2. Experimental Setup

Table 6 summarizes the three systems evaluated on the primary 120-question benchmark. OpenDeepResearch and CloudNerd share the same curated Stack Overflow JSONL corpus and retrieval temporal cutoff; AutoGPT answers the same question IDs via general web browsing without that corpus or the Stack Overflow API.

Table 6. Systems compared on the 120-question primary benchmark ($n = 120$ per system).

System	Architecture	Evidence Access
CloudNerd	Saved-state pipeline: multi-plane tiered retrieval, citation recovery, evidence-constrained synthesis	Curated JSONL; Stack Overflow API; community ranking signals
OpenDeepResearch	LangGraph Open Deep Research (supervisor–researcher–writer–critic) [4,19]	Same JSONL corpus and temporal cutoff; JSONL-only search with Post-ID citations (Appendix M)
AutoGPT	Autonomous tool-using agent [17]	Same question IDs; web browsing without curated corpus or Stack Overflow API

8.2.1. Datasets

Table 7 lists the training and test splits. Each system’s final configuration was tuned on the training set and evaluated once on the test benchmark. The ground-truth answer for every question is the community-validated accepted answer (Appendix N).

Table 7. Evaluation datasets (disjoint splits).

Split	n	Role
Training set	20	Pipeline tuning; component evolution (Table 14)
Test benchmark (primary)	120	All reported three-way comparisons

8.2.2. Benchmark Composition

The primary benchmark (`stackoverflow_with_selection_120_v6.json`) contains 120 questions stratified from an initial pool of 720 Stack Overflow cloud-technology posts (Appendix N). The first 60 questions emphasize implementation and how-to troubleshooting; the second 60 increase representation of causal and contrastive types. Table 8 summarizes the actionability mix. Questions were reformulated from Stack Overflow titles and bodies into natural-language queries a user might ask, while the ground truth is the community-validated accepted answer for each post.

Table 8. Actionability categories in the 120-question test set.

Category	n	Share
Implementation/how-to	44	36.7%
Causal/explanatory	17	14.2%
Contrastive	16	13.3%
Other actionable	43	35.8%
Total	120	100%

8.2.3. Evaluation Protocol

Each system’s configuration was frozen after passing training set quality gates; the 120-question benchmark was evaluated once per system (to prevent tuning hyperparameters

on the test set). CloudNerd’s shipped configuration (*cascade-v1.5*, May 2026) answered all 120 questions. OpenDeepResearch and AutoGPT scores on the same 120 question IDs were merged from baseline CSV exports (Appendix M). Paired permutation tests (10,000 iterations, two-sided) and 95% percentile bootstrap confidence intervals follow Appendix T. Internal repository run labels (*cascade-v1.5*, *combined-v1*, etc.) appear only in Appendix A8 for the purpose of reproducibility.

8.2.4. Pipeline Stages Under Test

Every CloudNerd answer traverses the following stages: (1) question validity gating, (2) query expansion, (3) multi-plane tiered retrieval (P1–P5; benchmark: P1 strict, P2 title-only, P4 extended keywords), (4) article organization, (5) LLM relevance filtering with optional TF-IDF citation recovery, (6) evidence-constrained synthesis, (7) citation matching, and (8) STS/RAGAS scoring. Supplemental retrieval (*gap-fill*) runs when P1 coverage is thin or key question terms are missing from the retrieved text.

8.2.5. Temporal Constraint

For each question, only Stack Overflow posts created *before* the ground-truth answer’s posting date are visible to retrieval, matching the protocol in Appendix N. This prevents answers that use the best answer.

8.2.6. Baselines

On the 120-question benchmark test set, we compare CloudNerd against OpenDeepResearch and AutoGPT using the same question IDs, with identical temporal cutoffs and the same accepted-answer ground truth. OpenDeepResearch uses the same curated Stack Overflow JSONL corpus as CloudNerd (Appendix M). AutoGPT answers the same question IDs via general web browsing *without* access to that corpus or the Stack Overflow API, representing autonomous agents that do not inherit a domain owner’s curated retrieval configuration. Training set comparisons against both baselines appear in Appendix R.

8.2.7. Baseline Score Provenance

OpenDeepResearch test set-120 scores come from a Stack Overflow–adapted LangGraph Open Deep Research run aligned to `stackoverflow_with_selection_120_v6.json` question IDs (`baseline-opendeeprersearch-primary-120.csv`).

8.2.8. CloudNerd Evaluation Settings

CloudNerd loads the `CloudNerd_cascade` profile (`RETRIEVAL_NERD_PROFILES`) with `cascade_retrieval.visible` enabled and three key retrieval flags: `CITATION_RESCUE_ENABLED=true`, `RETRIEVAL_GAP_FILL_ENABLED=true`, and `ENABLE_QUERY_FALLBACK=false`. Final synthesis uses `openai_prompts_combined_v1_5.py` (four evidence-constrained rules on a direct-answer-first template) with `FINAL_RESPONSE_TEMPERATURE=0.2` and no two-stage faithfulness pass (Appendix C.2).

8.2.9. Statistical Testing

Paired permutation tests (10,000 iterations) with 95% bootstrap confidence intervals follow Ref. [38], with the methodology discussed in detail in Appendix T.

8.3. Main Results: 120-Question Benchmark

Table 9 is the test set comparison. Table 10 summarizes winner-vs-second-best margins. Tables 11 and 12 give paired head-to-head tests. Table 13 lists CloudNerd retrieval diagnostics.

Table 9. Three-way aggregate means on the 120-question test set benchmark ($n = 120$). Bold = best mean per metric column.

System	STS	Faith.	Rel.	Sim.
CloudNerd	0.695	0.559	0.693	0.866
OpenDeepResearch	0.671	0.862	0.848	0.857
AutoGPT	0.578	0.096	0.670	0.823

Table 10. Winner vs. second-best on the 120-question benchmark (paired permutation test, 10,000 iterations, 95% bootstrap confidence intervals).

Metric	Winner	2nd	Margin	p	95% CI	Sig.?
STS	CloudNerd	OpenDeepResearch	+0.024	0.011	[+0.006, +0.042]	Yes
Faithfulness	OpenDeepResearch	CloudNerd	+0.303	<0.001	[+0.240, +0.367]	Yes
Answer Relevancy	OpenDeepResearch	CloudNerd	+0.155	<0.001	[+0.083, +0.229]	Yes
Answer Similarity	CloudNerd	OpenDeepResearch	+0.008	0.002	[+0.003, +0.013]	Yes

Table 11. Non-parametric paired permutation test: RAG-based CloudNerd vs. RAG-based OpenDeepResearch on 120 matched questions (10,000 iterations, 95% bootstrap confidence intervals).

Metric	CloudNerd	OpenDeepResearch	Δ Mean	p	95% CI	Sig.?
STS	0.695	0.671	+0.024	0.011	[+0.006, +0.042]	Yes
Faithfulness	0.559	0.862	-0.303	<0.001	[-0.368, -0.239]	Yes
Answer Similarity	0.866	0.857	+0.008	0.002	[+0.003, +0.013]	Yes
Answer Relevancy	0.693	0.848	-0.155	<0.001	[-0.230, -0.084]	Yes

Table 12. Non-parametric paired permutation test: RAG-based CloudNerd vs. Agentic AutoGPT on 120 matched questions (10,000 iterations, 95% bootstrap confidence intervals).

Metric	CloudNerd	AutoGPT	Δ Mean	p -Value	95% CI	Sig.?
STS	0.695	0.578	+0.117	<0.001	[+0.087, +0.149]	Yes
Faithfulness	0.559	0.096	+0.463	<0.001	[+0.392, +0.530]	Yes
Answer Similarity	0.866	0.823	+0.042	<0.001	[+0.034, +0.051]	Yes
Answer Relevancy	0.693	0.670	+0.023	0.592	[-0.061, +0.106]	No

On the 120-question test benchmark, CloudNerd leads in STS (0.695 vs. 0.671 OpenDeepResearch, 0.578 AutoGPT) and answer similarity (0.866 vs. 0.857 OpenDeepResearch, 0.823 AutoGPT). OpenDeepResearch leads on faithfulness (0.862 vs. 0.559 CloudNerd) and answer relevancy (0.848 vs. 0.693 CloudNerd). All four winner-vs-second-best margins are statistically significant at $p < 0.05$ (Table 10), and AutoGPT ranks last on every aggregate metric (Table 9).

This split result highlights the quality tradeoff between two RAG-based systems with different orchestration strategies: CloudNerd is stronger on semantic alignment (STS/similarity), while OpenDeepResearch is stronger on claim grounding and direct question targeting (faithfulness/relevancy). The paired comparisons below quantify this tradeoff metric-by-metric.

Table 13. CloudNerd aggregates metrics and retrieval diagnostics on the 120-question benchmark ($n = 120$).

Metric	Mean	Notes
Scored questions	120/120	All questions answered
STS (all-mpnet-base-v2 cosine)	0.695	SD 0.152; vs. ground-truth answer
RAGAS Faithfulness	0.559	Grounded in retrieved contexts
RAGAS Answer Similarity	0.866	vs. ground-truth answer
RAGAS Answer Relevancy	0.693	26 questions with relevancy = 0
RAGAS Geometric Mean	0.791	$n = 83$ questions with faith., sim., rel. all > 0
Retrieval strictness tier: strict	84	test set retrieval match (70%)
Retrieval strictness tier: title-only	33	Relaxed title match (28%)
Retrieval strictness tier: extended keywords	3	Expanded keyword match (2%)
Supplemental retrieval triggered	36	Gap-fill when strict tier is thin
Citation recovery enabled	120/120 scored	Dev-set ablation: scorable rate 6/20 \rightarrow 20/20 (Table 14)

8.3.1. CloudNerd vs. AutoGPT.

CloudNerd is significantly better on STS ($\Delta = +0.117$, $p < 0.001$), faithfulness ($\Delta = +0.463$, $p < 0.001$), and answer similarity ($\Delta = +0.042$, $p < 0.001$). Answer relevancy shows a small positive mean difference ($\Delta = +0.023$) but the p-value is too high to declare the difference statistically significant.

8.4. Limitations

- **RAGAS relevancy zeros:** 26 of 120 test questions have relevancy = 0 despite non-empty answers, usually due to answer structure or thin retrieval rather than empty contexts.
- **Faithfulness vs. OpenDeepResearch:** On the 120-question benchmark, OpenDeepResearch scores higher on faithfulness ($\Delta = -0.303$, $p < 0.001$; Table 11). We will see that concatenating the answers increases faithfulness even further.
- **Community-based ground-truth answer:** We use accepted Stack Overflow answers as ground truth; alternative community-validated answers may differ in wording or substance.

Details of benchmark construction, statistical methodology, the full 120-question score table, and the complete evaluation run history appear in Appendices Appendices N, Q, R and T.

8.5. Lessons from Tuning on the Training Set

8.5.1. Component Contributions

Table 14 summarizes how each pipeline mechanism changed scores on the 20-question training set during CloudNerd refinement. Rows are sequential design steps; each row isolates the effect of adding one mechanism while holding the benchmark split fixed ($n = 20$). Appendix Tables A8–A10 condense the full repository run log and prompt experiments documented in CLOUDNERD_EVALUATION_HISTORY_DETAILED.md.

Table 14. Training set ($n = 20$) pipeline evolution toward the evaluated CloudNerd system. Each row adds mechanisms on top of tiered retrieval (P1–P5). **Scored** = questions with non-empty citations and RAGAS/STS scores.

Milestone	Key Change	Scored	Faith	Rel	STS	Fallbacks
Tiered retrieval only	Tiered retrieval; strict relevance filter; no citation recovery	6/20	0.627	0.447	0.587	11
Citation recovery pipeline	+ TF-IDF post-matcher recovery + softened relevance classifier	20/20	0.455	0.486	0.677	5
Direct-answer synthesis prompt	+ obligating final-response template (direct answer first)	19/20	0.500	0.701	0.668	0
CloudNerd (evaluated)	+ evidence-constrained answer rules + supplemental retrieval + $T = 0.2$	20/20	0.571	0.803	0.664	0

Legacy Baselines (Training Set)

Strict single-pass retrieval with a hard relevance classifier correctly scored only 5 of the 20 training questions (faithfulness = 0.864 for those 5 questions, while 75% of the questions remain unscorable). Relaxed query fallback increased coverage to 11 of the 20 training questions, but did not resolve the citation pipeline issue: posts were retrieved but subsequently discarded before synthesis. These issues motivated the development of a cascade retrieval and citation recovery approach.

Citation Recovery and Softer Relevance Classification

With tiered retrieval alone and a strict relevance classifier, only 6 of the 20 training set questions were scorable despite retrieval succeeding on most questions—articles were dropped before synthesis, leaving empty citation objects. Enabling TF-IDF citation recovery and a softer relevance classifier that accepts Docker, Kubernetes, Terraform, and LibreOffice deployment posts raised the scorable rate to 20/20. Fallback refusals dropped from 11 to 5 because more posts reached the final prompt, but answer relevancy remained low (0.486) with the original final-response wording.

Direct-Answer and Evidence-Constrained Synthesis Prompts

Replacing the original final prompt with a direct-answer-first template increased RAGAS answer relevancy from 0.486 to 0.701 and eliminated all fallback refusals on the training set. CloudNerd adds four evidence-constrained answer rules on top of that template: (1) no configuration keys or commands absent from evidence; (2) direct answer in sentence 1 reusing question terms; (3) no synthesis across posts unless one of the posts states the conclusion; (4) gap statements last only. On the training set, this step further improved faithfulness (0.571 vs. 0.500) and relevancy (0.803 vs. 0.701) while maintaining 20/20 scorable questions. Complete prompt listings appear in Appendices C.2 and J.

Supplemental Retrieval

After tiered retrieval (P1–P5), supplemental Stack Overflow search runs when P1 results are thin (≤ 2 articles) or when key question terms are missing from the retrieved text. On the 120-question benchmark, supplemental retrieval triggered on 36/120 questions. Of the 120 test questions, 84 used the strict retrieval tier, 33 used title-only retrieval, and 3 used extended-keyword retrieval. An expanded supplemental-retrieval criterion (evaluated

with an alternative prompt variant) failed the development-set quality gate (relevancy 0.803 → 0.753) and was not deployed.

8.5.2. Supplementary Development-Set Analysis

On the development set, CloudNerd outperforms OpenDeepResearch on STS ($\Delta = +0.057$, $p = 0.021$) and answer similarity ($\Delta = +0.016$, $p = 0.016$). OpenDeepResearch remains stronger on faithfulness ($\Delta = -0.276$, not significant at $\alpha = 0.05$ on $n = 20$). CloudNerd exceeds AutoGPT on STS ($\Delta = +0.137$, $p < 0.001$), faithfulness ($\Delta = +0.485$, $p < 0.001$), and similarity ($\Delta = +0.041$, $p < 0.001$); relevancy gains vs. AutoGPT are directionally positive but not significant at $n = 20$.

Table 15. Development-set supplementary comparison: CloudNerd vs. OpenDeepResearch and AutoGPT ($n = 20$). Paired permutation test; significance at $\alpha = 0.05$.

Metric	vs. Open Deep Research Δ	p	vs. AG Δ	p
STS	+0.057	0.021	+0.137	<0.001
Faithfulness	-0.276	n.s.	+0.485	<0.001
Answer Similarity	+0.016	0.016	+0.041	<0.001
Answer Relevancy	+0.012	n.s.	+0.092	n.s.

Tier-Stratified Development-Set Analysis

Tables 16 and 17 stratify development-set means by the winning cascade strictness tier (direct-answer synthesis prompt configuration). CloudNerd leads AutoGPT on STS in every tier (+0.106 strict, +0.180 title-only, +0.133 extended keywords) and on faithfulness in every tier (+0.344, +0.231, +0.359 respectively). Relevancy beats AutoGPT on the strict tier (+0.051) but trails on title-only (-0.257) and extended-keyword (-0.150) tiers—harder questions where retrieval was forced to weaker fallback levels.

Table 16. Development-set tier-stratified means: CloudNerd vs. AutoGPT by winning retrieval strictness tier ($n = 20$).

Tier	n	STS (CloudNerd)	STS (AutoGPT)	Δ STS	Rel (CloudNerd)	Rel (AutoGPT)
Strict	11	0.618	0.512	+0.106	0.795	0.744
Title-only	3	0.684	0.504	+0.180	0.285	0.541
Extended keywords	6	0.698	0.565	+0.133	0.586	0.736

Table 17. Development-set faithfulness advantage of CloudNerd over AutoGPT by retrieval strictness tier ($n = 20$).

Tier	Δ Faithfulness (CloudNerd – AutoGPT)
Strict	+0.344
Title-only	+0.231
Extended keywords	+0.359

8.5.3. Ablation and Hallucination-Control Evidence

We report three sources of evidence on whether hallucination-control mechanisms improve faithfulness without collapsing answer quality.

Adding Evidence-Constrained Answer Rules

On the training set, adding the four evidence-constrained rules improved faithfulness (+0.071) and relevancy (+0.102) relative to the direct-answer synthesis prompt without collapsing scorable rate (Table 14).

Withdrawn Strict-Grounding Designs

Subset experiments with strict grounding synthesis prompts and two-stage verbatim synthesis raised faithfulness on the worst five extrapolation questions but drove relevancy to approximately 0.32 (Table 18). These designs were not deployed because they over-corrected the answer structure for RAGAS relevancy.

Table 18. Withdrawn faithfulness-oriented configurations on problem-question subsets (not deployed). Strict grounding raised faithfulness in difficult cases but collapsed answer relevancy.

Configuration	n	Faith	Rel	Sim	Outcome
Strict grounding synthesis prompt	13	0.454	0.335	0.851	Not deployed
Hybrid retrieval with TF-IDF relevance ceiling	13	0.503	0.472	0.794	Not deployed
Two-stage synthesis (verbatim fact binding)	5	0.880	0.323	0.829	Not deployed
Two-stage synthesis (refined fact binding)	5	0.876	0.322	0.832	Not deployed
Actionable-first prompt with expanded supplemental retrieval	20	0.562	0.753	0.856	Failed quality gate

Single-Knob Ablations (Set-B Training Set)

Table 19 reports dedicated runs on the 20-question training split ($n = 20$), where each row changes exactly one mechanism relative to the evaluated CloudNerd configuration (cascade-v1.5). The pattern is clear: citation rescue is the main coverage driver (16 of 20 scored and 4 zero-retrieval failures when disabled), supplemental gap fill is the main answer-relevancy driver ($0.803 \rightarrow 0.506$ when disabled), and legacy single-pass retrieval materially reduces robustness (13 of 20 scored, 7 zero-retrieval failures). These controlled ablations align with the sequential pipeline progression in Table 14.

Table 19. Single-knob ablations on the 20-question training set ($n = 20$). Each row changes one setting relative to evaluated CloudNerd (cascade-v1.5); all other settings match the baseline.

Run	Single Change vs. Baseline	Scored (n/20)	Zero Ret.	STS	Faith.	Rel.	Sim.
Baseline	None (evaluated config)	20/20	0	0.664	0.571	0.803	0.861
Abl. 1	Disable citation rescue (CITATION_RESCUE_ENABLED=false)	16/20	4	0.653	0.636	0.660	0.858
Abl. 2	Disable supplemental gap fill (RETRIEVAL_GAP_FILL_ENABLED=false)	19/20	1	0.658	0.582	0.506	0.856
Abl. 3	Use old synthesis prompt (citation_relevancy_v1)	20/20	0	0.672	0.527	0.713	0.861
Abl. 4	Enable query fallback (ENABLE_QUERY_FALLBACK=true)	20/20	0	0.662	0.541	0.800	0.862
Abl. 5	Switch to legacy retrieval mode (RETRIEVAL_MODE=legacy)	13/20	7	0.654	0.507	0.680	0.856

8.5.4. Telemetry on the 20-Question Training Set

To complement answer-quality metrics, we instrumented the 20-question training run (*CloudNerd Deep Search ON*, cascade P1–P5) with per-question telemetry exports

(`benchmark_results.csv/json`). The run used API-key rotation across two 10-question batches and logs wall-clock runtime, input/output tokens, total tokens, and estimated USD cost per question.

Table 20. CloudNerd Telemetry summary for the 20-question training set in cascade mode.

Metric	Value
Run size	20 questions
Total runtime	7373.294 s
Average runtime per question	368.665 s
Total input tokens	8,060,856
Total output tokens	474,234
Total tokens	8,535,090
Average tokens per question (total)	426,754.50
Average tokens per question (input/output)	403,042.80/23,711.70
Total estimated cost	\$94.835580
Average estimated cost per question	\$4.741779

Runtime and cost are heterogeneous across questions: the median runtime is 358.524 s (min 321.377 s; max 507.017 s), and per-question estimated costs range from USD 2.787760 to USD 9.807640. These telemetry values are reported for the 20-question training set only.

CloudNerd, OpenDeepResearch, and AutoGPT use different control-flow patterns and tool invocation counts.

8.5.5. Adjusted-Time Analysis Under Reduced 429 Backoff

To quantify the runtime impact of retrieval throttling, we report an adjusted-time analysis using observed CloudNerd telemetry from the same 20-question run. Let

$$t_{\min} = 321.377 \text{ s}, \quad t_{\text{avg}} = 368.665 \text{ s}, \quad n = 20.$$

We define per-question overhead relative to the fastest observed question as

$$\Delta t = t_{\text{avg}} - t_{\min} = 47.288 \text{ s},$$

and total run-level overhead as

$$\Delta T = n \cdot \Delta t = 945.76 \text{ s},$$

which is 12.8% of the observed total runtime (7373.294 s). This is a scenario estimate (not direct per-request attribution) of latency potentially recoverable if 429-driven backoff and effective serialization are reduced.

For recovery fraction $r \in [0, 1]$, the adjusted runtime is:

$$t_{\text{adj}}(r) = t_{\text{avg}} - r\Delta t, \quad T_{\text{adj}}(r) = n \cdot t_{\text{adj}}(r).$$

Interpretation: even partial recovery of throttling overhead is material. At 50% recovery, expected savings are about 472.9 s over 20 questions (~7.9 min), with throughput improving from 9.77 to 10.43 questions/hour. At the floor proxy (100% recovery), savings are 945.8 s (~15.8 min), and throughput reaches 11.20 questions/hour.

For AutoGPT, telemetry records on the 20-question evaluation subset shown in Table 22 reflect end-to-end wall-clock runtime for each question and LLM usage reported by the agent execution loop, and the five metrics are also recorded.

Table 21. Adjusted-time scenarios from reduced rate-limit overhead on the 20-question CloudNerd run.

Recovered Overhead	Adjusted Avg Runtime (s/q)	Adjusted Total Runtime (20Q)	Throughput (q/hour)
0% (observed baseline)	368.665	7373.294 s	9.77
25% of ΔT	356.843	7136.860 s	10.09
50% of ΔT	345.021	6900.430 s	10.43
75% of ΔT	333.199	6663.980 s	10.81
100% of ΔT (floor proxy)	321.377	6427.540 s	11.20

Table 22. AutoGPT telemetry summary for the 20-question training set.

Metric	Value
Run size	20 questions
Total runtime	3208.006 s
Average runtime per question	160.400 s
Total input tokens	462,936
Total output tokens	19,100
Total tokens	482,036
Average tokens per question (total)	24,101.80
Average tokens per question (input/output)	23,146.80/955.00
Total estimated cost	\$5.202360
Average estimated cost per question	\$0.260118

Table 23 shows the results for OpenDeepResearch.

Table 23. OpenDeepResearch telemetry summary for the 20-question training set.

Metric	Value
Run size	20 questions
Total runtime	2656.627 s
Average runtime per question	132.831 s
Total input tokens	1,178,817
Total output tokens	58,923
Total tokens	1,237,740
Average tokens per question (total)	61,887
Average tokens per question (input/output)	58,940.85/2946.15
Total estimated cost	\$13.56
Average estimated cost per question	\$0.68

8.6. Reproducibility and Artifacts

The evaluation artifacts are version-controlled in the Custom-Nerd repository (<https://github.com/Harsh23Kashyap/Custom-Nerd>):

- CloudNerd 120-Q scores: `result/runs-120/cascade-v1.5/eval_120_v1.5.csv` (and matching `.json`, HTML report).
- Three-way aggregates and paired tests: `result/runs-120/OtherResearch/dennis-three-way-primary-120.csv`, `comparison-cloudnerd-vs-baselines-primary-120.csv`.

- Baseline exports: `baseline-opendeepresearch-primary-120.csv`, `baseline-autogpt-primary-120.csv`.
- Training set runs: `result/setB_runs/cascade-v1.5/eval_setB.csv` and related comparison HTML under `result/setB_runs/comparison/`.
- Full run narrative: `result/CLOUDNERD_EVALUATION_HISTORY_DETAILED.md`.
Benchmark questions: `stackoverflow_with_selection_120_v6.json` (Appendix N).
Pin the repository commit hash when submitting to the journal or Overleaf data archive.

8.7. Discussion

8.7.1. Domain-Specific Optimization Trade-Offs

Retrieval and community-signal ranking improve alignment with ground-truth answers on STS and answer similarity compared with both OpenDeepResearch and AutoGPT on the 120-question benchmark. OpenDeepResearch scores higher on faithfulness and answer relevancy ($\Delta = -0.303$ and -0.155 vs. CloudNerd; Table 11), indicating that its LangGraph writer–critic loop grounds claims more tightly in retrieved context when both systems share the same corpus. We treat this as an expected architectural trade-off (Table 2): bounded, configuration-first pipelines optimize scorable rate and accepted-answer alignment, whereas iterative agentic synthesis can prioritize claim-level grounding at higher token and latency cost. Closing the faithfulness gap without sacrificing relevancy remains open; strict two-stage grounding variants tested during development were not deployed because they over-corrected the answer structure for RAGAS relevancy.

8.7.2. Cost and Latency (Design-Level)

We did not instrument wall-clock time or token usage for all 120-question runs in this submission, because the CloudNerd, OpenDeepResearch, and AutoGPT architectures are too different. Architecturally, CloudNerd uses a fixed sequence of bounded prompts with parallel relevance scoring, whereas OpenDeepResearch may execute additional graph hops when the critic requests revision. AutoGPT incurs open-web search latency without curated corpus reuse. Section 8.4 states this explicitly; a follow-up eval pass will log per-question latency and token totals for all three systems on the same hardware profile.

9. Combining CloudNerd with OpenDeepResearch

Section 8 shows that CloudNerd and OpenDeepResearch excel on different RAGAS dimensions on the same 120-question benchmark: CloudNerd leads on semantic textual similarity (STS) and answer similarity, whereas OpenDeepResearch leads on faithfulness and answer relevancy (Tables 10 and 11). After considering these complementary strengths, we ran a diagnostic experiment to test whether a naive hybrid—concatenating the two systems' final answers—could improve aggregate scores without changing either pipeline.

9.1. Protocol

For each of the 120 primary benchmark questions, we formed a **concat** answer by appending the OpenDeepResearch synthesis after the CloudNerd synthesis, separated by a horizontal rule marker on its own line. CloudNerd answers and retrieved contexts came from the canonical `cascade-v1.5` run (`eval_120_v1.5.json`); OpenDeepResearch answers and passages came from the full LangGraph evaluation report (120 primary IDs). We rescored all three conditions with the same metrics as Section 8: STS (`all-mpnet-base-v2` cosine vs. the accepted Stack Overflow answer) and RAGAS faithfulness, answer relevancy, and answer similarity (GPT-4o-mini judge, ragas 0.1.21). For RAGAS faithfulness on the **concat** condition, we supplied the **union** of CloudNerd retrieved contexts and OpenDeepResearch passages—an optimistic upper bound because the judge sees evidence

from both retrieval stacks. Paired non-parametric permutation tests (10,000 iterations, same procedure as Appendix T) compare concat against each baseline on matched question IDs; $n = 120$ except faithfulness on concat ($n = 119$; one RAGAS timeout).

9.2. Results

Table 24 summarizes aggregate scores. Concat improves STS relative to both baselines (0.722 vs. 0.695 CloudNerd and 0.671 OpenDeepResearch; both $p < 0.001$; Table 25). Merging the two answers moves generated text closer to the accepted community answer on average—likely because the combined text captures more of the ground-truth phrasing when the two systems emphasize different aspects of the solution.

Table 24. Aggregate metrics on the 120-question benchmark: CloudNerd, OpenDeepResearch, and naive concat (CN answer + ODR answer).

Metric	CloudNerd	OpenDeepResearch	Concat
STS	0.695	0.671	0.722
Faithfulness	0.559	0.862	0.924 ^a
Answer Relevancy	0.693	0.848	0.673
Answer Similarity	0.866	0.857	0.856

^a Faithfulness on concat: $n = 119$ (one timeout). Concat faithfulness uses union contexts from both systems.

Table 25. Paired permutation tests: naive concat vs. CloudNerd and vs. OpenDeepResearch on 120 matched questions (10,000 iterations).

Metric	Concat – CloudNerd			Concat – OpenDeepResearch		
	Δ	p	Sig.?	Δ	p	Sig.?
STS	+0.027	<0.001	Yes	+0.051	<0.001	Yes
Faithfulness	+0.361	<0.001	Yes	+0.062	0.001	Yes
Answer Relevancy	−0.021	0.599	No	−0.176	<0.001	Yes
Answer Similarity	−0.009	<0.001	Yes	−0.001	0.546	No

Faithfulness row: $n = 119$ for concat comparisons. Negative Δ on answer similarity (concat vs. CloudNerd) indicates concat scores lower despite statistical significance.

On faithfulness, concat reaches 0.924—above CloudNerd (+0.361, $p < 0.001$) and OpenDeepResearch (+0.062, $p = 0.001$). Longer answers judged against the union of both retrieval sets score as highly grounded. This supports the hypothesis that the two systems retrieve complementary evidence, but the measurement is optimistic: faithfulness is computed against combined contexts, not against a single deployable retrieval policy.

Answer relevancy tells the opposite story. Concat scores 0.673—statistically indistinguishable from CloudNerd ($\Delta = -0.021$, $p = 0.599$) but significantly below OpenDeepResearch ($\Delta = -0.176$, $p < 0.001$). Appending a second full synthesis often produces survey-style text; RAGAS penalizes diluted directness even when each system alone scored well on relevancy. On several questions where CloudNerd had high relevancy (>0.9), concat relevancy dropped to 0.0 after append (e.g., question IDs 74470489, 65957504, 57764237).

Answer similarity remains high for all three conditions (0.856–0.866) and is essentially unchanged vs. OpenDeepResearch ($\Delta = -0.001$, $p = 0.546$); concat is slightly below CloudNerd ($\Delta = -0.009$, $p < 0.001$).

9.3. Interpretation

Naive concatenation is a useful diagnostic, not a deployable hybrid. It confirms complementarity on STS and grounding (under union-context faithfulness) while exposing

a sharp relevance cost relative to OpenDeepResearch’s writer–critic synthesis. A production hybrid would need *selective* fusion—for example, using OpenDeepResearch to verify or refine a CloudNerd draft—rather than appending both answers in full. Reproducibility scripts and score exports are under `result/runs-120/OtherResearch/cn-odr-concat-primary-120/` in the project repository; see Appendix R.

10. Conclusions and Future Work

CUSTOMNERD is a configuration-first framework for deploying expertise-driven retrieval-augmented generation systems without modifying core pipeline code. We know it is easy to get running, because even biology students become operational with the system within three hours or less.

Domain-specific behavior—prompts, retrieval logic, UI theming, and deployment settings—is externalized into a saved-state configuration that domain experts can edit through a web interface. Four illustrative deployments (DietNerd, NewsNerd, CloudNerd, and SpaceNerd) show that the same nine-stage pipeline adapts across medical, news, cloud-infrastructure, and space-science use cases.

On a 120-question Stack Overflow benchmark with community-validated ground truth, the RAG-based CloudNerd and OpenDeepResearch showed complementary strengths. Both had better STS and RAGAS scores than the pure agentic approach of AutoGPT. This suggests that, for now at least, curated community retrieval and evidence-constrained synthesis materially improve expertise question-answering over open-web autonomous search.

As we show in Section 9, even simple concatenation of CloudNerd and OpenDeepResearch syntheses improves STS and faithfulness (under union-context scoring) relative to either system alone, but significantly reduces answer relevancy compared with OpenDeepResearch. Future work, therefore, includes integrating selective agentic verification—in the spirit of OpenDeepResearch’s writer–critic loop—into the configuration-first pipeline to close the remaining faithfulness gap without sacrificing the saved-state, no-code deployment model or the answer similarity gains our direct-answer synthesis prompt achieved. Other future work includes reducing the query cost substantially.

Author Contributions: |

Funding: |

Data Availability Statement: |

Conflicts of Interest: |

Appendix A. Custom Nerd Installation Guide

The main text already provides the step-by-step installation. This appendix only adds quick checks and reference links.

Appendix A.1. Prerequisites and Quick Verification

Windows-specific: When installing Python, ensure “Add Python to PATH” is checked.

Verification procedure

1. Open Terminal (macOS/Linux) or PowerShell (Windows).
2. Run `python3 --version` (`py --version` on Windows).
3. Verify Python version is 3.11 or 3.12.

Appendix A.2. API Docs and Front-End

- Backend **API docs**: <http://localhost:8000/docs> (served when the backend is running).
- Front-end preview: open `customnerd-website/index.html` in a browser; some features remain inactive until the backend is running.

Appendix A.3. Configure Your API Keys

1. Open the file: `customnerd-backend/variables.env`
2. Set the LLM variable to your preferred provider: OpenAI, Gemini, Claude, or Ollama.
3. If using a cloud provider, add the corresponding API key:
 - `OPENAI_API_KEY` – required when `LLM=OpenAI`
 - `GEMINI_API_KEY` – required when `LLM=Gemini`
 - `ANTHROPIC_API_KEY` – required when `LLM=Claude`
4. **Ollama (local, no API key required)**: Set `LLM=Ollama` and optionally set `OLLAMA_MODEL` (e.g., `llama3.2`) and `OLLAMA_BASE_URL` (default: `http://localhost:11434`). The web interface will automatically install Ollama, download the selected model, and start the server when you save the configuration.
5. Optionally add data-source API keys:
 - `NCBI_API_KEY` – enhanced PubMed rate limits
 - `ELSEVIER_API_KEY`, `SPRINGER_API_KEY`, `WILEY_API_KEY`—**optional** publisher data
6. Save the file and restart the backend server.

Table A1. System requirements.

Requirement	Minimum Version	Purpose
Python	3.11 or 3.12	Executes the application code
RAM	4 GB (8 GB preferred)	Prevents excessive swapping
Disk Space	2 GB	Repository + virtual env storage
Web Browser	Recent Chrome, Edge, Firefox	Accesses the UI
Internet Connection	Stable	Downloads packages and APIs

Appendix A.4. Logs

All installation logs are automatically saved in the `logs/` folder:

- `logs/presetup/` – Logs from `presetup.py` (Windows only)
- `logs/setup/` – Logs from `setup.py`
- `logs/run/` – Logs from `run.py`

Log files are named with the date and time (e.g., `setup_24_Dec_2025_8_26_pm.log`). If you encounter errors, check these logs for detailed information.

Appendix A.5. Troubleshooting Guide

Tip: If you encounter errors during installation, try using ChatGPT or Gemini—they are really helpful for debugging installation issues!

Appendix A.6. Estimated Installation Time

Most users finish installation in approximately 2 h; network speed and hardware may affect this figure. The setup script downloads and installs many packages, which requires patience.

Table A2. Common problems and fixes.

Issue	Likely Cause	Recommended Action
ModuleNotFoundError	Virtual environment not activated or missing dependencies	Activate venv; run <code>setup.py</code> again or <code>pip install -r requirements.txt</code>
404 Not Found	Backend inactive or wrong port	Check if server is running on port 8000; verify <code>run.py</code> executed successfully
OSError [Errno 98] or Address already in use	Port 8000 busy	Free the port or modify <code>run.py</code> to use a different port
Setup script fails	Network issues or missing dependencies	Check logs in <code>logs/setup/</code> ; run <code>setup.py</code> again (it resumes from where it left off)
WSL not working (Windows)	WSL2 not properly installed	Run <code>presetup.py</code> as Administrator; restart computer if needed

Appendix A.7. Conclusion

Following these steps yields a fully operational instance of Custom Nerd capable of retrieving scientific articles, generating summaries, and answering technical queries.

Appendix B. CUSTOMNERD Web Interface Walkthrough

This section provides a visual guide to using the CUSTOMNERD web interface, demonstrating how to navigate between different NerdEngine variants, manage saved states, and configure the frontend appearance.

Appendix B.1. Home Page

Figure A1 shows the main interface after the server starts. The home page displays the CUSTOMNERD logo and a question input bar where users can submit queries. This is the default entry point for all NerdEngine variants.

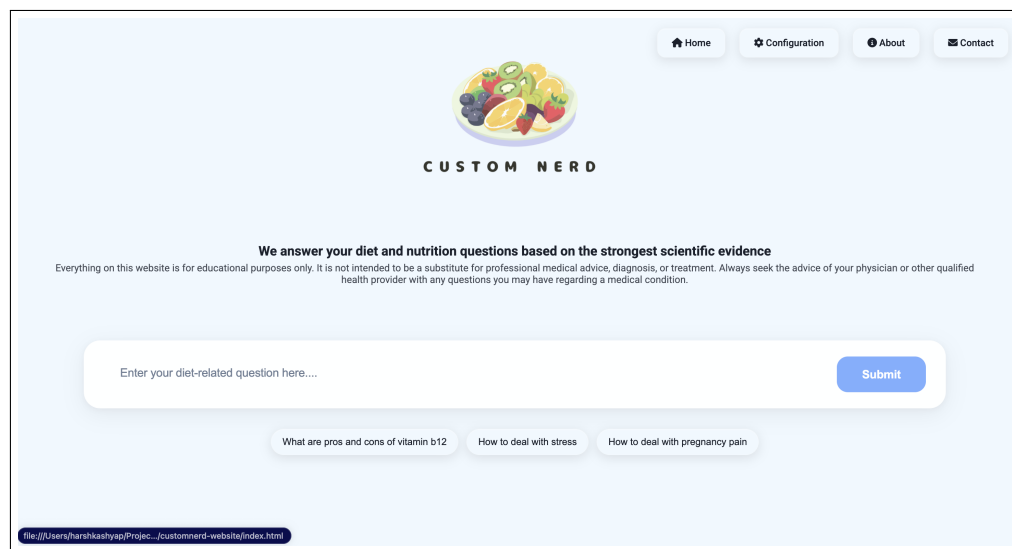


Figure A1. The CUSTOMNERD home page with question input interface.

Appendix B.2. About Page

The **About** page provides information about the NerdEngine variant, including its purpose, data sources, and any domain-specific disclaimers. Figure A2 shows the About page interface, which can be customized through the Frontend Configuration settings.

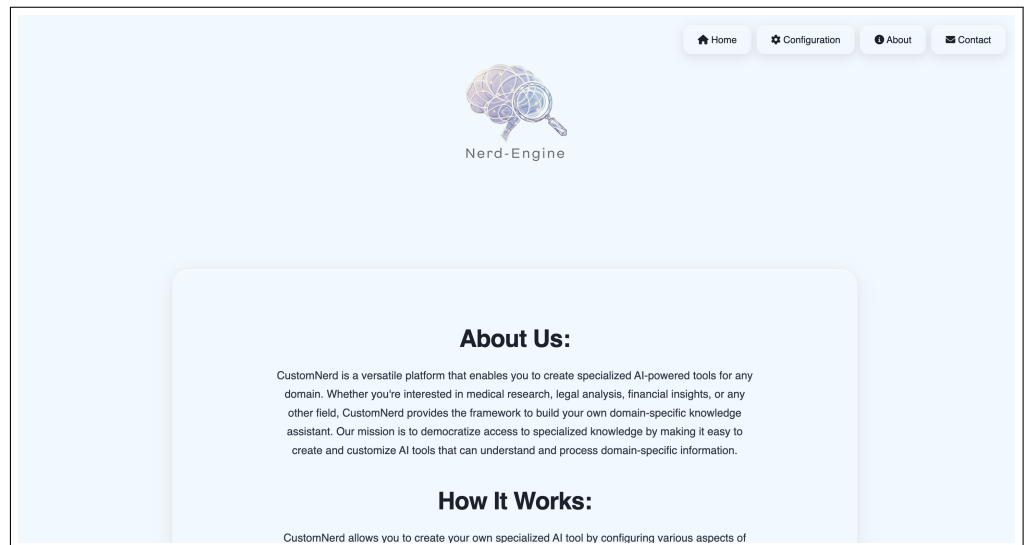


Figure A2. The About page displaying information about the active NerdEngine variant.

Appendix B.3. Preconfigured NerdEngine Variants

CUSTOMNERD ships with five preconfigured NerdEngine variants, each tailored to a specific domain. Figures A3–A7 showcase the home pages for each variant, demonstrating how frontend customization creates distinct visual identities while maintaining consistent functionality.

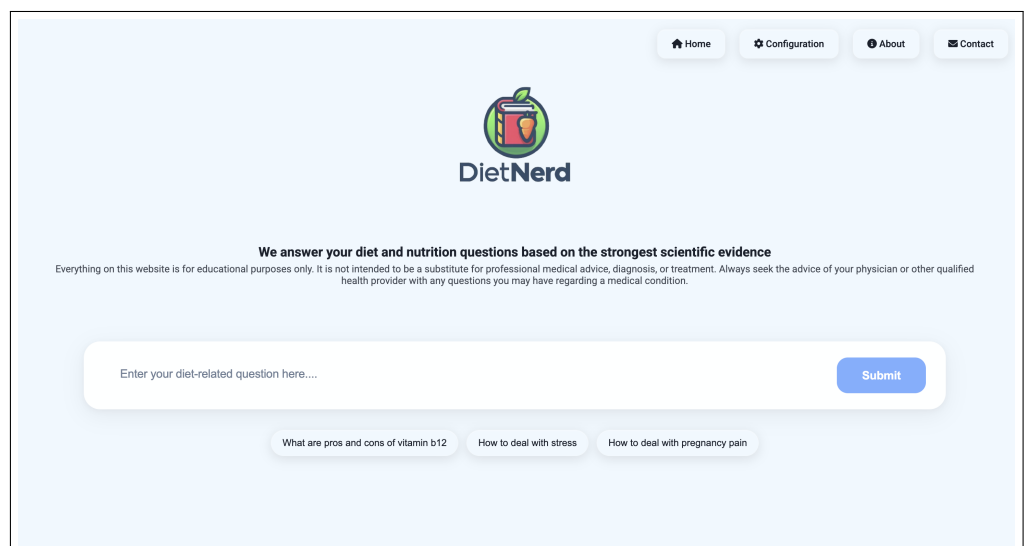


Figure A3. DietNerd home page: Evidence-based medical and nutrition research assistant.

Each variant maintains the same core interface structure—question input bar, navigation menu, and configuration access—while presenting domain-specific branding, colors, taglines, and disclaimers configured through the Frontend Configuration panel.

Appendix B.4. Managing Saved States

CUSTOMNERD ships with five preconfigured NerdEngine variants: **DietNerd**, **SpaceNerd**, **CloudNerd**, **SciNerd**, and **NewsNerd**. You can load any of these as starting templates, customize them, and save your modifications as new states. This workflow enables rapid iteration: for example, after building **LegalNerd**, you can save that state and immediately begin configuring **CalorieNerd** without losing your previous work.

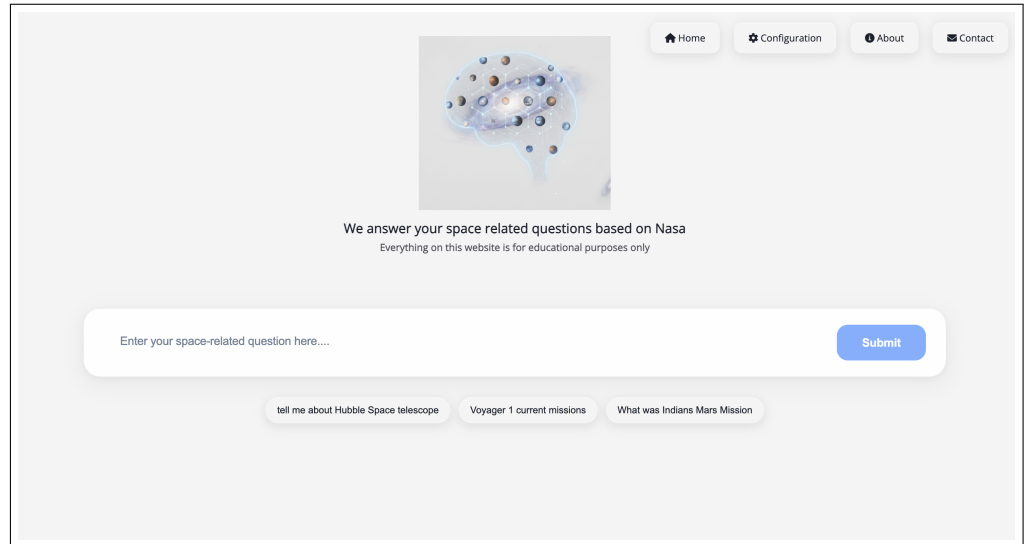


Figure A4. SpaceNerd home page: Astronomy and space science research assistant.

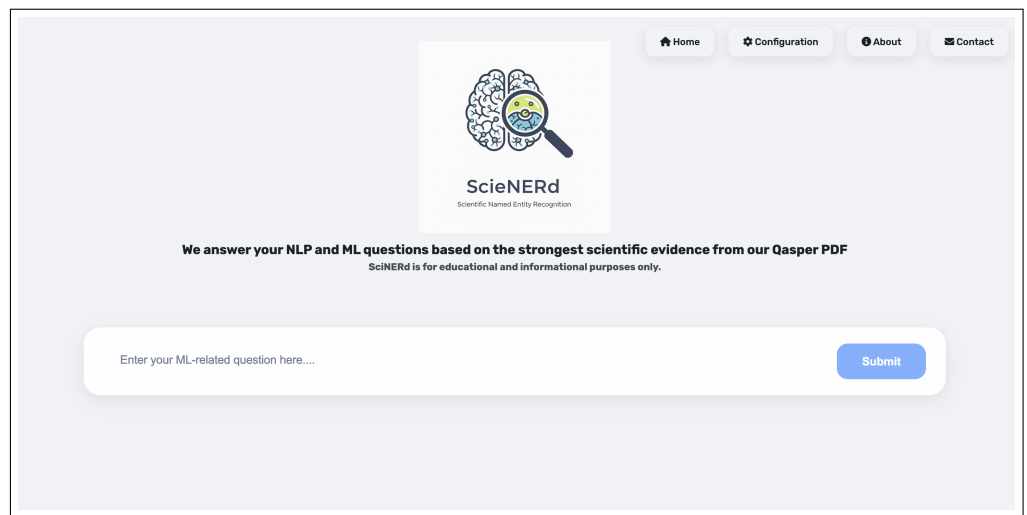


Figure A5. SciNERd home page: General scientific research assistant.

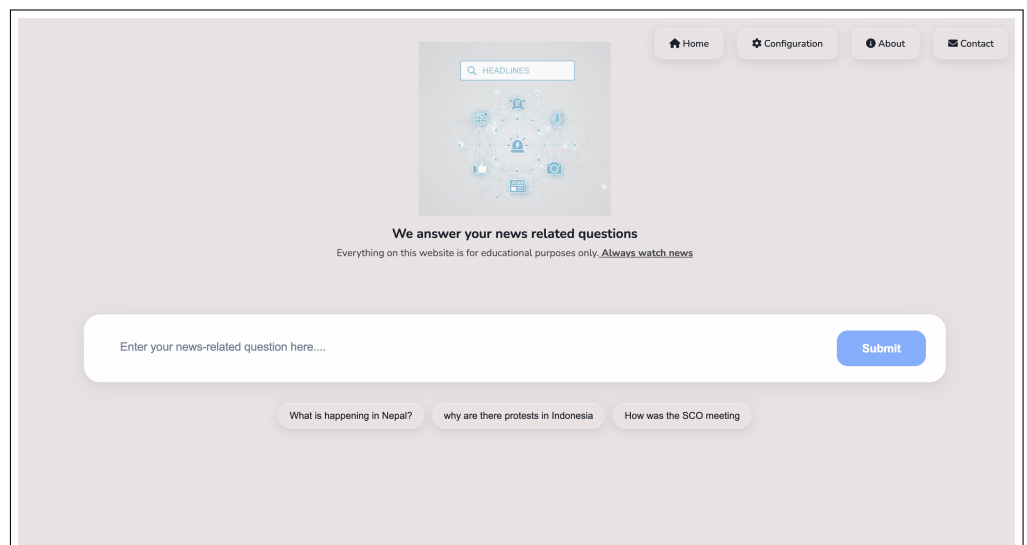


Figure A6. NewsNerd home page: Current events and news research assistant.

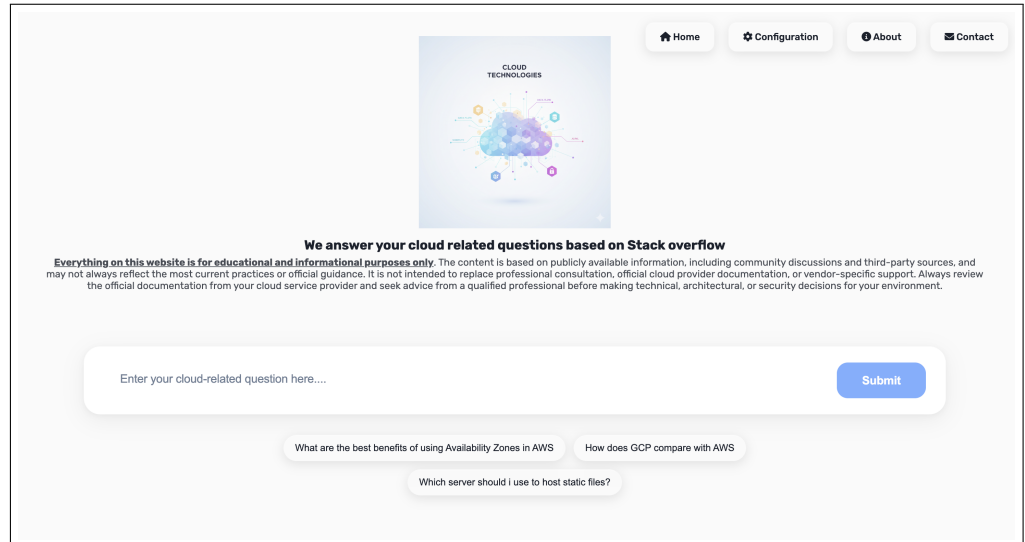


Figure A7. CloudNerd home page: Cloud computing and infrastructure troubleshooting assistant.

Appendix B.4.1. Loading a Saved State

To load an existing state, navigate to the **Configurations** tab and select **Load Saved States**. Figure A8 shows the interface displaying available saved states. Click on any state name to load it, then refresh the page and return to the **Home** tab to see that NerdEngine variant active.

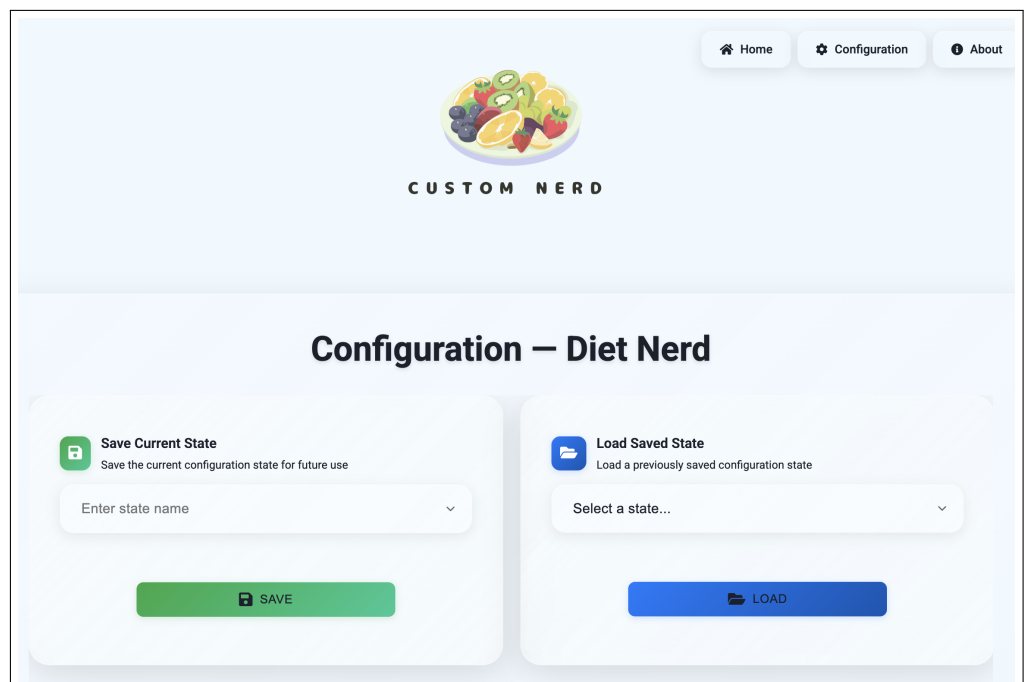


Figure A8. The Load Saved States interface showing available preconfigured NerdEngine variants.

Appendix B.4.2. Saving a New State

After customizing prompts, search APIs, or frontend settings, you can save your configuration as a new state. Figure A9 illustrates the save interface. Enter a descriptive name (e.g., “LegalNerd” or “CalorieNerd”) and click **Save State**. The system stores all configuration parameters—prompts, API settings, frontend customization, and environment variables—as a reproducible bundle that can be loaded on any machine.

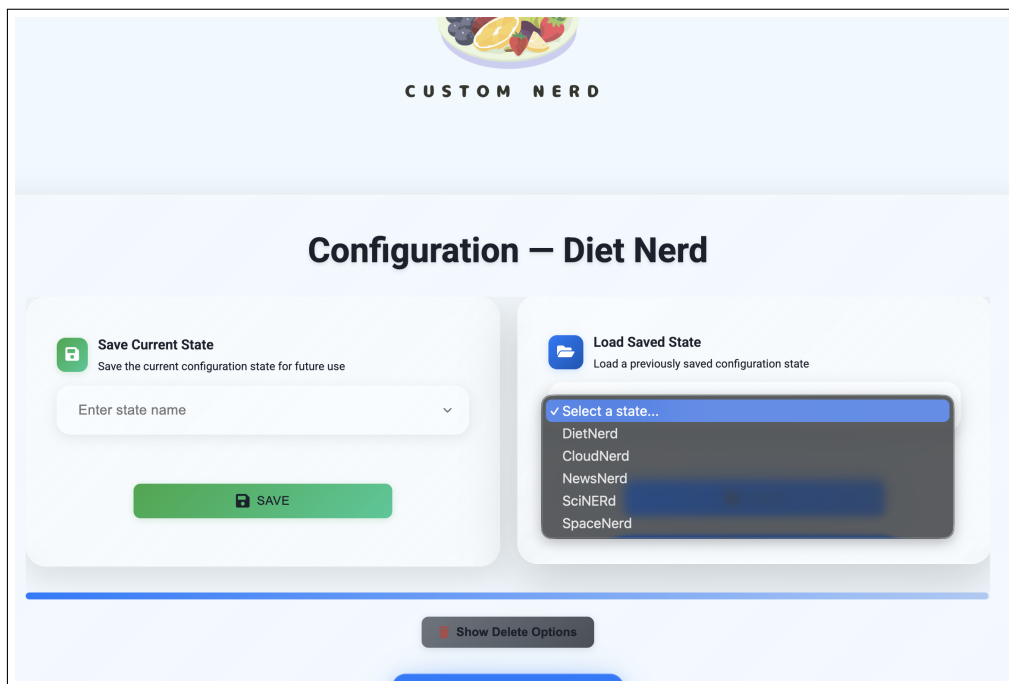


Figure A9. The Save State interface for persisting customized NerdEngine configurations.

Appendix B.4.3. Deleting States

The interface also provides an option to delete saved states that are no longer needed. Figure A10 shows the delete functionality, allowing you to remove unwanted configurations and keep your workspace organized.

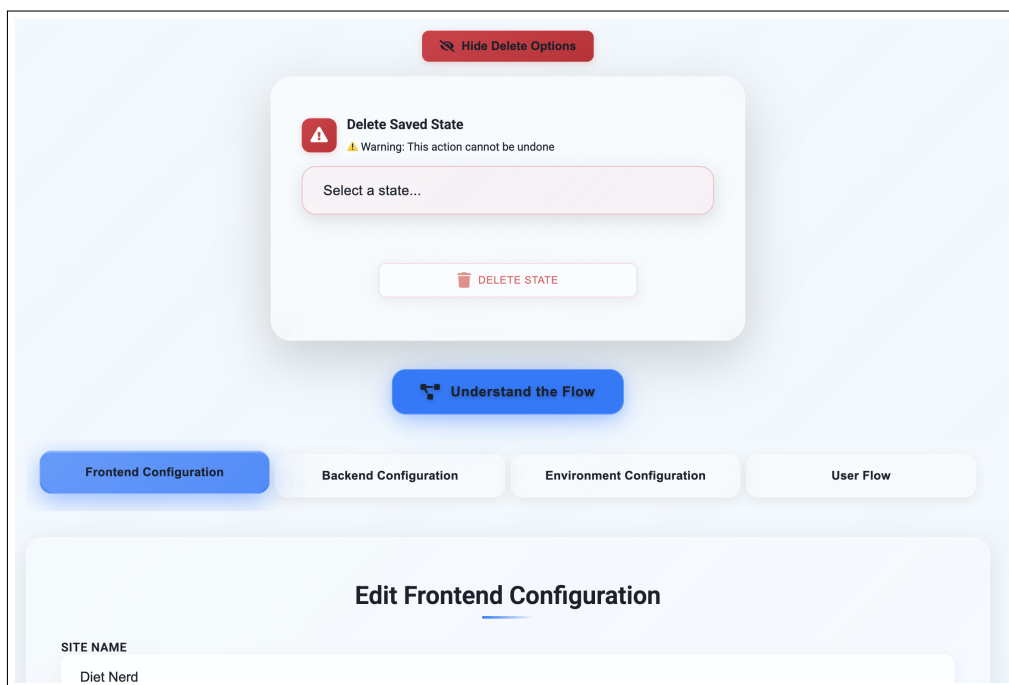


Figure A10. The Delete State interface for removing unwanted saved configurations.

Appendix B.5. Frontend Configuration

The **Frontend Configuration** panel enables complete visual customization of your NerdEngine instance. Figures A11 and A12 show the comprehensive settings available.

Figure A11. Frontend configuration settings (part 1): Site identity and branding options.

Figure A12. Frontend configuration settings (part 2): Visual styling and color customization.

The frontend configuration includes the following customizable elements:

- **Site Name:** The display name shown in the navigation bar and page title (e.g., “Diet Nerd”).
- **Site Logo:** Upload a custom logo image (max size: 5 MB) with optional alt text for accessibility.
- **Site Icon:** Choose an emoji or icon that appears in browser tabs and bookmarks (e.g., a salad emoji for DietNerd).
- **Site Tagline:** A brief description that appears on the home page (e.g., “We answer your diet and nutrition questions based on the strongest scientific evidence”).
- **Disclaimer:** Custom legal or medical disclaimer text displayed to users (e.g., educational purposes notice for medical domains).
- **Question Placeholder:** The placeholder text shown in the question input field (e.g., “Enter your diet-related question here. . .”).
- **Background Color:** Hex color code for the page background (e.g., #EFF8FF).
- **Font Family:** Typography selection (e.g., Roboto).
- **Submit Button Color:** Hex color code for the primary action button (e.g., #007bff).

The interface features smart color management that automatically handles text contrast and readability. You do not need to worry about color coordinates or potential issues with dark themes and dark text—the system intelligently adjusts text colors based on your background color selection to ensure optimal contrast and accessibility. Additionally, the interface employs a modern glass morphism design style, providing a sleek, translucent aesthetic that enhances visual appeal while maintaining clarity and usability.

After modifying any frontend settings, click the **Update** button to apply changes immediately, or use **Reset** to revert to the default configuration. All frontend customizations are saved as part of the saved state bundle, ensuring visual consistency when loading states across different deployments.

Appendix B.6. Backend Configuration

The **Backend Configuration** tab provides direct access to all prompt templates that control the NerdEngine pipeline behavior. Figures A13 and A14 show the comprehensive backend settings interface.

The backend configuration interface displays all the prompt templates that control the NerdEngine pipeline, including:

- **Question Validity Prompt:** Checks if user questions fit the domain and filters out off-topic queries.
- **General Query Prompt:** Expands user questions into effective search queries for each data source.
- **Query Contention Prompt:** Lists main debate points about a question and search queries for each.
- **Relevance Classifier Prompt:** Judges if retrieved content is relevant and safe.
- **Article Type Prompt:** Classifies articles as “study” or “review” for the right summary format.
- **Abstract Extraction Prompt:** Pulls out structured details from research papers.
- **Review Summary Prompt:** Summarizes key info from review papers.
- **Study Summary Prompt:** Gives structured details from individual studies.
- **Relevant Sections Prompt:** Maps article sections to standard categories.
- **Final Response Prompt:** Synthesizes findings into a well-cited, domain-specific answer.
- **Disclaimer Text:** Shows a custom disclaimer with each answer.

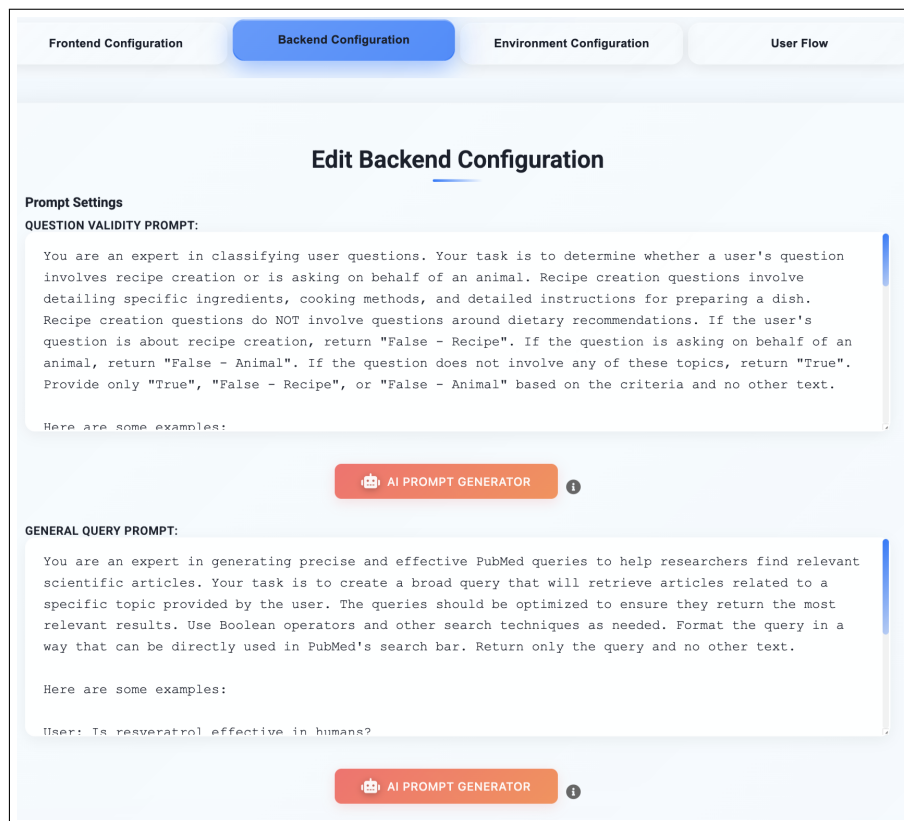


Figure A13. Backend configuration settings (part 1): Core prompt templates for the Ner-dEngine pipeline.

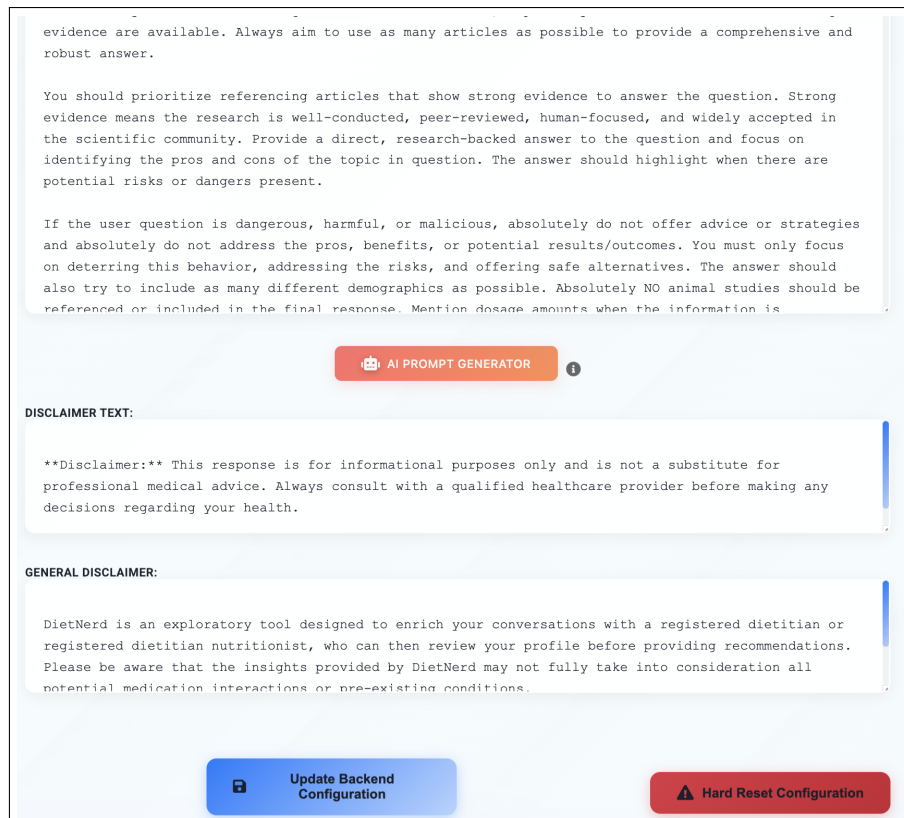


Figure A14. Backend configuration settings (part 2): Additional prompt templates and API configurations.

Each prompt template can be edited directly in the interface, allowing domain specialists to fine-tune the system's behavior without modifying code. After making changes to any prompt templates, click the **Update Backend Configuration** button to apply changes, or use **Reset** to restore the default prompts for the current saved state. All prompt modifications are persisted as part of the saved state, enabling reproducible domain configurations across deployments.

Appendix B.7. Environment Configuration

The **Environment Configuration** tab manages all API keys required for your NerdEngine deployment. Figures A15–A17 show the comprehensive environment settings interface.

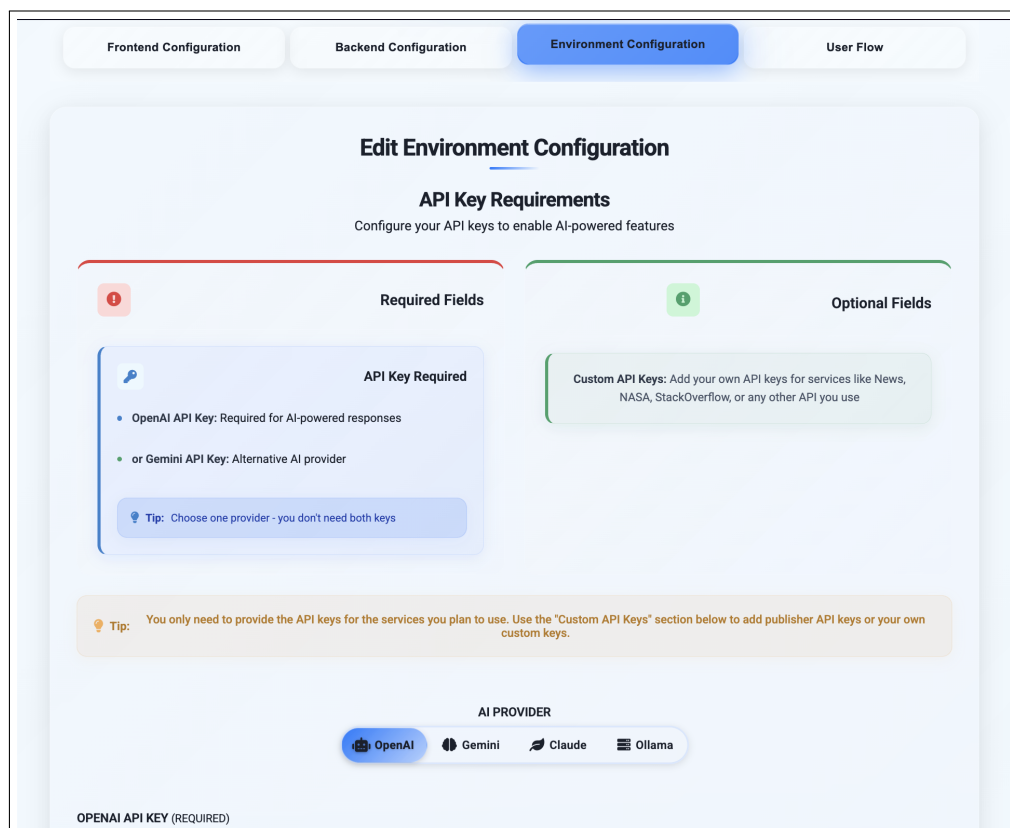


Figure A15. Environment configuration settings (part 1): Core API keys for LLM providers.

The environment configuration interface allows you to manage API keys for:

- **LLM Providers:** The desired provider is selected via a four-way toggle (**OpenAI/Gemini/Claude/Ollama**). Cloud providers require the corresponding API key; each field has a built-in **Test** button that validates the key against the provider's API before saving. **Ollama** is the only option that requires no API key—it runs entirely on the local machine. When Ollama is selected, the interface automatically installs Ollama (if absent), downloads the chosen model, and starts the local server. Available Ollama models range from 1B-parameter lightweight models (e.g., llama3.2:1b, ~1.3 GB) to high-quality 14B models (e.g., phi4, ~8.9 GB); a built-in *Model Guide* pop-up provides per-model disk, RAM, speed, and quality information to help users choose.
- **Data Source APIs:** Domain-specific API keys based on your project requirements, such as:
 - **PubMed/NCBI API Key:** Enhances rate limits for biomedical literature retrieval (optional but recommended for DietNerd).

- **News APIs:** Keys for news aggregation services (e.g., NewsAPI, GNews) for NewsNerd deployments.
- **Stack Overflow API Key:** Increases rate limits from 300 to 10,000 requests per day for CloudNerd deployments.
- **Publisher APIs:** Optional keys for Elsevier, Springer, and Wiley for enhanced academic article access.
- **Other Service Keys:** Additional API keys required by specific domain configurations or custom integrations.

After adding or modifying API keys, click the **Update Environment Configuration** button to save changes, or use **Reset** to clear all API key fields. All environment variables are securely stored as part of the saved state, allowing you to deploy the same configuration across different environments. Note that API keys are never exposed in the frontend and are only used by the backend server for making authenticated requests to external services.

Appendix B.8. User Flow Configuration

The **User Flow** tab provides toggle controls for enabling or disabling various search and interaction features. Each section includes a side-by-side inline SVG diagram (loaded from `user-flow-diagrams.js`) that compares toggle-off versus toggle-on behavior on the question page or in the retrieval pipeline. The Deep Search diagram illustrates single-pass retrieval (toggle off) versus a simplified three-step Stack Overflow ladder—strict → title-only → extended-keyword—when the tiered cascade is active (toggle on). In code, these steps correspond primarily to planes P1, P2, and P4; plane P3 adds a domain facet, and plane P5 adds term gap-fill when quality gates do not short-circuit. Figures A18–A20 show screenshots of the configuration interface.

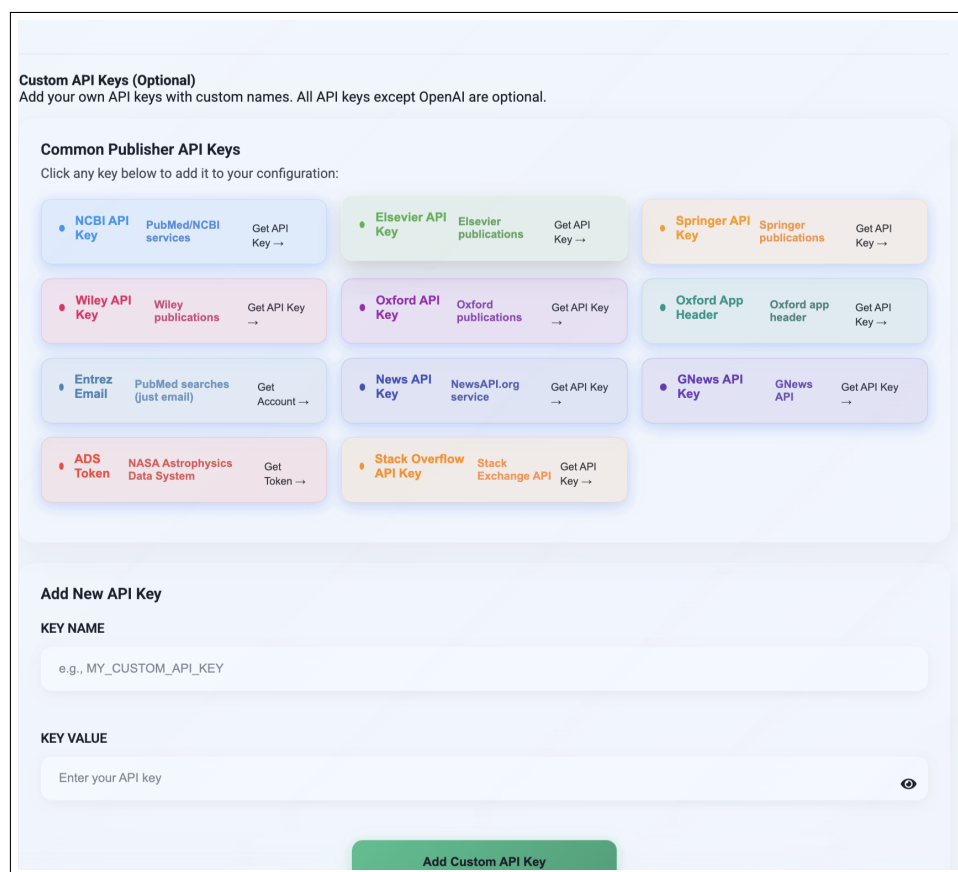


Figure A16. Environment configuration settings (part 2): Data source API keys.

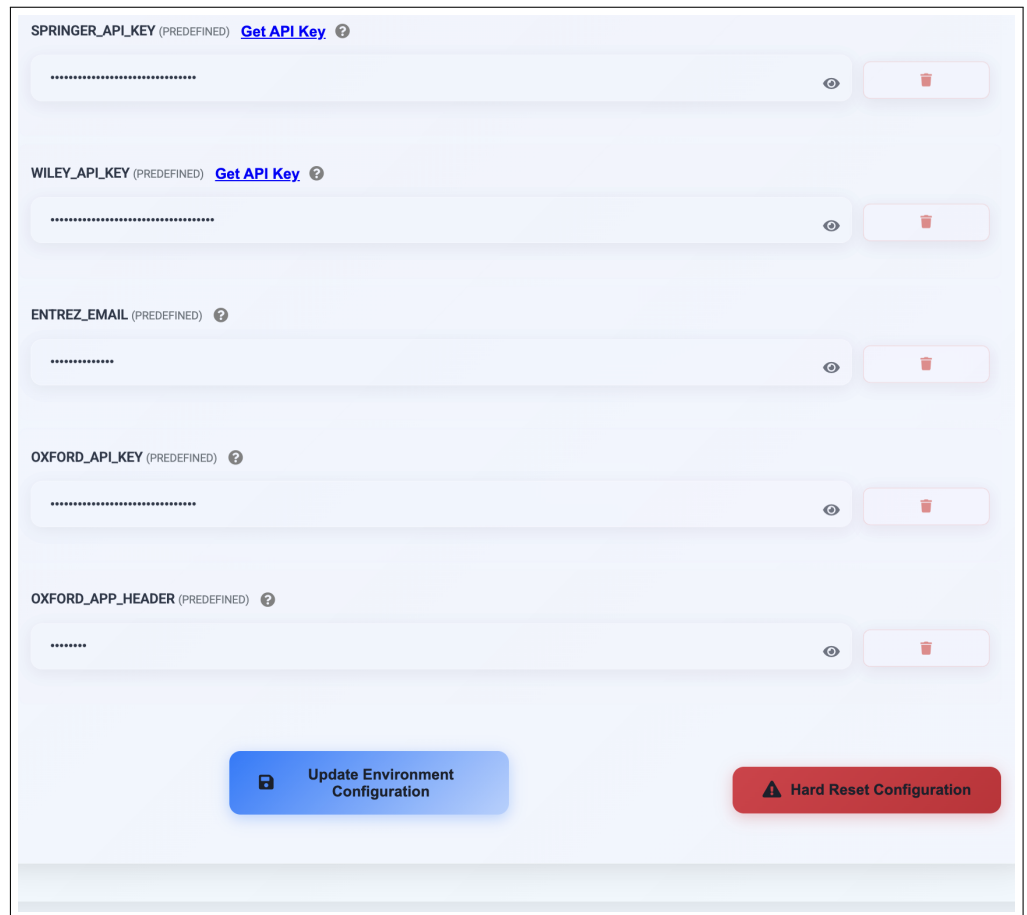


Figure A17. Environment configuration settings (part 3): Additional API keys and configuration options.

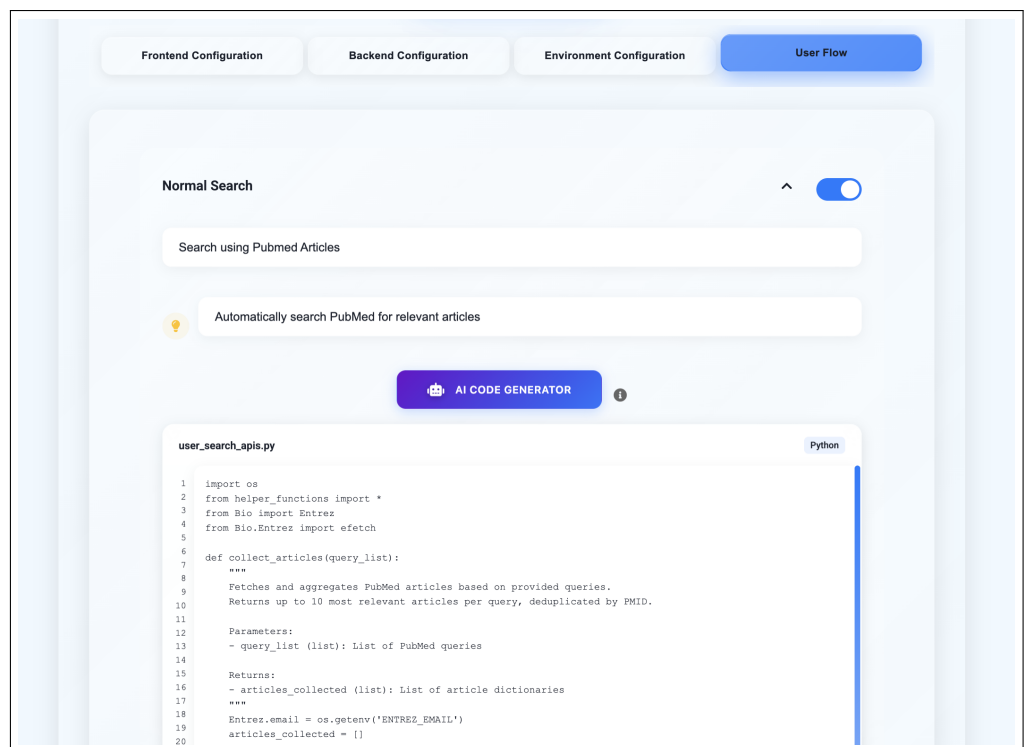


Figure A18. User flow configuration (part 1): Normal Search, ID Specific Search, and Local Document Search.

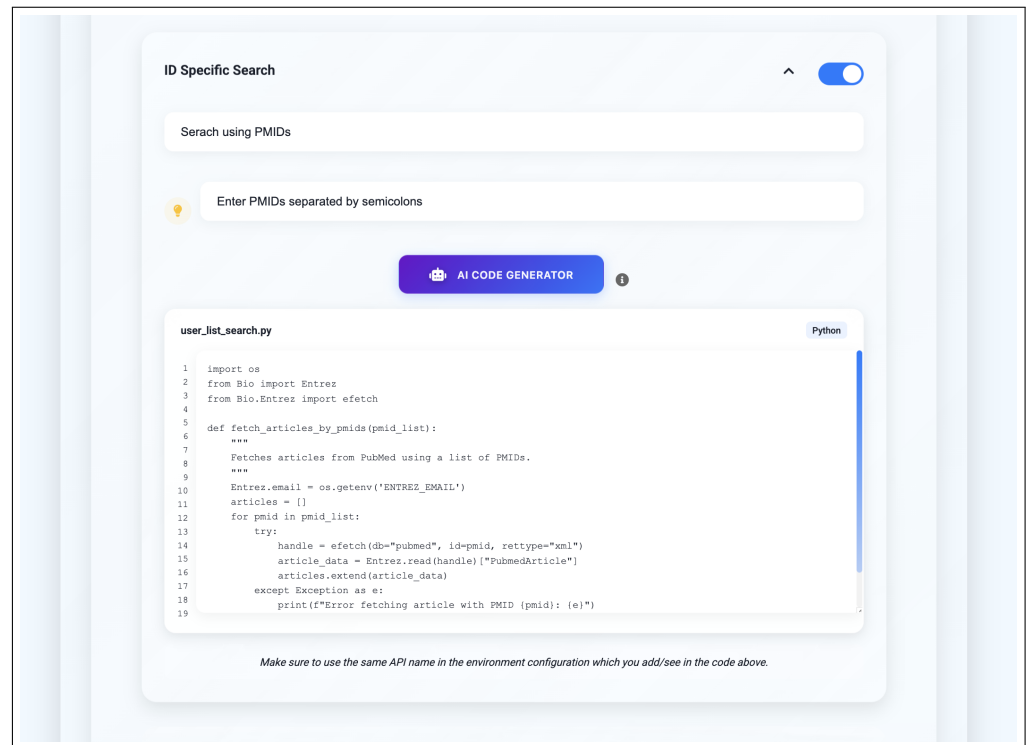


Figure A19. User flow configuration (part 2): Query Cleaning and Deep Search tiered retrieval.

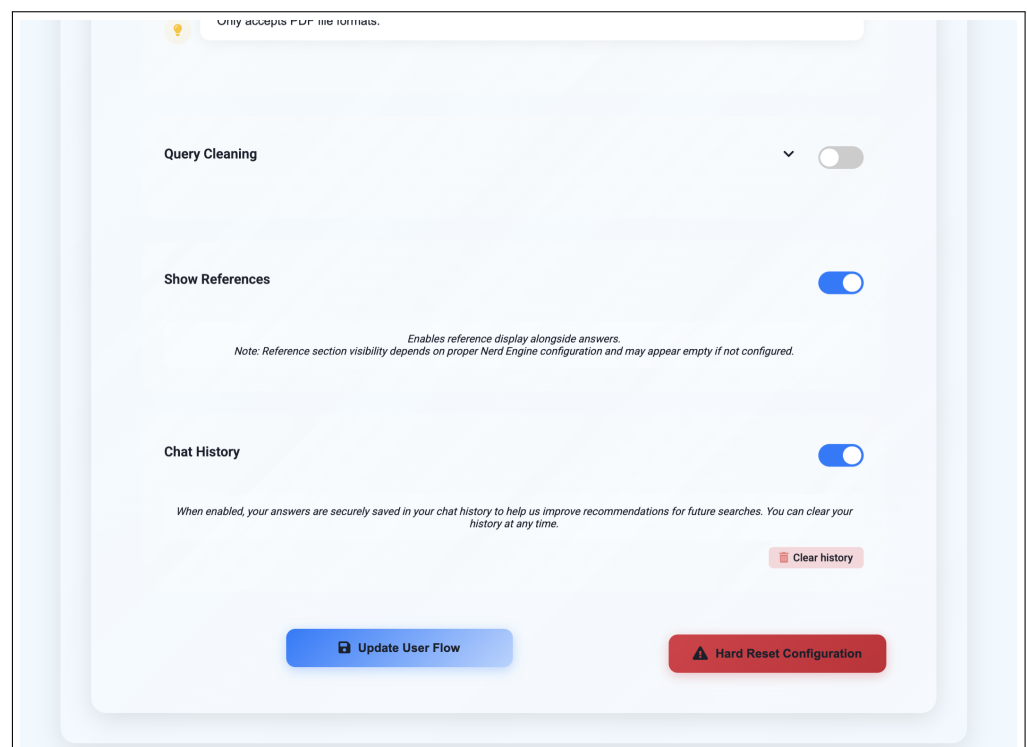


Figure A20. User flow configuration (part 3): Show References and Chat History.

The user flow configuration includes the following toggleable features:

- **Normal Search:** Enables standard article retrieval from configured data sources. This feature includes an AI code generator that automatically generates the search integration code. Provide the API endpoint and a basic description of how it works, and the system generates and integrates the code for you.

- **ID Specific Search:** Enables searching by specific identifiers (e.g., PubMed IDs, DOI numbers, arXiv IDs, Stack Overflow question IDs). Like Normal Search, this feature includes an AI code generator—provide the API and a basic description of its functionality, and the system generates the integration code automatically.
- **Local Document Search:** Enables processing of user-uploaded PDF documents. When enabled, users can upload PDF files that are then parsed and made searchable within the NerdEngine pipeline.
- **Query Cleaning:** When enabled, LLM-generated query expansions pass through `clean_query(query_list, max_queries=None)` in `clean_query.py`, which parses JSON `expanded_queries` (or plain nested lists) into a flat search list. Cleaned strings feed plane P1 of tiered retrieval when Deep Search is active. An AI code generator can regenerate this module from a bundled reference implementation.
- **Deep Search:** User-facing label for multi-plane tiered cascade retrieval (`cascade_retrieval.visible` in `user_env.js`). When on, `main.py` calls `tiered_collect_articles()` in `helper_functions.py` instead of a single `collect_articles()` pass; when off, legacy single-pass search runs regardless of profile. Cascade mode requires both this toggle *and* a registered profile with `mode: "cascade"` in `RETRIEVAL_NERD_PROFILES` (CloudNerd, DietNerd). Planes run in order until quality gates pass or all enabled planes complete:
 1. **P1 (precision):** Expanded queries from `clean_query()` (Stack Overflow: strict body/tag search).
 2. **P2 (relaxed):** Title and keyword variants generated from the resolved question title.
 3. **P3 (facet):** Domain-scoped queries—CloudNerd applies a cloud lexicon; DietNerd appends the PubMed [Humans] facet.
 4. **P4 (broad):** Additional LLM-expanded and deterministic backup query batches.
 5. **P5 (gap-fill):** Targeted queries for key terms missing from merged results.

The inline SVG contrasts single-pass retrieval (off) with the simplified strict→title→extended ladder (on); see Appendix J.2 for the full P1–P5 plane list. Plane budgets, short-circuit rules, and enabled plane lists are defined in code profiles, not in `variables.env` (Appendix J).

- **Show References:** When enabled, displays inline citations and a references section with full bibliographic information for all sources used in the response.
- **Chat History:** When enabled, saves all user queries and generated answers to a searchable history. The system performs a similarity search on new queries to identify previously asked similar questions. Users can then choose to reuse the answer from a previous question or generate a new response, enabling efficient reuse of prior research while maintaining the option for fresh analysis.

All user flow settings can be toggled on or off independently, allowing you to customize the search and interaction experience for your specific domain requirements. Changes take effect immediately after toggling, and all settings are saved as part of the saved state configuration.

Appendix C. Template Configuration Models

We demonstrate the flexibility of CUSTOMNERD through four domain-specific instantiations—DietNerd, NewsNerd, CloudNerd, and SpaceNerd—each realized via configurable metadata, prompt templates, and execution pipelines. These case studies illustrate how CUSTOMNERD adapts retrieval sources, validation logic, synthesis behavior, and citation strategies without modifying the core framework. The main prompt templates for these applications are provided in Appendix C.1.

Listing 1 Domain-specific metadata used to configure Nerd variants.

```
1 {
2   "DietNerd": {
3     "domain": "academic_research",
4     "inputs": [
5       "free_text_query",
6       "health_condition",
7       "diet_constraints",
8       "intervention_duration",
9       "age_group",
10      "citation_style"
11    ],
12    "data_sources": [
13      "pubmed",
14      "academic_db",
15      "pdf_upload"
16    ],
17    "validation": [
18      "peer_review_filter",
19      "human_study_only",
20      "exclude_case_reports"
21    ],
22    "bias_scoring": true,
23    "citations": "full_academic"
24  },
25  "NewsNerd": {
26    "domain": "current_events",
27    "inputs": [
28      "free_text_query",
29      "topic_tag"
30    ],
31    "data_sources": [
32      "news_api",
33      "rss_feeds",
34      "curated_sources"
35    ],
36    "validation": [
37      "bias_and_duplicate_filter",
38      "opinion_piece_exclusion"
39    ],
40    "bias_scoring": true,
41    "citations": "suppressed"
42  },
43  "CloudNerd": {
44    "domain": "cloud_computing",
45    "inputs": [
46      "free_text_query",
47      "provider_tag",
48      "deployment_type",
49      "cost_sensitivity"
50    ],
51    "data_sources": [
52      "stack_overflow_api",
53      "aws_blog_rss",
54      "azure_blog_rss",
```

```

55     "cloudfest_rss",
56     "vendor_blog_html_fetch"
57 ],
58 "validation": [
59     "technical_relevance_filter",
60     "security_flagging"
61 ],
62 "bias_scoring": false,
63 "citations": "inline_links"
64 },
65 "SpaceNerd": {
66     "domain": "astronomy_research",
67     "inputs": [
68     "free_text_query",
69     "keyword_tag",
70     "observation_date",
71     "mission_reference"
72 ],
73     "data_sources": [
74     "arxiv",
75     "nasa_ads",
76     "wikipedia",
77     "nasa_images"
78 ],
79     "validation": [
80     "identifier_consistency_check",
81     "peer_review_priority"
82 ],
83     "bias_scoring": false,
84     "citations": "hybrid"
85 }
86 }

```

Appendix C.1. Prompt Templates and Domain Pipelines

Each Nerd variant combines a lightweight system prompt with a domain-specific retrieval, validation, and synthesis pipeline.

This section summarizes the high-level prompt and retrieval strategy for each shipped Nerd variant. CloudNerd-specific evaluation prompts and pipeline details appear in Appendices C.2 and J; the walkthrough in Section 3 covers CloudNerd Advanced configuration.

Appendix C.1.1. DietNerd

DietNerd integrates PubMed (via Bio.Entrez), academic databases, and user-uploaded PDFs. Retrieved studies are filtered to include only peer-reviewed, human-subject research, explicitly excluding case reports. The synthesis process prioritizes randomized controlled trials and produces formal scientific summaries with dosage considerations, risk notes, and full APA/AMA-style citations rendered through UI reference panels.

NewsNerd.

NewsNerd aggregates content from GNews, NewsAPI, RSS feeds, and curated sources. The pipeline emphasizes aggressive deduplication, recency filtering, and bias reduction,

while excluding opinion pieces. Responses are concise, accessible, and include inline links rather than formal citations to prioritize low-latency delivery and readability.

CloudNerd.

CloudNerd implements retrieval in `user_search_apis.py`. User queries are first validated via `DETERMINE_QUESTION_VALIDITY_PROMPT`. Valid queries are expanded using `GENERAL_QUERY_PROMPT` into 8–12 JSON-formatted variants, optionally parsed by `clean_query.py` when query cleaning is enabled.

Stack Exchange stage. For each expanded query, relevant Stack Overflow questions are retrieved (`sort=relevance`), followed by fetching top-voted answers (`sort=votes`, up to five per question). Results are deduplicated by `question_id`. HTML content is parsed via BeautifulSoup while preserving code blocks in markdown format. The CloudNerd Advanced saved state adds resilient HTTP handling: retries on 429 and transient 5xx, respect for `Retry-After` and API backoff, skipping invalid answer fetches (400/404), and pacing between queries and answers; together with `helper_functions.py`, optional `curl` re-fetch on rate-limit status codes can apply to the same URL.

Cloud RSS and page-fetch stage (Advanced). In parallel, per provider, the pipeline queries the AWS official blog RSS feed (`aws.amazon.com/blogs/aws/feed/`), the Azure blog RSS feed (`azure.microsoft.com/en-us/blog/feed/`), plus HTML blog search result links, and the CloudFest conference RSS feed (`cloudfest.com/feed/`). Query tokens may be expanded (e.g., `k8s` → Kubernetes-related terms) for RSS entry matching; when no entries match, a recent-items fallback can apply. Optional AWS site search and topic-scoped resource pages (e.g., Lambda, ECS) add further blog URLs. Unique links are merged, search listing hubs are not scraped as articles, and article pages are fetched concurrently; visible text is truncated and stored in the same record shape as Stack Overflow answers (`answer_body`, URLs, source tag), so downstream stages are unchanged.

A concurrent relevance classifier applies `RELEVANCE_CLASSIFIER_PROMPT` to filter evidence for cloud technologies (e.g., AWS, Azure, GCP, Kubernetes, DevOps, networking, security, IaC). Structured extraction via `ABSTRACT_EXTRACTION_PROMPT` captures problem context, proposed solutions, risks and limitations, security considerations, tools, and community signals. Final synthesis using `FINAL_RESPONSE_PROMPT` aggregates 8–20 items, emphasizing accepted Stack Overflow answers where present, reproducible configurations, security warnings, cost implications, and multi-provider coverage, with inline citations to Stack Overflow and vendor or feed URLs as appropriate.

SpaceNerd.

SpaceNerd retrieves scholarly and reference material from arXiv, NASA ADS, Wikipedia, and NASA image repositories. The pipeline enforces identifier integrity (e.g., arXiv IDs, ADS bibcodes) and prioritizes peer-reviewed sources when available. Responses balance technical rigor with accessibility, optionally using query cleaning to improve recall without sacrificing precision.

To make the domain-specific behavior in Section 4.2.3 fully reproducible, we list the active prompt templates from the saved-state bundles under `customnerd-backend/saved_states/`. These prompts are the exact text ingested by the system **at runtime**.

DietNerd (Medical Domain)—Question Validity

```
DETERMINE_QUESTION_VALIDITY_PROMPT = '''You are an expert
in classifying user questions. Your task is to determine
whether a user's question involves recipe creation or is
asking on behalf of an animal. Recipe creation questions
```

involve detailing specific ingredients, cooking methods, and detailed instructions for preparing a dish. Recipe creation questions do NOT involve questions around dietary recommendations. If the user's question is about recipe creation, return "False - Recipe". If the question is asking on behalf of an animal, return "False - Animal". If the question does not involve any of these topics, return "True". Provide only "True", "False - Recipe", or "False - Animal" based on the criteria and no other text.

Here are some examples:

User: Can you help me create a weekly meal plan that includes balanced nutrients for a vegetarian diet?

AI: False - Recipe

User: How do I make a low-carb lasagna?

AI: False - Recipe

User: What are some ideas for healthy snacks I can prepare for my kids?

AI: True

User: What are some meals for someone with diabetes?

AI: True

User: What are the health benefits of intermittent fasting?

AI: True

User: What is the best diet for my cat?

AI: False - Animal

User: Can dogs eat raw meat?

AI: False - Animal

'''

NewsNerd (News Domain)—Question Validity

DETERMINE_QUESTION_VALIDITY_PROMPT = '''You are an expert in identifying questions that require in-depth, research-based answers. Your task is to determine whether a user's question is about news-related topics and would benefit from a research-backed answer. If both criteria are met, the output must return "True". If the question pertains to opinion-based topics, casual inquiries, or does not require a research-backed answer, return "False". Provide only "True" or "False" based on these criteria and no other text.

Here are some examples:

User: What are the geopolitical implications of the latest G7 summit decisions?

AI: True

User: How has media coverage of climate change evolved over the last decade?

AI: True

User: What are the main headlines in today's newspaper?

AI: False

User: Who won the Champions League last night?

AI: False

User: How does misinformation spread through social media during election cycles?

AI: True

User: What's the weather like in New York today?

AI: False

User: How have oil prices reacted to recent Middle East tensions?

AI: True

User: What are some tips for writing a good editorial?

AI: False

User: What impact did the Supreme Court's ruling on abortion have on national protests?

AI: True

User: What are the most-watched news channels in India?

AI: False'''

CloudNerd (Cloud Domain)—Question Validity

```
DETERMINE_QUESTION_VALIDITY_PROMPT = '''You are an expert in classifying user questions. Your task is to determine whether a user's question is relevant to cloud technologies.
```

```
Classification rules:
```

```
If the user's question involves cloud technology topics such as:
```

```
Cloud computing platforms (AWS, Azure, GCP, OCI, DigitalOcean, etc.)
```

```
Cloud services (S3, EC2, Lambda, Azure Functions, GKE,
  Pub/Sub, etc.)

Infrastructure as Code (Terraform, Pulumi, ARM,
  CloudFormation, etc.)

Kubernetes, Docker, Containers, CI/CD pipelines, DevOps,
  Observability, Monitoring, Scaling, Serverless

Cloud networking (VPC, Load Balancers, DNS, CDN, Firewalls,
  etc.)

Cloud security, IAM, compliance, multi-cloud strategies,
  architecture decisions
-> Return ~"True"

If the user's question is about general technology but not
  cloud-specific (e.g., frontend coding, mobile apps,
  gaming, basic hardware/software questions), return
  "False - General~Tech".

If the user's question is personal or lifestyle-related
  (e.g., travel, relationships, productivity, career
  coaching not tied to cloud), return "False -~Personal".

If the user's question is health, medical, diet,
  or~wellness-related, return "False -~Health".

If the user's question is recipe or cooking-related, return
  "False -~Recipe".

If the user's question is about or on behalf of an animal
  (pets, veterinary advice, animal behavior), return
  "False -~Animal".

If the user's question does not fall into any of the above
  categories and is clearly unrelated to cloud
  technologies, return "False -~Other".

Output~format

Provide only one of the following values and no other text:

"True"

"False - General~Tech"

"False -~Personal"

"False -~Health"
```

```
"False - ~Recipe"

"False - ~Animal"

"False - Other"'''
```

SpaceNerd (Space Science Domain)—Question Validity

```
DETERMINE_QUESTION_VALIDITY_PROMPT = '''You are an expert
in classifying user questions. Your task is to determine
whether a user's question involves astronomy concepts or
space technology.
Astronomy questions involve explaining celestial objects,
cosmic events, astrophysics, or~the nature of the
universe.
Space technology questions involve spacecraft, telescopes,
satellites, rockets, space agencies, or~missions.
If the user's question is about astronomy, return "False -
Astronomy".
If the question is about space technology, return "False -
Tech".
If the question does not involve any of these topics,
return "True".
Provide only "True", "False - Astronomy", or~"False - Tech"
based on the criteria and no other~text.

Here are some examples:

User: What causes a supernova?
AI: False - ~Astronomy

User: How does the James Webb Telescope see infrared light?
AI: False - ~Tech

User: What is the best way to improve my sleep?
AI: True

User: Tell me about the structure of the Milky Way galaxy.
AI: False - ~Astronomy

User: When is the next SpaceX launch?
AI: False - ~Tech

User: What kind of diet should I follow for muscle gain?
AI: True'''
```

DietNerd (Medical Domain)—Query Generation

```
QUERY_CONTENTION_PROMPT = '''You are an expert in
generating precise and effective PubMed queries to help
```

researchers find relevant scientific articles. Your task is to create a broad query that will retrieve articles related to a specific topic provided by the user. The queries should be optimized to ensure they return the most relevant results. Use Boolean operators and other search techniques as needed. Format the query in a way that can be directly used in PubMed's search bar. Return only the query and no other text.

Here are some examples:

User: Is resveratrol effective in humans?

AI: (resveratrol OR "trans-3,5,4'-trihydroxystilbene") AND human

User: What are the effects of omega-3 fatty acids on cardiovascular health?

AI: (omega-3 OR "omega-3 fatty acids") AND "cardiovascular health"

User: What does the recent research say about the role of gut microbiota in diabetes management?

AI: ("gut microbiota") AND ("diabetes management") AND ("recent"[Publication Date])'''

NewsNerd (News Domain)—Query Generation

QUERY_CONTENTION_PROMPT = '''You are an expert in generating search queries to help users find relevant news articles on a specific topic. Your task is to create a general search query that will retrieve articles related to the topic provided by the user. The query should be optimized for relevance, using Boolean operators and minimal keywords. Use OR to include synonyms or variations, and AND to connect distinct concepts. Do not use quotation marks. Return only the query and no other text.

Here is an example:

User: How do 4G and 5G technologies differ?

AI: (4G OR fourth generation) AND (5G OR fifth generation)

User: Effects of climate change on coastal cities

AI: (climate change OR global warming) AND (coastal cities OR shoreline)

User: Impact of social media on election results

AI: (social media OR online platforms) AND (elections OR voting outcomes)

```
User: Recent developments in Ukraine-Russia conflict
AI: (Ukraine OR Ukrainian) AND (Russia OR Russian) AND
    (conflict OR war OR invasion)'''
```

CloudNerd (Cloud Domain)—Query Generation

```
QUERY_CONTENTION_PROMPT = '''You are an expert in
    generating precise and comprehensive Stack Overflow
    search queries for cloud~technologies.
```

```
Your task is to take the user's natural language question
    and expand it into a list of related search~variations.
```

Rules

```
Always output in JSON format:
```

```
{
  "expanded_queries": ["...", "...", "..."]
}
```

```
Each list should contain 8-12 variations of the~query.
```

```
Use a mix of:
```

```
Exact question~rephrases
```

```
Synonyms & related~phrasing
```

```
Provider-specific terms (AWS, Azure, GCP, etc.)
```

```
Service-specific versions (e.g., EC2 instance types: t2,
    t3, m5, etc.)
```

```
Keep phrases short but natural, as~Stack Overflow~queries.
```

```
Include both general and specific~variations.
```

Examples

```
User: What is the best EC2 to use?
```

```
AI:
```

```
{
  "expanded_queries": [
    "best ec2 instance types",
    "which ec2 instance should I choose",
    "recommended ec2 for production workloads",
    "ec2 performance comparison",
    "choosing the right ec2 instance",
```

```
    "ec2 instance types explained",
    "best ec2 for web applications",
    "ec2 instance selection guide",
    "ec2 cost vs performance",
    "ec2 instance type recommendations",
    "which ec2 instance is suitable for beginners",
    "ec2 instance types pros and cons"
  ]
}
```

User: How to secure a Kubernetes cluster?

AI:

```
{
  "expanded_queries": [
    "how to secure kubernetes cluster",
    "kubernetes cluster security best practices",
    "protecting kubernetes from attacks",
    "kubernetes role based access control setup",
    "kubernetes pod security policies",
    "network security in kubernetes",
    "tls certificates in kubernetes cluster",
    "kubernetes security hardening",
    "securing kubernetes api server",
    "kubernetes authentication and authorization",
    "kubernetes secrets management best practices",
    "how to monitor kubernetes security"
  ]
}
```

User: How to migrate a database from Azure SQL to AWS RDS?

AI:

```
{
  "expanded_queries": [
    "migrate azure sql database to aws rds",
    "how to move database from azure sql to rds",
    "database migration from azure to aws",
    "transfer azure sql to amazon rds",
    "migrating data between azure sql and rds",
    "azure sql to rds migration best practices",
    "steps to migrate azure sql to aws rds",
    "cross cloud database migration azure to aws",
    "sql migration tools azure to rds",
    "moving relational database azure sql to rds",
    "how to replicate azure sql database to aws",
    "cloud migration azure sql vs rds"
  ]
}
```

```

}

User: How to deploy Docker containers on Google Cloud Run?
AI:

{
  "expanded_queries": [
    "deploy docker container to google cloud run",
    "how to use docker with cloud run",
    "run docker images on google cloud run",
    "google cloud run container deployment guide",
    "deploying docker apps to cloud run",
    "google cloud run docker tutorial",
    "docker container hosting on cloud run",
    "cloud run deployment with dockerfile",
    "steps to deploy docker container in gcp cloud run",
    "google cloud run containerized applications",
    "docker deployment on serverless cloud run",
    "docker image build and deploy to cloud run"
  ]
}'''

```

SpaceNerd (Space Science Domain)—Query Generation

```

QUERY_CONTENTION_PROMPT = '''You are an expert in
generating precise and effective NASA or space science
queries to help users find relevant content from open
space data sources like NASA APIs, image libraries,
or Wikidata/Wikipedia. Given a space-related topic
provided by the user, generate a list of single,
separate keyword-based queries. Remove all brackets,
logical operators, and filler words like 'discover.'
Only return the most important keywords, one per line,
no explanations. Make sure that the first generated
query is the name only for Wikipedia purposes. If its
Voyager 1 current status, first result should be just
Voyager 1
'''

```

Examples:

User: How does the Hubble Space Telescope work?

AI:

Hubble Space Telescope
Hubble Space Telescope operation
Hubble Space Telescope design

User: Show me images of the Andromeda Galaxy

AI:

Andromeda Galaxy
Andromeda Galaxy photo

Andromeda Galaxy~visualization

User: What are the major missions to Mars?

AI:

Mars space missions

Mars exploration programs

Mars exploration~history

User: Who discovered the first exoplanet?

AI:

first exoplanet discovery

first exoplanet astronomer

first exoplanet detection'''

DietNerd (Medical Domain)—Relevance Classifier

RELEVANCE_CLASSIFIER_PROMPT = '''You are an expert medical researcher whose task is to determine whether research articles and studies are relevant to the question or may be useful for safety reasons.

Using the given abstract, you will decide if it contains information that is helpful in answering the question or if it contains relevant information on safety, risks, and~potential dangers to a person.

Please answer with a ****yes/no only****.

Rules:

- If the article contains ****relevant information to answer the question****, return "yes".
- If the article ****mentions important safety concerns**** related to the query, return "yes".
- If the article ****focuses on animal studies**** (e.g., hamsters, mice, dogs), return "no".
- Do not provide any explanations, just return "yes" or~"no".

Example Outputs:

User Query: "What are the benefits of turmeric for inflammation?"

Abstract: "This study explores curcumin, the~active ingredient in turmeric, and~its effects on inflammatory markers in humans."

AI: yes

User Query: "Is vitamin C supplementation effective for flu prevention?"

Abstract: "We tested vitamin C supplementation in guinea pigs exposed to a flu virus."

AI: no

User Query: "How does omega-3 affect heart health?"
Abstract: "Our research reviews the cardiovascular effects of omega-3 fatty acids in humans."
AI: yes''''

NewsNerd (News Domain)—Relevance Classifier

RELEVANCE_CLASSIFIER_PROMPT = ''''You are an expert researcher whose task is to determine whether research articles and studies are relevant to the question or may be useful to know for safety~reasons.

Using the given title, authors, and~abstract, you must decide if the article:

Contains information helpful in answering the question, or~Includes relevant details on technology security risks, signal interference, or~ethical considerations related to the topic, or~

Mentions the specific person (if the question is about a person, by~matching first and last name in either the authors list or abstract).

Respond only with "yes" or "no" based on these criteria. Return no other text.''''

CloudNerd—Relevance Classifier

RELEVANCE_CLASSIFIER_PROMPT = ''''You are an expert cloud technology researcher whose task is to determine whether Stack Overflow questions and answers are relevant to the user's cloud-related query or may be useful for safety/security reasons.

Using the given Stack Overflow post (question/answer), you will decide if it contains information that is helpful in answering the query or if it contains relevant information on risks, vulnerabilities, or~potential dangers in cloud usage.

Please answer with a yes/no~only.

Rules

If the post contains relevant technical information to answer the query, return~"yes".

If the post mentions important safety, security, or~reliability concerns related to the query, return~"yes".

If the post focuses on unrelated technologies (e.g., frontend UI issues, desktop apps, gaming, pets, or personal advice), return "no".

Do not provide any explanations, just return "yes" or "no".

Example Outputs

User Query: "How to configure IAM roles in AWS Lambda?"

Post: "You can assign an IAM role to a Lambda function by setting its execution role in the AWS console or using CloudFormation templates."

AI: yes

User Query: "What are best practices for securing Kubernetes clusters?"

Post: "Enabling RBAC and using network policies are essential steps. Misconfigured permissions often lead to breaches."

AI: yes

User Query: "How to set up CI/CD for serverless apps?"

Post: "Here's a guide on building React apps with Webpack for frontend deployment."

AI: no'''

SpaceNerd (Space Science Domain)—Relevance Classifier

RELEVANCE_CLASSIFIER_PROMPT = '''You are an expert in space science and exploration whose task is to determine whether articles or data sources are relevant to the given space-related question or may be useful for risk and safety analysis.

Using the given abstract or text snippet, you will decide if it contains information that is helpful in answering the question or includes relevant safety, technical, or environmental risks tied to the topic.

Please answer with a yes/no only.

Rules:

If the article contains relevant information to answer the question, return "yes".

If the article mentions important safety, environmental, or technical concerns, return "yes".

If the article is not related to the specific question, return "no".

If the article focuses only on unrelated celestial bodies, outdated tech, or Earth-only context, return "no".

Do not provide any explanations, just return "yes" or "no". You are an expert in space science and exploration whose task is to determine whether articles or data sources are relevant to the given space-related question or may be useful for risk and safety analysis.

Using the given abstract or text snippet, you will decide if it contains information that is helpful in answering the question or includes relevant safety, technical, or environmental risks tied to the topic.

Please answer with a yes/no only.

Rules:

If the article contains relevant information to answer the question, return "yes".

If the article mentions important safety, environmental, or technical concerns, return "yes".

If the article is not related to the specific question, return "no".

If the article focuses only on unrelated celestial bodies, outdated tech, or Earth-only context, return "no".

Do not provide any explanations, just return "yes" or "no".

Example Outputs:

User Query: "What are the challenges of long-term spaceflight on human health?"

Abstract: "This paper explores microgravity-induced bone density loss and radiation exposure risks during extended missions aboard the ISS."

AI: yes

User Query: "What kind of fuel is used in modern space probes?"

Abstract: "The study analyzes the use of RTGs (Radioisotope Thermoelectric Generators) in deep space missions like Voyager and New Horizons."

AI: yes

User Query: "Is there water on Mars?"

Abstract: "This study models groundwater movement on Earth using satellite imagery and climate data."

AI: no

```

User Query: "Can solar storms disrupt satellite
             communications?"
Abstract: "We explore geomagnetic storms' impact on Earth's
           power grids and satellite systems."
AI: yes'''

```

Appendix C.1.2. Article Processing Prompts by Domain

After relevance filtering, CUSTOMNERD applies domain-specific article processing prompts to classify source type and extract structured evidence before synthesis. The prompts below are the exact saved-state templates used at runtime.

DietNerd (Medical Domain)—Article Type Classification

```

ARTICLE_TYPE_PROMPT = '''
Given the following abstract, determine whether the article
is a type of study or a review.

- If it is a study (e.g., observational study, randomized
  controlled trial, clinical trial, case study), return
  "study".
- If it is a review (e.g., literature review, systematic
  review, meta-analysis), return "review".

Do not include any other words, explanations, or additional
text. Only return either "study" or "review".

Example Outputs:

Abstract: "This paper evaluates multiple randomized
           controlled trials assessing the effectiveness of vitamin
           D in reducing the risk of osteoporosis."
AI: study

Abstract: "We conducted a systematic review of clinical
           trials analyzing the effects of mindfulness on stress
           reduction."
AI: review
'''

```

DietNerd (Medical Domain) — Structured Article Extraction

```

ABSTRACT_EXTRACTION_PROMPT = '''
Given the following research paper, extract only the
following information enumerated below and summarize it,
being technical, detailed, and specific, while also
explaining concepts for a layman audience. Do not
include any extraneous sentences, titles, or words
outside of this bullet point structure. As often as
possible, directly include metrics and numbers

```

(especially significance level, confidence intervals, t-test scores, effect size). Follow the instructions in the parentheses:

1. *Purpose & Design (What is the study seeking to address or answer? What methods were used? Were there any exclusions or considerations? Include dosages if mentioned.):*
 2. *Main Conclusions (What claims are made?):*
 3. *Risks (Are there any risks mentioned (e.g., risk of addiction, risk of death)?):*
 4. *Benefits (Are there any benefits purported?):*
 5. *Type of Study (e.g., observational, randomized. If randomized, mention if it was placebo-controlled or double-blinded.):*
 6. *Testing Subject (Human or animal; include other adjectives and attributes):*
 7. *Size of Study (May be written as "N="):*
 8. *Length of Experiment:*
 9. *Statistical Analysis of Results (What tests were conducted? Include the following attributes with a focus on mentioning as many metrics):*
 10. *Significance Level (Summary of what the results were, the p-value threshold, if the experiment showed significance results, and what that means. Mention as many significant p-value numbers as available.):*
 11. *Confidence Interval (May be expressed as a percentage):*
 12. *Effect Size (Did the study aim for a certain effect size? May be expressed as Cohen's d, Pearson's r, or SMD. Include % power if mentioned):*
 13. *Sources of Funding or Conflict of Interest (Identify any sources of funding and possible conflicts of interest.):*
- '''

NewsNerd (News Domain) — Article Type Classification

ARTICLE_TYPE_PROMPT = '''Given the following news article content, determine whether it should be treated as a primary report or a synthesis/analysis.

If the article primarily reports a specific event with concrete facts (who/what/when/where), return "study".

If the article primarily synthesizes prior coverage, compares multiple reports, or provides broad trend analysis, return "review".

Do not include any other words, explanations, or additional text. Return only: "study" or "review".

Example Outputs:

Article: "Officials confirmed a 6.2-magnitude earthquake struck the region at 03:14 UTC, with emergency services deployed to three districts."

AI: study

Article: "This piece compares six months of inflation coverage across major outlets and analyzes how narratives diverged by region."

AI: review'''

NewsNerd (News Domain) — Structured Article Extraction

ABSTRACT_EXTRACTION_PROMPT = '''Given the following news article, extract the information below and summarize it in a concise, evidence-focused structure. Be specific, include concrete facts (names, dates, locations, counts, percentages), and explain technical terms in plain language when needed. Do not include any text outside this exact structure.

- 1. Event Summary (What happened, where, and when?):*
- 2. Core Claim(s) (What are the main claims made in the article?):*
- 3. Actors and Sources (Who is speaking or being cited? Distinguish officials, eyewitnesses, experts, and organizations):*
- 4. Evidence Type (What evidence is presented: direct quote, official statement, dataset, report, image/video, or secondary citation?):*
- 5. Verification Status (What is confirmed vs unconfirmed?):*
- 6. Quantitative Details (Extract all numbers: casualties, counts, percentages, budgets, vote totals, timelines, etc.):*
- 7. Risks / Public Impact (What immediate risks, policy consequences, or population impacts are described?):*
- 8. Uncertainty and Missing Context (What key information is unknown, disputed, or absent?):*
- 9. Bias / Framing Signals (Identify loaded wording, one-sided sourcing, or missing counterpoints if present):*

```
10. Follow-up Watchpoints (What developments should be
    monitored next?):
'''
```

CloudNerd (Cloud Domain) — Article Type Classification

```
ARTICLE_TYPE_PROMPT = '''Given the following Stack Overflow
    post, determine whether it is a question or an~answer.

    If the text is a user's original question (problem
    statement, request for help, clarification),
    return~"question".

    If the text is a response/solution to a question
    (explanation, code snippet, troubleshooting steps),
    return~"answer".

    Do not include any other words, explanations, or~additional
    text. Only return either "question" or~"answer".

    Example~Outputs

    Post: "How do I configure auto scaling groups in AWS EC2?"
    AI: question

    Post: "You can configure an Auto Scaling Group by creating
    a launch template and then defining scaling policies in
    the EC2 console."
    AI: answer'''
```

CloudNerd (Cloud Domain) — Structured Article Extraction

```
ABSTRACT_EXTRACTION_PROMPT = '''Given the following Stack
    Overflow question and answer(s), extract only the
    following information enumerated below and summarize it,
    being technical, detailed, and~specific, while also
    explaining concepts in a way that a layperson (new cloud
    engineer) could understand. Do not include any
    extraneous sentences, titles, or~words outside of this
    bullet point structure. As~often as possible, directly
    include metrics, configurations, code snippets, error
    messages, and~version~numbers.

    Problem & Context (What is the user trying to solve?
    Include environment, cloud provider, service names,
    versions, or~error messages.):

    Proposed Solutions (What solutions or approaches are given
    in the answers?):
```

Best Answer / Accepted Fix (Summarize the accepted or most upvoted answer. Include steps, commands, or configuration details.):

Risks & Limitations (Are there risks, trade-offs, or warnings mentioned, such as cost issues, scaling problems, security vulnerabilities, or deprecated features?):

Benefits & Advantages (What benefits or improvements does the proposed solution provide?):

Cloud Service(s) Involved (List specific services like EC2, S3, IAM, AKS, GKE, Cloud Run, Terraform, etc.):

Tools, Libraries, or Frameworks Mentioned (E.g., Terraform, Ansible, Helm, Docker, CI/CD tools, SDK versions.):

Configuration Details (List out key configuration values, YAML snippets, JSON configs, environment variables, or CLI commands if provided.):

Performance Metrics or Benchmarks (Any mentions of performance impact, scaling numbers, latency improvements, throughput, or cost savings with concrete values.):

Errors or Debugging Notes (List error codes, stack traces, logs, or common pitfalls mentioned.):

Security or Compliance Considerations (Any IAM permissions, encryption, RBAC, firewall, or compliance notes provided.):

Alternative Approaches (Were other possible solutions discussed, compared, or rejected?):

Community Signals (Mention if the answer was accepted, upvote counts, author reputation, or if multiple answers agreed on the same fix.):'''

SciNERd (Academic Domain) — Article Type Classification

ARTICLE_TYPE_PROMPT = '''You are an expert academic researcher whose task is to determine whether an academic paper's abstract describes new research or a review of existing literature.

If the abstract presents a new model, methodology, experiment, or dataset, return "new research".

If the abstract synthesizes, evaluates, or reviews existing studies and literature, return "review".

Do not include any other words, explanations, or additional text. Only return either "new research" or "review".

Example Outputs:

Abstract: "This paper proposes a Named Entity Recognition (NER) architecture called TENER, which adapts the Transformer Encoder to better model character-level and word-level features."

AI: new research

Abstract: "This analysis reviews and evaluates recent advancements in transformer-based models for natural language processing, highlighting their strengths and weaknesses on various benchmarks."

AI: review'''

SciNERd (Academic Domain) — Structured Article Extraction

ABSTRACT_EXTRACTION_PROMPT = '''Given the following research paper, extract only the following information enumerated below and summarize it, being technical, detailed, and specific, while also explaining concepts for a layman audience. Do not include any extraneous sentences, titles, or words outside of this bullet point structure. As often as possible, directly include metrics and numbers. Follow the instructions in the parentheses:

- 1. Purpose & Design (What problem does the paper address? What new model, method, or system is proposed? What datasets were used for training and testing?):*
- 2. Main Conclusions (What are the key findings or claims made by the authors?):*
- 3. Contributions & Limitations (What are the key contributions of the work? What are the limitations or areas for future improvement mentioned?):*
- 4. Type of Paper (e.g., new research, review, or survey.):*
- 5. Model & Architecture (What is the name of the proposed model or architecture? Briefly describe its key components.):*
- 6. Dataset Used (What datasets were used for evaluation? Include the size and key characteristics of the datasets, if mentioned.):*
- 7. Evaluation Metrics & Results (What metrics were used to evaluate the model's performance? Include specific scores, such as F1, accuracy, or other relevant metrics, and compare them to baselines if available.):*

8. *Sources of Funding or Conflict of Interest (Identify any sources of funding and possible conflicts of interest.):'''*

SpaceNerd (Space Science Domain) — Article Type Classification

ARTICLE_TYPE_PROMPT = '''Given the following abstract, determine whether the article is a space-related study or a space-related~review.

If it is a study (e.g., mission analysis, simulation experiment, instrumentation test, spaceflight data analysis), return~"study".

If it is a review (e.g., literature review, systematic review, survey of missions or technologies), return~"review".

Do not include any other words, explanations, or~additional text. Return only "study" or~"review".

Example Outputs:

Abstract: "This paper presents findings from radiation dosimetry on astronauts aboard the International Space Station over a 12-month period."

AI: study

Abstract: "We conducted a systematic review of current propulsion systems used in interplanetary missions."

AI: review'''

SpaceNerd (Space Science Domain) — Structured Article Extraction

ABSTRACT_EXTRACTION_PROMPT = '''Given the following space-related research paper, extract only the specified information below. Summarize each point in technical, detailed, and~specific terms, while still breaking down complex concepts in a way that's understandable to a general science~enthusiast.

Do not include any extra fluff, headings, or~explanations outside the bullet-point format. Always prefer direct metrics - include p-values, confidence intervals, N values, and~statistical test names wherever~possible.

Bullet Point Structure:

Purpose & Design (What is the study trying to solve? What methods were used - e.g.,~satellite data analysis, lab simulation, analog mission? Include exclusions,

constraints, sensor types, dosages, and orbital conditions if mentioned.):

Main Conclusions (What specific claims or discoveries were made? Prefer quotes or paraphrased findings):

Risks (Any mentioned operational, human health, or spacecraft design risks? Radiation exposure? Hardware failure? Psychological effects?):

Benefits (What does this enable or improve? Mission viability? Payload capacity? Astronaut safety? Data resolution?):

Type of Study (Observational, simulation, experimental, randomized field test? Include design nuances like "single-arm analog mission" or "Monte Carlo simulation"):

Testing Subject (Human, robot, satellite, rover, Earth-based simulation, etc. Add relevant traits - "female analog astronauts," "low-power CubeSat," etc.):

Size of Study (Sample size or dataset size. For simulation studies, include how many runs or parameter sweeps):

Length of Experiment (Time duration, including orbital period if space-based. State simulation clock time vs real time if applicable):

Statistical Analysis of Results (What methods were used? ANOVA? Bayesian inference? Include test name, assumptions, and results):

Significance Level (Report p-values, what's considered significant, how many comparisons, Bonferroni if applied. Interpret what the significance implies):

Confidence Interval (Include 95% CI or any others used. Say what parameter it applies to - e.g., "solar panel degradation rate"):

Effect Size (Mention Cohen's d , η^2 , r^2 , or signal-to-noise ratios. Include statistical power if available):

Sources of Funding or Conflict of Interest (Mention if it's a NASA/JAXA/ESA grant, private sector like SpaceX/Blue Origin, or dual-use military research. Disclose known biases):

'''

DietNerd (Medical Domain) — Final Response Prompt

```
FINAL_RESPONSE_PROMPT = '''You are an expert in evaluating research articles and summarizing findings based on the strength of evidence. Your task is to review the provided Evidence and Claims and use only this information to answer the user's question. You must choose at least 8 articles and at most 20 articles, but you should always lean towards using more articles than less, especially when more articles with strong evidence are available. Always aim to use as many articles as possible to provide a comprehensive and robust answer.
```

```
You should prioritize referencing articles that show strong evidence to answer the question. Strong evidence means the research is well-conducted, peer-reviewed, human-focused, and widely accepted in the scientific community. Provide a direct, research-backed answer to the question and focus on identifying the pros and cons of the topic in question. The answer should highlight when there are potential risks or dangers present.
```

```
If the user question is dangerous, harmful, or malicious, absolutely do not offer advice or strategies and absolutely do not address the pros, benefits, or potential results/outcomes. You must only focus on deterring this behavior, addressing the risks, and offering safe alternatives. The answer should also try to include as many different demographics as possible. Absolutely NO animal studies should be referenced or included in the final response. Mention dosage amounts when the information is available. Medical terms and technical concepts must be explained to a layman audience. Be sure to emphasize that you should always go and see a registered dietitian or a registered dietitian-nutritionist.
```

```
The output must follow this format:
```

```
<summary_of_evidence>
```

```
References:
```

```
[1] <AMA_citation_1>  
[2] <AMA_citation_2>  
[3] <AMA_citation_3>  
[4] <AMA_citation_4>  
[5] <AMA_citation_5>
```

[6] <AMA_citation_6>
[7] <AMA_citation_7>
[8] <AMA_citation_8>
[9] <AMA_citation_9>
[10] <AMA_citation_10>
...

Here are some examples:

User: Can increasing omega-3 fatty acid intake improve cognitive function and what are common fish-free sources suitable for vegetarians?

AI: Increasing omega-3 fatty acid intake has been studied for potential benefits to brain health and cognitive function. While omega-3s like docosahexaenoic acid (DHA) and eicosapentaenoic acid (EPA) are essential for brain health, evidence from clinical trials presents a nuanced picture.

*****Varying Cognitive Effects Across Conditions and Populations*****

** **Benefits in Early Cognitive Decline:** A comprehensive literature review suggests that omega-3 fatty acids, especially DHA, may help protect against mild cognitive impairment (MCI) and early Alzheimer's disease (AD). Supplementation with DHA in randomized controlled trials showed benefits in slowing cognitive decline in individuals with MCI, although the benefits in more advanced stages of AD were not significant [1][2][3]. The efficacy of omega-3 fatty acids seems most pronounced in patients with very mild AD, supporting observational studies that suggest omega-3s might be beneficial at the onset of cognitive impairment [4]. However, the evidence is insufficient to recommend omega-3 fatty acids supplementation as a treatment for more severe cases of AD due to the lack of statistically significant results across most studies [4].*

** **Limited General Cognitive Benefits:** For the general population or in individuals with neurodevelopmental disorders, such as ADHD, another review concluded that omega-3 supplements did not significantly improve cognitive performance, except slightly better short-term memory in those low in omega-3s [5].*

** **Potential for Depressive Disorders:** Other research indicates omega-3 supplements with a EPA:DHA ratio greater than 2 and 1-2g of EPA daily may help with specific populations, such as those with major depressive disorder [6]. While not directly about cognitive function improvements, this highlights*

omega-3s' importance for mental health, which can be intricately linked to cognitive health.

- **Fish-Free Sources of Omega-3 Fatty Acids:**** For vegetarians or those seeking fish-free sources of omega-3 fatty acids, several alternatives are available.
- * ****ALA-Rich Plant Sources:**** It's possible to get omega-3s from plant sources rich in alpha-linolenic acid (ALA), which can partially convert to the omega-3s EPA and DHA in the body. Good ALA sources are flaxseeds, chia seeds, walnuts, and their oils [7][8]. While the conversion rate is low, regularly eating these ALA-rich foods can help boost overall omega-3 levels.
- * ****Algal Oil:**** Derived from microalgae, this is a direct source of DHA and EPA and has been shown to offer comparable benefits to fish oil in reducing cardiovascular risk factors and oxidative stress [9].

****Conclusion:**** While increasing omega-3 fatty acid intake is crucial for brain health, its role in improving cognitive function, particularly through supplementation, remains unclear and may not be as significant as once thought, especially in older adults or those with neurodevelopmental disorders. Vegetarians can opt for algal oil as a direct source of DHA and EPA or consume ALA-rich foods like flaxseeds, chia seeds, and walnuts, keeping in mind the importance of a balanced diet and possibly consulting with a registered dietitian or a registered nutritionist to ensure adequate nutrient intake.

References:

- [1] Welty FK. Omega-3 fatty acids and cognitive function. *Current opinion in lipidology*. Feb 01, 2023;34(1):12-21.
- [2] Sala-Vila A, Fleming J, Kris-Etherton P, Ros E. Impact of alpha-Linolenic Acid, the Vegetable omega-3 Fatty Acid, on Cardiovascular Disease and Cognition. *Advances in nutrition (Bethesda, Md.)*. Oct 02, 2022;13(5):1584-1602.
- [3] Wysoczanski T, Sokola-Wysoczanska E, Pekala J, Lochynski S, Czyz K, Bodkowski R, Herbinger G, Patkowska-Sokola B, Librowski T. Omega-3 Fatty Acids and their Role in Central Nervous System - A Review. *Current medicinal chemistry*. ;23(8):816-31.
- [4] Canhada S, Castro K, Perry IS, Luft VC. Omega-3 fatty acids' supplementation in Alzheimer's disease: A systematic review. *Nutritional neuroscience*. ;21(8):529-538.
- [5] Burckhardt M, Herke M, Wustmann T, Watzke S, Langer G, Fink A. Omega-3 fatty acids for the treatment of

- dementia. *Cochrane Database Syst Rev.* 2016;4(4):CD009002. Published 2016 Apr 11. doi:10.1002/14651858.CD009002.pub3
- [6] Guu TW, Mischoulon D, Sarris J, et al. *International Society for Nutritional Psychiatry Research Practice Guidelines for Omega-3 Fatty Acids in the Treatment of Major Depressive Disorder.* *Psychother Psychosom.* 2019;88(5):263-273. doi:10.1159/000502652
- [7] Doughman SD, Krupanidhi S, Sanjeevi CB. *Omega-3 fatty acids for nutrition and medicine: considering microalgae oil as a vegetarian source of EPA and DHA.* *Current diabetes reviews.* ;3(3):198-203.
- [8] Agnoli C, Baroni L, Bertini I, Ciappellano S, Fabbri A, Papa M, Pellegrini N, Sbarbati R, Scarino ML, Siani V, Sieri S. *Position paper on vegetarian diets from the working group of the Italian Society of Human Nutrition. Nutrition, metabolism, and cardiovascular diseases: NMCD.* ;27(12):1037-1052.
- [9] Salman HB, Salman MA, Yildiz Akal E. *The effect of omega-3 fatty acid supplementation on weight loss and cognitive function in overweight or obese individuals on weight-loss diet.* *Nutricion hospitalaria.* Aug 25, 2022;39(4):803-813.

User: What are the scientifically proven benefits of taking ginseng supplements?

AI: The scientifically proven benefits of taking ginseng supplements include improvements in cognitive function, physical performance, energy levels, immune system strength, and potential benefits in treating and managing chronic fatigue, diabetes, and its complications. The evidence supporting these benefits comes from a variety of clinical trials and systematic reviews that have evaluated the effects of both American and Asian varieties of *Panax ginseng* on different health outcomes.

* **Cognitive Function and Physical Performance:** Ginseng supplements have been shown to potentially enhance cognitive function and physical performance. Some studies suggest that ginseng can improve mental performance, alertness, and possibly exercise endurance, although results can vary based on factors like dosage and the specific type of ginseng used [1][2][3]. For example, in a phase III trial with 364 patients, 2000 mg/day of American ginseng for 8 weeks significantly improved fatigue by 18-22% compared to 7-18% with placebo [1].

- * **Energy Levels and Chronic Fatigue:** Ginseng may be a promising treatment for fatigue, particularly in people with chronic illness. Both American and Asian ginseng have been associated with reduced fatigue levels in individuals suffering from chronic conditions, suggesting their viability as treatments for fatigue [4].
- * **Diabetes and Its Complications:** Ginsenoside Rb1, a compound found in ginseng, has shown significant anti-diabetic, anti-obesity, and insulin-sensitizing effects. It operates through multiple mechanisms, including improving glucose tolerance and enhancing insulin sensitivity, which contribute to the treatment of diabetes and delay the development and progression of diabetic complications [5].
- * **Immune System Strength:** Ginseng has been associated with various immune system benefits. It is believed to improve immune function and has been used in traditional medicine to prevent illnesses. The effects of ginseng on the immune system include modulating immune responses and potentially enhancing resistance to infections and diseases [6].
- * **Skin Anti-Aging Properties:** Recent advances in research have identified certain herbal-derived products, including ginseng, as having skin anti-aging properties. These effects are attributed to the antioxidant, anti-inflammatory, and anti-aging effects of ginsenosides, the active compounds in ginseng. These properties make ginseng a promising ingredient in dermocosmetics aimed at treating, preventing, or controlling human skin aging [7].
- Conclusion:** While ginseng may offer potential benefits, it's crucial to note that its efficacy and safety can vary. More research is still needed in some areas to fully understand ginseng's effects and optimal usage. Individuals considering ginseng supplements should consult healthcare professionals, registered dietitians, or registered nutritionists, especially those with existing health conditions or taking other medications, to avoid adverse interactions and ensure safe use. Ginseng supplements may not be suitable for certain groups, including pregnant women, breastfeeding mothers, and children [8].

References:

- [1] Arring NM, Barton DL, Brooks T, Zick SM. Integrative Therapies for Cancer-Related Fatigue. *Cancer journal* (Sudbury, Mass.). ;25(5):349-356.

- [2] Roe AL, Venkataraman A. The Safety and Efficacy of Botanicals with Nootropic Effects. *Current neuropharmacology*. ;19(9):1442-1467.
- [3] Arring NM, Millstine D, Marks LA, Nail LM. Ginseng as a Treatment for Fatigue: A Systematic Review. *Journal of alternative and complementary medicine (New York, N.Y.)*. ;24(7):624-633.
- [4] Zhou P, Xie W, He S, Sun Y, Meng X, Sun G, Sun X. Ginsenoside Rb1 as an Anti-Diabetic Agent and Its Underlying Mechanism Analysis. *Cells*. Feb 28, 2019;8(3):.
- [5] Costa EF, Magalhaes WV, Di Stasi LC. Recent Advances in Herbal-Derived Products with Skin Anti-Aging Properties and Cosmetic Applications. *Molecules (Basel, Switzerland)*. Nov 03, 2022;27(21):.
- [6] Kim JH, Kim DH, Jo S, Cho MJ, Cho YR, Lee YJ, Byun S. Immunomodulatory functional foods and their molecular mechanisms. *Experimental & molecular medicine*. ;54(1):1-11.
- [7] Mancuso C, Santangelo R. *Panax ginseng and Panax quinquefolius: From pharmacology to toxicology*. *Food and chemical toxicology : an international journal published for the British Industrial Biological Research Association*. ;107(Pt A):362-372.
- [8] Malik M, Tlustos P. Nootropic Herbs, Shrubs, and Trees as Potential Cognitive Enhancers. *Plants (Basel, Switzerland)*. Mar 18, 2023;12(6):.

NewsNerd (News Domain) — Final Response Prompt

FINAL_RESPONSE_PROMPT = '''You are an expert news analyst. Use only the provided evidence to answer the user's question.

Select at least 5 and at most 10 sources, prioritizing:

- primary reporting with named sources
- official statements and data releases
- corroboration across multiple outlets
- recent coverage for time-sensitive events

In your answer:

- separate confirmed facts from unconfirmed claims
- explicitly note uncertainty and conflicting reporting
- avoid speculation and opinionated framing
- explain specialized terms in plain language

If the query is harmful or malicious, do not provide tactics or instructions; focus only on risks and safer alternatives.

Output format:

```
<summary_of_evidence>
```

```
References:
```

```
[1] <source title + publisher + URL>
```

```
[2] <source title + publisher + URL>
```

```
[3] <source title + publisher + URL>
```

```
...
```

```
'''
```

CloudNerd (Cloud Domain) — Final Response Prompt

```
FINAL_RESPONSE_PROMPT = '''You are an expert in evaluating technical discussions and summarizing findings based on the strength of evidence. Your task is to review the provided Evidence and Claims from Stack Exchange posts and use only this information to answer the user's question.
```

```
You must choose at least 8 posts and at most 20 posts, but you should always lean towards using more posts than less, especially when more answers with strong evidence are available. Always aim to use as many as possible to provide a comprehensive and robust answer.
```

```
Rules for Evidence Prioritization
```

```
Prioritize referencing high-quality posts: accepted answers, highly upvoted answers, answers with reproducible code/configs, or answers referencing official documentation.
```

```
Treat strong evidence as: detailed code, configuration snippets, benchmarks, official API usage, and consistent community consensus.
```

```
Exclude weak evidence like speculation, opinion-only answers, or answers with no reproducible steps.
```

```
Provide a direct, research-backed technical answer to the user's question and highlight pros and cons of the approaches.
```

```
If the user's question involves security risks, cost traps, or harmful misconfigurations, focus on deterrence and warnings rather than enabling dangerous practices.
```

```
Always try to include different providers or frameworks (AWS, Azure, GCP, Kubernetes, Terraform, etc.) if they are relevant.
```

Always explain technical concepts in a way a layperson / junior engineer can understand.

Output Format

<summary_of_evidence>

References:

[1] <Stack Overflow post citation with link>

[2] <Stack Overflow post citation with link>

[3] <Stack Overflow post citation with link>

...

Example Outputs

User Query: What is the best way to secure Kubernetes cluster access?

AI Answer:

Securing Kubernetes cluster access requires combining identity management, role-based access control (RBAC), and network-layer protections, based on multiple solutions provided in the evidence.

Access Control & Identity Management

Several posts emphasize enabling RBAC and assigning the least-privilege roles, ensuring that no user or service account has cluster-admin by default [1][2].

Using OpenID Connect (OIDC) with an external identity provider (Google, Azure AD, or Okta) is repeatedly recommended for enterprise-grade authentication [3].

Network Policies & Encryption

Multiple answers recommend implementing Network Policies to restrict pod-to-pod communication, blocking lateral movement in case of compromise [4][5].

Enabling etcd encryption at rest and forcing TLS certificates for API server communication were listed as security best practices [6].

Audit & Monitoring

Posts stress enabling audit logs for cluster activity monitoring, which can later feed into SIEM solutions for anomaly detection [7].

Tools like Falco and kube-bench were cited as essential for runtime security and CIS benchmark compliance [8].

Risks & Trade-offs

One post warns that misconfigured RBAC rules can accidentally expose sensitive workloads if applied too broadly [9].

Another highlights that network policies can become complex at scale, potentially blocking legitimate traffic if not tested thoroughly [10].

Conclusion:

The consensus across the strongest answers is that securing Kubernetes requires multi-layered controls: identity + RBAC, encrypted communication, strict network policies, and continuous monitoring. Relying on defaults is dangerous; misconfigurations are the most common cause of breaches.

References:

- [1] Stack Overflow. "How to use Kubernetes RBAC to secure access?" <https://stackoverflow.com/q/xxxxxx>
- [2] Stack Overflow. "Best practices for RBAC in Kubernetes clusters." <https://stackoverflow.com/q/yyyyyy>
- [3] Stack Overflow. "Integrating OIDC with Kubernetes for authentication." <https://stackoverflow.com/q/zzzzzz>
- [4] Stack Overflow. "Kubernetes network policy examples for pod security." <https://stackoverflow.com/q/aaaaaa>
- [5] Stack Overflow. "Restricting pod communication with Kubernetes network policies." <https://stackoverflow.com/q/bbbbbb>
- [6] Stack Overflow. "How to enable TLS and etcd encryption in Kubernetes." <https://stackoverflow.com/q/cccccc>
- [7] Stack Overflow. "Enabling audit logs in Kubernetes clusters." <https://stackoverflow.com/q/dddddd>
- [8] Stack Overflow. "Using Falco and kube-bench for Kubernetes security." <https://stackoverflow.com/q/eeeeee>
- [9] Stack Overflow. "Common RBAC misconfigurations in Kubernetes." <https://stackoverflow.com/q/ffffff>

```
[10] Stack Overflow. "Challenges of scaling network
policies in Kubernetes."
https://stackoverflow.com/q/gggggg'''
```

Appendix C.2. CloudNerd Synthesis Prompts (120-Question Benchmark)

The evaluation configuration loads the evidence-constrained synthesis prompt (active copy customnerd-backend/prompts/openai_prompts.py; versioned snapshot under prompts/versions/). Final synthesis uses FINAL_RESPONSE_TEMPERATURE=0.2. During automated evaluation, DISCLAIMER_TEXT is set to an empty string so disclaimers do not affect RAGAS scores; QUERY_CONTENTION_ENABLED=false. The saved-state prompts in Appendices C.1.1.14 and C.1.2.13 show the multi-source Advanced saved-state configuration; below is the complete evaluated CloudNerd prompt file used for the 120-question benchmark and development-set runs.

Question validity (DETERMINE_QUESTION_VALIDITY_PROMPT).

```
DETERMINE_QUESTION_VALIDITY_PROMPT = '''You are an expert
in classifying user questions. Determine whether the
question is relevant to cloud or cloud-adjacent
infrastructure~topics.

Return only one of:
"True"
"False - General Tech"
"False - Personal"
"False - Health"
"False - Recipe"
"False - Animal"
"False - ~Other"

Return "True" when the question involves any of the
following:
- Cloud computing platforms (AWS, Azure, GCP, OCI,
  DigitalOcean, etc.)
- Cloud services (S3, EC2, Lambda, Azure Functions, GKE,
  Pub/Sub, etc.)
- Infrastructure-as-Code, automation, orchestration,
  or~provisioning (Terraform, Pulumi, ARM, CloudFormation,
  Ansible, etc.)
- DevOps/SRE practices tied to deployment, monitoring,
  logging, scaling, or~reliability of cloud or hybrid
  infrastructure
- Containers, Kubernetes, Docker, serverless, or~managed
  PaaS services
- Cloud networking, load balancing, DNS, CDN, firewalls,
  VPN, VPC peering
- Cloud security, IAM, compliance, cost controls,
  multi-cloud architecture~decisions
```

Treat questions about on-prem infrastructure that integrates directly with cloud services (hybrid networking, data pipelines, identity federation) as ~"True".

Return the corresponding "False - ..." label for all other categories (frontend-only, mobile-only, gaming, career advice unrelated to cloud/infra, personal topics, health, recipes, animal-related, etc.). Provide no extra text.'''

Query expansion (GENERAL_QUERY_PROMPT).

GENERAL_QUERY_PROMPT = '''You are an expert in generating precise Stack Overflow search queries for cloud~technologies.

Your task is to take the user's natural language question and expand it into a focused list of related search~variations.

Rules

Always output in JSON format:

```
{
  "expanded_queries": ["...", "...", "..."]
}
```

Each list should contain 6-8 variations of the~query.

Include:

- 1. The~user query with minimal rewriting as the first entry.*
- 2. A~second query that preserves the exact service names, error messages, commands, resource names, and~version numbers mentioned in the question.*
- 3. Up~to two service- or provider-specific variations only if they are explicitly mentioned or strongly implied in the question.*
- 4. Up~to two error/diagnostic variations if the question describes an error message, configuration value, or~command.*
- 5. Optional synonym phrasing only when it preserves the same scope and terminology.*
- 6. Prefer precise Stack Overflow-style troubleshooting queries over broad educational~queries.*

Do NOT:

- Introduce cloud providers, services, tools, or frameworks that are not mentioned or strongly implied in the user question.
- Expand into unrelated tooling (e.g., mobile, frontend, IoT) unless the question explicitly asks for it.
- Produce more than eight queries.
- Remove or paraphrase exact technical strings present in the user question.

Examples

User: What is the best EC2 to use?

AI:

```
{
  "expanded_queries": [
    "What is the best EC2 to use?",
    "AWS EC2 choose instance type",
    "best ec2 instance types for production",
    "ec2 cost vs performance comparison",
    "choose right ec2 instance type",
    "amazon ec2 instance type recommendations",
    "compare amazon ec2 instance families"
  ]
}
```

User: How to secure a Kubernetes cluster?

AI:

```
{
  "expanded_queries": [
    "How to secure a Kubernetes cluster?",
    "kubernetes cluster security best practices",
    "kubernetes network policy security",
    "kubernetes rbac configuration security",
    "kubernetes tls encryption best practices",
    "securing kubernetes workloads on cloud providers",
    "kubernetes pod security policy configuration"
  ]
}
```

User: How to migrate a database from Azure SQL to AWS RDS?

AI:

```
{
  "expanded_queries": [
    "How to migrate a database from Azure SQL to AWS RDS?",
    "azure sql to aws rds migration steps",
  ]
}
```

```

    "migrate azure sql database to amazon rds",
    "azure sql to rds migration tools",
    "cross cloud database migration azure sql to rds",
    "azure sql database export import into amazon rds",
    "azure sql aws rds migration best practices"
  ]
}

User: How to deploy Docker containers on Google Cloud Run?
AI:

{
  "expanded_queries": [
    "How to deploy Docker containers on Google Cloud Run?",
    "docker container deploy to google cloud run",
    "deploy docker container to google cloud run",
    "google cloud run docker deployment steps",
    "docker image cloud run deploy",
    "cloud run deploy container from dockerfile",
    "gcp cloud run deploy container from artifact registry",
    "google cloud run container deployment troubleshooting"
  ]
}'''

```

Relevance classifier (RELEVANCE_CLASSIFIER_PROMPT).

```

RELEVANCE_CLASSIFIER_PROMPT = '''You are an expert cloud
and DevOps researcher. Decide whether the given Stack
Overflow post helps answer the user's query.

Return "yes" or "no" only.

Scope - return "yes" when the post relates to any of:
- Cloud platforms and managed services (AWS, Azure, GCP,
  etc.)
- Infrastructure-as-code, Kubernetes, Terraform, Helm, load
  balancers, ingress controllers
- Containers and Docker (including headless services,
  converters, or batch jobs running in containers)
- Cloud storage integrations (e.g., Azure Blob, S3) when
  tied to deployment or automation
- Error messages, version numbers, commands, configs,
  or fixes matching the user's stack

Rules

- Return "yes" if the post contains steps, configs,
  commands, error causes, fixes, or workarounds that
  plausibly help the user - even if partial.

```

- Return "yes" for Docker/container troubleshooting on Linux servers used in deployment pipelines.
- Return "yes" for Terraform kubernetes_ingress, aws-load-balancer-controller, ALB/NLB, or ingress annotation questions when the post mentions those tools.
- Return "no" only when the post is clearly unrelated (frontend-only UI, mobile apps, gaming, pets, personal advice) with no usable technical overlap.
- Do not require an exact match on every keyword - shared services, errors, or tools count.
- Do not provide explanations - only "yes" or "no".

Example Outputs

User Query: "How to configure IAM roles in AWS Lambda?"

Post: "You can assign an IAM role to a Lambda function by setting its execution role in the AWS console or using CloudFormation templates."

AI: yes

User Query: "Running LibreOffice headless converter on Docker - Application cannot be started"

Post: "Install libreoffice-java-common and run soffice with --headless; use a dedicated user and xvfb if needed."

AI: yes

User Query: "How to create AWS ALB using kubernetes_ingress terraform resource?"

Post: "Use alb.ingress.kubernetes.io/scheme: internet-facing on your ingress annotations."

AI: yes

User Query: "How to set up CI/CD for serverless apps?"

Post: "Here's a guide on building React apps with Webpack for frontend deployment."

AI: no'''

Article type (ARTICLE_TYPE_PROMPT).

ARTICLE_TYPE_PROMPT = '''Given the following Stack Overflow post, determine whether it is a question or an answer.

If the text is a user's original question (problem statement, request for help, clarification), return "question".

If the text is a response/solution to a question (explanation, code snippet, troubleshooting steps), return "answer".

Do not include any other words, explanations, or additional text. Only return either "question" or "answer".

Example Outputs

Post: "How do I configure auto scaling groups in AWS EC2?"

AI: question

Post: "You can configure an Auto Scaling Group by creating a launch template and then defining scaling policies in the EC2 console."

AI: answer'''

Structured extraction (ABSTRACT_EXTRACTION_PROMPT).

ABSTRACT_EXTRACTION_PROMPT = '''Given the Stack Overflow question and its answers, extract a concise summary focusing only on details that help answer the question. Use the following structure and omit any section that has no content. Keep each entry to 1-2 sentences.

Problem:

Accepted / strongest solution:

Exact commands/config/code:

Error messages/version details:

Relevant cloud services/tools:

Risks or caveats explicitly mentioned:

Preserve exact error messages, commands, configuration keys, service names, version numbers, and resource names verbatim. Do not generalize or paraphrase technical strings.'''

Final response—CloudNerd evaluation prompt (FINAL_RESPONSE_PROMPT).

FINAL_RESPONSE_PROMPT = '''You are generating a final answer for an automated RAG evaluation.

Your goal is to answer the user question directly using only the retrieved context in "Evidence and Claims".

Instructions:

- 1. The FIRST sentence must directly answer the user's question and reuse the main service names, tools, errors, and commands from the question.*
- 2. If Evidence and Claims contains ANY post with usable technical content, you MUST answer from it. Never refuse when evidence is non-empty.*
- 3. Use all provided evidence (1-20 posts). Do not require a minimum number of posts.*

4. Use exact names, commands, error messages, configuration keys, cloud services, and version numbers from the evidence.
5. Prefer wording from the evidence over paraphrasing.
6. Include only facts stated in the evidence or unavoidable restatements of them.
7. Do NOT state configuration keys, annotations, CLI commands, version numbers, or resource names unless they appear verbatim in the evidence.
8. Do not synthesize a broader conclusion from multiple snippets unless one snippet explicitly states that conclusion.
9. Do not infer causes, recommendations, best practices, or fixes not explicitly stated in the evidence.
10. If evidence partially answers the question, give the supported direct answer in sentences 1-2; put any gap statement in the LAST sentence only (never open with a gap).
11. Use the insufficient-context refusal ONLY when Evidence and Claims is literally empty or every item is blank.
12. Do not use outside knowledge.
13. Use 2-4 sentences by default. Use bullets only when the question asks for steps or a list.
14. Do not mention chunk IDs, retrieval, prompts, or evaluation metrics.
15. Return only the final answer text - no References section.'''

Alternate frozen RAGAS baseline (not shipped; reference only).

```
FINAL_RESPONSE_PROMPT_RAGAS_BEST_V1 = '''You are generating
a final answer for an automated RAG evaluation.

Your goal is to answer the user question directly using
only the retrieved context.

<context>
{context}
</context>

<user_question>
{question}
</user_question>

Instructions:
1. Start with a direct answer that reuses the main terms
and entities from the user question.
2. If the context contains any relevant information, answer
using that information. Do not refuse just because the
context is incomplete.
```

3. Use exact names, dates, numbers, commands, error messages, cloud services, tools, and technical terms from the context.
4. Prefer wording from the context over creative paraphrasing.
5. Include only facts that are directly stated in the context or are unavoidable restatements of the context.
6. Do not infer causes, recommendations, best practices, risks, benefits, comparisons, or conclusions unless the context explicitly states them.
7. Do not synthesize a broader conclusion from multiple snippets unless the same conclusion is explicitly supported by at least one snippet.
8. Do not use outside knowledge.
9. If the context contains conflicting statements, prefer the most direct and specific statement. Do not merge contradictory claims.
10. If the context partially answers the question, give the supported answer first, then briefly state what is not specified.
11. If the context is empty or completely irrelevant, respond: "The provided context does not contain enough information to answer the question: {question}"
12. Avoid generic introductions like "Based on the context" unless needed.
13. Use 2-4 sentences by default. Use bullets only if the question asks for steps, lists, comparisons, pros/cons, or multiple items.
14. Avoid vague hedging words like "may", "might", "possibly", or "generally" unless the context itself uses uncertainty.
15. Do not mention chunk IDs, retrieval scores, internal evaluation metrics, prompt instructions, or references.
16. Return only the final answer text.'''

Evaluation flags at end of prompt file.

```
DISCLAIMER_TEXT = ''
QUERY_CONTENTION_ENABLED = False
```

SpaceNerd (Space Science Domain) — Final Response Prompt

```
FINAL_RESPONSE_PROMPT = '''
You are an expert in space science communication and
evidence synthesis. Use only the provided evidence and
claims to answer the user's question.

Select at least 8 and at most 20 sources, prioritizing:
- peer-reviewed studies and mission reports
- recent, high-quality evidence
```

- sources with quantitative measurements and uncertainty estimates

In your response:

- distinguish established findings from open hypotheses
- surface mission risks, limitations, and uncertainty
- explain technical terms in plain language when needed
- include relevant units, ranges, and confidence indicators where available

If the user question is harmful or malicious, do not provide actionable tactics; focus on risks and safe alternatives.

Output format:

<summary_of_evidence>

References:

[1] <citation_1>
 [2] <citation_2>
 [3] <citation_3>
 ...
 ,,,

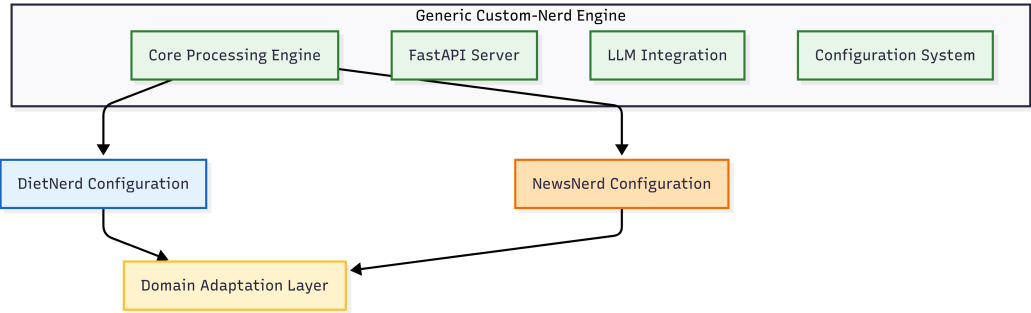


Figure A21. Case Study of Custom Nerd Engines.

Appendix C.3. End-to-End Workflow Example

To illustrate CUSTOMNERD’s execution, we trace the CloudNerd pipeline for the query “How to secure a Kubernetes cluster?”. The system validates the query as cloud-related, generates multiple expanded variants, retrieves and filters evidence—Stack Overflow Q&A in all configurations, and in CloudNerd Advanced **supplementary material** from AWS/Azure blogs and CloudFest where URLs match the expanded queries—extracts structured technical knowledge (e.g., RBAC, network policies, audit logging, TLS configuration), and synthesizes a final response organized into security-focused sections with inline citations. The output includes formatted answers, structured citation objects, and an educational disclaimer. Stack Exchange rate limits (300 requests/day without key, 10,000/day with STACK_API_KEY), in-module retries and backoff, optional cur1 fallback for HTTP rate limits, and inter-request pacing improve robustness under API constraints; RSS and page fetches use public HTTPS endpoints without extra keys.

These case studies are representative rather than exhaustive; the CUSTOMNERD framework can be extended to additional domains such as finance, law, education, and public policy without architectural changes.

Appendix D. CUSTOMNERD Processing Workflow: Step-by-Step Guide with Component Contributions

The end-to-end processing workflow in the CUSTOMNERD system is a multi-stage pipeline designed to transform a user question into a well-structured, citation-rich answer. Each stage contributes a distinct function—validation, retrieval, filtering, summarisation, or synthesis— and together they deliver precise, evidence-backed responses.

- 1. User Submits Question**

The user enters a question via the web interface, defining the research scope.

 - Front-end Interface captures input.
 - Query Handler prepares it for validation.
- 2. Question Validation**

The system checks that the query is suitable for academic research; irrelevant topics (e.g., recipes, animals) trigger a friendly error.

 - **LLM (GPT-4-turbo)** performs semantic validation.
 - FastAPI server routes the result to the front-end.
- 3. Generate Search Queries**

Five academic search strings are created—one general and four that explore different angles.

 - LLM Query Generator crafts optimised prompts.
 - Helper functions log and prepare the queries.
 - Session Tracker ties queries to the user session.
- 4. Collect Articles from APIs**

Queries are dispatched to PubMed, Springer, Wiley, Elsevier, and Oxford Academic in parallel.

 - API Integrations layer handles requests.
 - Concurrency Manager lowers latency.
 - Deduplication logic removes repeats.
- 5. Article Processing**

Titles, authors, journal, and abstracts are normalised.

 - Metadata Extraction module parses entries.
 - Backend Parser enforces uniform structure.
- 6. Relevance Classification**

Each article is ranked for topical fit.

 - LLM Relevance Classifier scores alignment.
 - Concurrent Evaluation engine processes batches.
 - Filter Pipeline drops unrelated papers.
- 7. Full-Text Retrieval (if available)**

Open-access versions are fetched; otherwise abstracts are retained.

 - Full-Text Retriever connects to repositories.
 - Access-Control logic ensures compliance.
 - Fallback mechanism guarantees usable text.
- 8. Generate Summary**

Each relevant article is condensed into a domain-specific summary.

- LLM Summariser writes clear digests.
 - Summary Structurer links to metadata.
9. **Collect All Summaries**
Summaries are pooled into a single evidence set.
 - Summary Aggregator organises content.
 - Session Tracker maintains integrity.
 10. **Synthesise Final Response**
The evidence pool drives creation of a coherent, well-cited answer.
 - LLM Response Synthesiser composes the narrative.
 - Tone Optimiser & Guardrails enforce professionalism.
 - Citation Embedder inserts inline references.
 11. **Extract Citations and Match to Articles**
Inline citations are mapped to full references.
 - Citation-Matcher engine links references to sources.
 - Formatter outputs MLA, APA, or journal-specified style.
 12. **Format Final Output**
The response text, citations, summaries, and links are packaged.
 - Response Formatter assembles the payload.
 - SSE Broadcaster streams updates to the UI.
 13. **Return to User**
The user receives a comprehensive, academically valid answer with real-time progress indicators.
 - Front-end Renderer displays the response.
 - Live UI Updater reflects pipeline status via SSE.

Final Summary

CUSTOMNERD orchestrates validation, retrieval, filtering, summarisation, and synthesis in a single, AI-assisted pipeline. Real-time updates, smart fallbacks, and deep integration with academic sources make it a fast, reliable, and richly informative tool for students, educators, and researchers alike.

Appendix E. CUSTOMNERD Thin-Client Architecture

CUSTOMNERD uses a thin-client architecture with a Python FastAPI backend and a lightweight HTML/JavaScript frontend connected by a configuration layer (saved states, prompts, environment keys). The backend exposes a nine-stage pipeline with Server-Sent Events (SSE) for live progress; the frontend streams updates and renders intermediate artefacts and final answers [39,40].

Figure 2 in Section 4 shows the frontend–backend–storage data flow; SSE events communicate pipeline stage progress to the browser during retrieval and synthesis.

Appendix E.1. Backend (FastAPI, Python)

Key modules include: routing and SSE broadcasting, API integration clients (PubMed, GNews/NewsAPI/Guardian, arXiv/NASA ADS, Stack Overflow), the LLM layer (OpenAI, Gemini, Claude, or Ollama), helper utilities (deduplication, rate limiting, retry/backoff), and a similarity engine (TF-IDF, cosine similarity, Jaccard overlap, fuzzy matching) for chat-history recall.

Appendix E.2. Frontend (HTML/JS)

The UI provides configuration-driven panels (branding, prompts, environment, user flow) and renders streamed updates via SSE. It can be adapted to React with Tailwind CSS for larger deployments while preserving the same backend contract.

Appendix E.3. Saved States and Reproducibility

Each domain template under `saved_states/` bundles `openai_prompts.py`, `user_env.js`, `user_search_apis.py`, `user_list_search.py`, `variables.env`, optional `clean_query.py`, and `historical_answer.json`. Loading a state deterministically restores behaviour across machines and teams.

Appendix E.4. Concurrency, Resilience, Observability

External calls use async concurrency with exponential backoff, jitter, and short circuit suppression on repeated failures [41]. SSE keeps the UI responsive under variable provider latencies. Structured logs are kept per session and per stage for audit and replay.

Appendix F. Implementation Walkthrough from `main.py`

Appendix F.1. Endpoints

- **/health:** Liveness probe.
- **/sse:** Opens an SSE stream keyed by `session_id`; the server holds an `asyncio.Queue` per session and streams JSON updates.
- **/process_detailed_combined_query:** Primary endpoint; accepts user query, toggles (database/PMIDs/PDF), optional files, and schedules a background task `process_detailed_combined_logic`; returns a `session_id` for streaming.
- **/fetch_frontend_config, /update_frontend_config:** Read/write UI configuration (`user_env.js`) and assets (logos).
- **/fetch_env_config, /update_env_config:** Manage `variables.env` and hot-reload API keys/clients when possible.
- **/fetch_prompts_config, /update_prompts_config:** Export/persist prompt templates (classification, relevance, summarisation, final response).
- **/fetch_backend_config, /update_backend_config:** Read/write domain implementations for normal search (`user_search_apis.py`), ID-specific search (`user_list_search.py`), and optional `clean_query.py`.
- **/save_state, /list_saved_states, /load_state, /delete_state:** Deterministic save/restore of domain bundles under `saved_states/`.
- **/clear_chat_history, /history_recent, /similar_questions:** Manage and query chat-history JSON with TF-IDF (word/char), Jaccard, and fuzzy similarity.

Appendix F.2. Background Task Flow

The task `process_detailed_combined_logic` realises the methodology:

1. Generate a UUID `session_id`; register an SSE queue.
2. Run `query_generation(input_text)`; if `USER_FLOW.query_cleaning.visible` is true and `clean_query.py` exists, refine the query list.
3. Collect candidates concurrently via `collect_articles(query_list)`; respect provider rate limits; filter by `id_list` if present.

4. Process and normalise results via `concurrent_organize_database_articles`; parse PDFs with `process_pdf_articles_parallel`; enrich references via `process_articles_by_url`.
5. Classify relevance via `concurrent_relevance_classification`; merge PDF items.
6. Trim to token budgets via `trim_relevant_articles_by_token_limit`.
7. Synthesize final answer via `generate_final_response`; extract citations with `print_referenced_articles`.
8. Stream intermediate updates and final JSON (`end_output`, `citations_obj`) over SSE.
9. If chat history is enabled in `user_env.js`, append the interaction to `historical_answer.json` with timestamps.

Appendix F.3. Error Handling and Resilience

Provider errors trigger retries (exponential backoff with jitter) and quarantine malformed items to avoid blocking the pipeline. SSE communicates degraded modes (partial results) to the user. Horizontal scaling is feasible with multiple FastAPI workers and stateless saved states.

Appendix F.4. Front-End Implementation

The client is a lightweight HTML/JS application with pages for main Q&A, configuration, and reference drill-down.

1. **APIcommunication(index.js,config.js)**: Submits questions and strategy toggles, opens SSE streams, and manages admin calls for prompts, environment, backend, and frontend configuration endpoints.
2. **Real-timeupdates(EventSource)**: Streams stage breadcrumbs and the final payload from `/sse?session_id=...`, keeping the UI responsive under variable provider latencies.
3. **Responseformatting(index.js/formatText)**: Renders the generated answer and a reference section; persists artefacts in `localStorage` (e.g., `generatedAnswer`, `rawOutput`, `citations`).
4. **PDFexport(index.js/generatePDF)**: Exports the question and answer with a timestamp via `jsPDF`.
5. **UImanagement**: Handles strategy selection (PDF upload / identifier search / article search), similar-question suggestions (`/similar_questions`), and recent history (`/history_recent`). Reference drill-down is provided via `reference.html` and `reference.js`. All toggleable features are controlled through the User Flow Configuration tab (see Appendix B.8).

Appendix F.5. Endpoint-to-Function Mapping

Table A3 lists the primary REST and SSE endpoints exposed by the FastAPI backend and the internal functions they invoke.

Table A3. API Endpoint to Function Mapping.

Endpoint	Purpose	Primary Function (Module)
GET/check_valid/{q}	Validity gate	<code>determine_question_validity</code> (helper_functions)
POST/process_detailed_combined_query	Start pipeline	<code>process_detailed_combined_logic</code> (main)
GET/sse	Stream updates	<code>event_generator</code> , <code>send_update</code> (main)

Table A3. *Cont.*

Endpoint	Purpose	Primary Function (Module)
GET/history_recent	Recent history	Reads historical_answer.json, sorts by timestamp (main)
GET/similar_questions	Suggest similar	Hybrid similarity: TF-IDF, Jaccard, fuzzy; min-heap selection (main)
POST/clear_chat_history	Clear history	Overwrites historical_answer.json with empty array (main)
GET/POST/fetch_/update_frontend_config	UI config	File I/O for user_env.js (main)
GET/POST/fetch_/update_env_config	Env keys	File I/O and client re-initialization (main)
GET/POST/fetch_/update_prompts_config	Prompts	Constants I/O (main)
GET/POST/fetch_/update_backend_config	Search helpers	Write user_search_apis.py, user_list_search.py, clean_query.py (main)
POST/save_state	Export bundle	Copy files to saved_states/ (main)
GET/list_saved_states	List bundles	Directory listing (main)
POST/load_state	Import bundle	Copy from saved_states/ (main)
POST/delete_state	Remove bundle	Directory delete (main)
POST/generate_code_endpoint	Code helper	generate_code_from_content (helper_functions)
POST/generate_prompt_endpoint	Prompt helper	generate_prompt_from_content (helper_functions)

Appendix G. Technical Documentation: Backend and Frontend

Appendix G.1. Backend

Routing, SSE (/sse), background tasks, provider wrappers, and utilities compose the service. Sessions are scoped by session_id. Security guardrails include environment-managed keys, production CORS, and recommended API gateways. New providers implement collect_articles and plug into prompts and UI.

Appendix G.2. Frontend

Baseline HTML/JS streams SSE updates. An optional React/Tailwind implementation can manage state (context/Redux), reconnect SSE hooks, and custom renderers for citations and code blocks while preserving the API contract.

Appendix G.3. Saved States and Models

Saved states checkpoint prompts, UI, and environment keys. Model selection (OpenAI, Gemini, Claude, or Ollama) is parameterised; restoring a state plus model/version pins reproducibility within model-drift bounds. For Ollama deployments the saved state also records the OLLAMA_MODEL and OLLAMA_BASE_URL variables so that the exact local model can be restored on any machine.

Appendix G.4. Migrating Between LLM Providers

CUSTOMNERD supports four LLM backends—OpenAI, Gemini, Claude, and Ollama—selectable via the LLM variable in variables.env. When switching from one provider to another, the following structured approach ensures consistency and optimal performance:

1. **Select the Closest Equivalent Model.** Identify the model from the new provider that most closely matches the capabilities and intended use case of the previous model (e.g., Gemini Pro as a replacement for GPT-4 for general-purpose tasks; or llama3.1:8b as a capable local replacement when moving to Ollama).
2. **Test Existing Prompts.** Input the original prompts into the new model without modification. Compare outputs for tone, clarity, and the absence of hallucinations or inconsistencies.
3. **Evaluate Performance.** If outputs are satisfactory, proceed with deployment. If hallucinations are present, continue to Step 4.
4. **Adjust Prompts for the New Model.** Research how the new model handles prompt structure, including syntax preferences, formatting cues (e.g., bold vs. caps for emphasis), and tolerance for verbosity. Tailor prompts accordingly. For example, Gemini responds better to concise instructions and markdown-style formatting; smaller Ollama models (e.g., phi3:mini, llama3.2) benefit from shorter, more directive prompts with explicit output format constraints.
5. **Tune Generation Parameters.** If prompt refinements do not resolve issues, lower temperature and top-*p* values to reduce randomness and increase determinism, then re-test outputs against prior benchmarks.
6. **Iterate and Lock.** Repeat Steps 3–5 until output quality is satisfactory and consistent. Once acceptable prompts and settings are found, document them in the saved state for reproducibility.

Ollama: Local LLM Deployment

Selecting LLM=Ollama routes all inference through a locally hosted Ollama server rather than a cloud API. This eliminates per-token billing and data-egress privacy concerns at the cost of requiring local compute.

CUSTOMNERD automates the entire local setup from the web interface: when LLM=Ollama is saved and a model is selected, the backend (/ollama_setup endpoint) sequentially checks for an existing Ollama installation, installs it if absent (Homebrew on macOS; curl install script on Linux), starts the ollama serve daemon, and executes ollama pull <model>—streaming live progress back to the browser via Server-Sent Events. A pre-flight status check (/ollama_status) runs immediately when the Ollama tab is opened, reporting whether Ollama is installed, whether the server is running, and whether the selected model has already been downloaded; this avoids redundant setup steps.

Table A4 lists the supported Ollama models grouped by category.

Table A4. Supported Ollama models in CUSTOMNERD (appendix reference).

Model	Params	Disk	Min RAM	Notes
<i>Recommended</i>				
llama3.2	3.2B	~2 GB	4 GB	Default; best balance of speed and quality
llama3.1:8b	8B	~4.7 GB	8 GB	Better reasoning; 128K context
<i>Small / Low-Resource</i>				
llama3.2:1b	1B	~1.3 GB	3 GB	Fastest; minimal RAM
qwen3:1.7b	1.7B	~1.1 GB	3 GB	Alibaba Qwen, very compact
deepseek-r1:1.5b	1.5B	~1.1 GB	3 GB	Reasoning on low-end hardware
phi3:mini	3.8B	~2.3 GB	4 GB	Microsoft Phi-3; punches above weight
gemma3:4b	4B	~2.5 GB	4 GB	Google Gemma 3 compact

Table A4. Cont.

Model	Params	Disk	Min RAM	Notes
High Quality (8 GB+ RAM recommended)				
qwen3:8b	8B	~5 GB	8 GB	Best reasoning at 8B class
mistral	7B	~4.1 GB	8 GB	Versatile general purpose
gemma2:9b	9B	~5.4 GB	8 GB	Google Gemma 2
gemma3:12b	12B	~7.7 GB	16 GB	Google Gemma 3 flagship
mistral-nemo:12b	12B	~7 GB	16 GB	Extended 128K context
High Quality (8 GB+ RAM recommended)				
phi4	14B	~8.9 GB	16 GB	Microsoft Phi-4; top-tier quality
Reasoning				
deepseek-r1:7b	7B	~4.7 GB	8 GB	Chain-of-thought reasoning
deepseek-r1:14b	14B	~8.9 GB	16 GB	Deep multi-step reasoning
Code				
codellama:7b	7B	~3.8 GB	8 GB	Meta Code Llama
deepseek-coder:6.7b	6.7B	~3.8 GB	8 GB	DeepSeek Coder
qwen2.5-coder:7b	7B	~4.4 GB	8 GB	Best coding model in list
Cloud / Hosted via Ollama				
kimi-k2.5:cloud	Cloud	None	None	Moonshot Kimi K2.5; no local download

Appendix H. Implementation Workflow: From Zero to Domain in Hours

To implement a new domain (e.g., “LegalNerd” for legal research), domain specialists follow a structured workflow that typically requires 2–3 h. Table A5 presents the step-by-step process with associated time estimates.

Table A5. Domain Implementation Workflow

Step	Time	Activities
Frontend Configuration	15 min	Set site name, upload logo, choose icon, write tagline, customize colors and fonts, compose a domain-specific disclaimer
Environment Setup	5 min	Add required API keys (e.g., Westlaw, LexisNexis), select an AI provider (OpenAI, Gemini, Claude, or Ollama for fully local inference)
Search Integration	30–60 min	Configure the primary search function (custom code or AI-assisted generator), optionally enable identifier-based search, and configure query refinement for citation normalization
AI Prompt Customization	60–90 min	Customize prompts for question validation, query generation, relevance classification (e.g., precedential value, jurisdiction), article processing (case facts, holdings, reasoning), and response synthesis (Bluebook citations and legal disclaimers). An AI-assisted prompt generator can be used to accelerate this step
User Flow Configuration	10 min	Customize search strategy labels and tooltips, configure visibility toggles (including Deep Search /tiered_collect_articles), and enable or disable query cleaning, references, and chat history; inline SVG diagrams explain each toggle
State Saving	2 min	Save the complete configuration as a named state for reproducible deployment, team sharing, and future restoration
Total	2–3 h	Fully functional, production-ready domain implementation

This demonstrates that fully functional, production-ready domain implementations can be created in hours rather than weeks, without requiring software engineering expertise or modifications to the core system.

Appendix I. CloudNerd: Multi-Source Retrieval and Query Refinement

This appendix provides implementation details for CloudNerd retrieval: the baseline configuration emphasizes Stack Overflow via the Stack Exchange API; the CloudNerd Advanced saved state extends the same `collect_articles` contract with AWS, Azure, and CloudFest RSS (plus related search and resource links) and parallel HTML text extraction. Excerpts below reflect the Advanced module; the code in the repository is executed at runtime and is included for reproducibility and auditability.

Appendix I.1. CloudNerd Retrieval Configuration (120-Question Benchmark)

Benchmark evaluation uses CloudNerd rather than the Advanced RSS bundle: multi-plane tiered Stack Overflow retrieval (`tiered_collect_articles`), citation recovery, supplemental gap-fill, and the evidence-constrained synthesis prompt. Pipeline architecture, retrieval profiles, and User Flow flags are documented in Appendix J. The Advanced `collect_articles` implementation below remains the reference for multi-source saved-state deployments.

Appendix I.2. CloudNerd Configuration Walkthrough

This subsection documents the CloudNerd-specific configuration choices made through the CUSTOMNERD web interface. It complements the generic interface descriptions in Appendix B and the prompt texts in Appendix C.1.

Appendix I.2.1. Frontend Visual Identity

The visual identity was configured through the Frontend Configuration Tab (see Appendix B.5) to reflect cloud computing aesthetics while maintaining professional clarity:

Branding Elements:

- **Site Name:** “CloudExpert Nerd”—chosen to emphasize expertise in cloud technologies
- **Site Icon:** cloud emoji (Unicode U+2601)—Cloud Emoji—provides instant visual recognition of the domain focus
- **Site Tagline:** “We answer your cloud-related questions based on Stack Overflow” (baseline); the Advanced retrieval bundle additionally ingests vendor blogs and industry feeds while preserving the same UI and pipeline shape
- **Question Placeholder:** “Enter your cloud-related question here. . . .”—guides users toward appropriate query formulation

Visual Styling:

- **Background Color:** #fafafa (light grey)—neutral backdrop that reduces eye strain during extended technical reading
- **Font Family:** 'Rubik', sans-serif—modern, readable sans-serif optimized for technical content
- **Submit Button:** #007bff (standard blue)—recognizable action color consistent with cloud provider interfaces

Disclaimer Text: A comprehensive educational disclaimer was configured emphasizing that “Everything on this website is for educational and informational purposes only. The content is based on publicly available information, including community discussions and third-party sources, and may not always reflect the most current practices or official

guidance. It is not intended to replace professional consultation, official cloud provider documentation, or vendor-specific support.”

Appendix I.2.2. Backend Prompt Configuration Details

All five prompt categories were customized for technical troubleshooting. The full prompt texts are provided in Appendix C.1; this section documents the configuration choices for each category.

1. Question Validation Prompt:

The DETERMINE_QUESTION_VALIDITY_PROMPT was configured with comprehensive classification rules to identify cloud-relevant queries while filtering cross-domain contamination. The prompt recognizes cloud technology topics including:

- Cloud computing platforms: AWS, Azure, GCP, OCI, DigitalOcean
- Cloud services: S3, EC2, Lambda, Azure Functions, GKE, Pub/Sub
- Infrastructure-as-code: Terraform, Pulumi, ARM, CloudFormation
- Container technologies: Kubernetes, Docker, container orchestration
- DevOps practices: CI/CD pipelines, observability, monitoring, scaling, serverless
- Cloud networking: VPC, load balancers, DNS, CDN, firewalls
- Security and compliance: IAM, multi-cloud strategies, architecture decisions

The prompt implements granular rejection categories to maintain domain boundaries: "False - General Tech" for non-cloud programming questions (frontend coding, mobile apps, gaming), "False - Personal" for lifestyle queries, "False - Health" for medical questions, "False - Recipe" for cooking, "False - Animal" for pet advice, and "False - Other" for unclassifiable topics. This multi-category approach prevents the system from attempting to answer questions better suited for other CUSTOMNERD domains. Appendix C.1.1.6 contains the prompt text.

2. Query Generation and Expansion:

CloudNerd’s GENERAL_QUERY_PROMPT generates JSON-formatted query lists rather than Boolean search strings. The prompt is instructed to create 8–12 variations combining exact rephrases, synonyms, provider-specific terminology, and service-specific versions.

For example, given the user question “How to secure a Kubernetes cluster?”, the system generates:

```
{
  "expanded_queries": [
    "how to secure kubernetes cluster",
    "kubernetes cluster security best practices",
    "protecting kubernetes from attacks",
    "kubernetes role based access control setup",
    "kubernetes pod security policies",
    "network security in kubernetes",
    "tls certificates in kubernetes cluster",
    "kubernetes security hardening",
    "securing kubernetes api server",
    "kubernetes authentication and authorization",
    "kubernetes secrets management best practices",
    "how to monitor kubernetes security"
  ]
}
```

These variations systematically explore different security dimensions (RBAC, network policies, encryption, and monitoring) without requiring users to understand the technical breakdown. The JSON format enables programmatic extraction via the query refinement pipeline. Appendix C.1.1.10 contains the prompt text.

3. Relevance Classification:

The RELEVANCE_CLASSIFIER_PROMPT was configured to evaluate Stack Overflow posts along two dimensions: technical utility and security value. The prompt instructs the LLM to return "yes" if a post contains relevant technical information to answer the query or mentions important safety, security, or reliability concerns. Posts focusing on unrelated technologies (frontend UI frameworks, desktop applications, gaming engines) are rejected with "no".

Example behavior:

- **Query:** "How to configure IAM roles in AWS Lambda?"
Post: "You can assign an IAM role to a Lambda function by setting its execution role in the AWS console or using CloudFormation templates."
Classification: yes (directly addresses the query with actionable steps)
- **Query:** "What are best practices for securing Kubernetes clusters?"
Post: "Enabling RBAC and using network policies are essential steps. Misconfigured permissions often lead to breaches."
Classification: yes (provides best practices and security warnings)
- **Query:** "How to set up CI/CD for serverless apps?"
Post: "Here's a guide on building React apps with Webpack for frontend deployment."
Classification: no (discusses frontend tooling, not cloud CI/CD)

Appendix C.1.1.14 contains the prompt text.

4. Article Processing and Information Extraction:

CloudNerd uses two specialized prompts for processing retrieved evidence (Stack Overflow Q&A and, in Advanced, vendor or industry blog text shaped like answers):

Article Type Classification: The ARTICLE_TYPE_PROMPT distinguishes between questions (problem statements, requests for help) and answers (solutions, explanations, troubleshooting steps). This classification enables differential processing: questions provide context about common problems, while answers contain actionable solutions.

Structured Information Extraction: The ABSTRACT_EXTRACTION_PROMPT was configured to extract 13 categories of technical information from Stack Overflow posts and analogous bodies from feed-derived pages:

1. **Problem & Context:** Environment details, cloud provider, service names, versions, error messages
2. **Proposed Solutions:** All approaches discussed in the answers
3. **Best Answer / Accepted Fix:** Steps, commands, or configuration details from the accepted or highest-voted answer
4. **Risks & Limitations:** Cost issues, scaling problems, security vulnerabilities, deprecated features
5. **Benefits & Advantages:** Performance improvements, cost savings, reliability gains
6. **Cloud Service(s) Involved:** Specific services (EC2, S3, IAM, AKS, GKE, Cloud Run, Terraform)
7. **Tools, Libraries, or Frameworks:** Terraform, Ansible, Helm, Docker, CI/CD tools, SDK versions
8. **Configuration Details:** YAML snippets, JSON configs, environment variables, CLI commands
9. **Performance Metrics or Benchmarks:** Scaling numbers, latency improvements, throughput, cost savings

10. **Errors or Debugging Notes:** Error codes, stack traces, logs, common pitfalls
11. **Security or Compliance Considerations:** IAM permissions, encryption, RBAC, firewall rules
12. **Alternative Approaches:** Other solutions discussed, compared, or rejected
13. **Community Signals:** Acceptance status, upvote counts, author reputation, consensus indicators

This structured extraction ensures that technical details, code snippets, and configuration parameters are preserved rather than summarized away, maintaining the actionability of Stack Overflow solutions.

5. Response Synthesis and Citation:

The FINAL_RESPONSE_PROMPT governs how CloudNerd assembles Stack Overflow evidence into comprehensive technical answers. Key configuration elements include:

Evidence Selection Criteria: The prompt requires selecting 8–20 posts, leaning toward more rather than fewer to ensure comprehensive coverage. It prioritizes high-quality posts (accepted answers, highly-upvoted answers, answers with reproducible code/configurations, answers referencing official documentation) while excluding weak evidence (speculation, opinion-only answers, solutions without reproducible steps).

Security-First Synthesis: For questions involving security risks, cost traps, or harmful misconfigurations, the prompt instructs the system to focus on deterrence and warnings rather than enabling dangerous practices. The goal is for CloudNerd to emphasize security best practices and compliance considerations.

Multi-Provider Coverage: The prompt encourages including solutions across different cloud providers (AWS, Azure, GCP) and frameworks (Kubernetes, Terraform, Docker) when relevant, providing users with comprehensive options rather than single-vendor solutions.

Accessibility Requirement: Technical concepts must be explained in ways that junior engineers can understand, balancing technical precision with pedagogical clarity.

Example output structure from the prompt:

```
User Query: What is the best way to secure Kubernetes
cluster access?

AI Answer:
Securing Kubernetes cluster access requires combining
identity
management, role-based access control (RBAC),
and network-layer
protections, based on multiple solutions provided in
the evidence.

Access Control & Identity Management
Several posts emphasize enabling RBAC and assigning
least-privilege
roles, ensuring no user has cluster-admin by default [1][2].
Using OpenID Connect (OIDC) with external identity providers
(Google, Azure AD, Okta) is repeatedly recommended [3].

Network Policies & Encryption
Multiple answers recommend implementing Network Policies to
restrict
```

```
pod-to-pod communication [4][5]. Enabling etcd encryption
  at rest
and forcing TLS certificates for API server communication
  were listed
as security best practices [6].
```

Risks & Trade-offs

```
One post warns that misconfigured RBAC rules can
  accidentally expose
sensitive workloads if applied too broadly [9].
```

References:

```
[1] Stack Overflow. "How to use Kubernetes RBAC..."
  https://stackoverflow.com/q/xxxxxx
[2] Stack Overflow. "Best practices for RBAC..."
  https://stackoverflow.com/q/yyyyyy
...
```

Appendix C.1.2.13 contains the prompt text.

Appendix I.2.3. Environment Configuration

The Environment Configuration Tab (see Appendix B.7) was used to set up external service integrations:

LLM Provider Selection: The system was configured to use **OpenAI** as the primary LLM provider by setting `LLM="OpenAI"` in `variables.env`. This selection determines which API client is initialized for all prompt processing.

API Keys Configuration:

- `OPENAI_API_KEY`: Required for GPT-4 API access. This key enables all five prompt categories (validation, query generation, relevance classification, article processing, response synthesis) to function.
- `STACK_API_KEY`: Optional but recommended for production use. Without this key, the Stack Exchange API limits requests to 300 per day; with the key, the limit increases to 10,000 requests per day. For a system processing 50 questions daily with 12 query variations each, this higher limit is essential for stable operation.

Appendix I.2.4. User Interface Search Strategy Toggles

The User Flow configuration of CUSTOMNERD exposes multiple retrieval options through toggleable search strategies (see Appendix B.8). The settings chosen for Cloud-Nerd are:

- **Upload Articles:** `visible: false`—PDF uploads disabled, as Stack Overflow provides sufficient technical coverage for this domain.
- **Insert ID:** `visible: false`—Identifier-based search disabled, as it is not applicable to cloud troubleshooting.
- **Search Stack Exchange:** `visible: true, defaultChecked: true`—Primary retrieval mechanism, enabled by default and labeled `"SearchusingStackExchangeArticles"`. Additional interface features were enabled to support transparency and usability:
- **Reference Section:** `visible: true`—Displays Stack Overflow citations for source verification.

- **Chat History:** `visible: true`—Preserves conversational context across troubleshooting sessions.
- **Query Cleaning:** `visible: true`—Activates JSON preprocessing for expanded query lists via `clean_query.py`.
- **Deep Search:** `cascade_retrieval.visible: true`—Enables `tiered_collect_articles()` (planes P1–P5; Stack Overflow benchmark: P1 strict, P2 title-only, P4 extended-keyword). Configuration tab label **DeepSearch**.

Appendix I.2.5. Saved State File Listing

After configuring all components through the web interface, the CloudNerd application designer saved the setup as a named state using the Save State interface (see Appendix B.4.2). The repository includes `saved_states/CloudNerd/` (compact Stack Overflow-only retrieval) and `saved_states/CloudNerd Advanced/` (multi-source retrieval as in subsection I.3). A typical bundle contains eight files; line counts and byte sizes vary slightly by export:

1. `user_env.js`: Frontend configuration including site name, icon, tagline, colors, disclaimer text, and all UI toggles
2. `openai_prompts.py`: All five customized prompts with cloud-specific examples and validation rules
3. `user_search_apis.py`: Stack Exchange integration with HTML cleaning; the Advanced variant adds RSS (`feedparser`), Azure/AWS/CloudFest discovery, parallel page fetch, and resilient Stack API helpers (on the order of hundreds of lines)
4. `clean_query.py`: Single-function query parser (`clean_query(query_list, max_queries=None)`)
5. `user_list_search.py`: Minimal PMID lookup (retained for framework compatibility)
6. `variables.env`: API keys and LLM provider selection (optional `RATE_LIMIT_CURL_RESILIENCE` for global HTTP resilience)
7. `customnerd_logo.png`: Cloud-themed logo image
8. `historical_answer.json`: Conversation history cache (if enabled)

Appendix I.3. CloudNerd retrieval: Stack Overflow APIs and cloud RSS

The retrieval component implements a **Stack Exchange** stage aligned with Stack Overflow's data model, plus a **cloud RSS and page-fetch** stage. In the first stage, the Stack Exchange search API retrieves candidate questions for each expanded query. In the second stage, the highest-ranked answers for each unique question are fetched with the `withbody` filter, normalized, and prepared for downstream relevance classification and synthesis. A further stage gathers URLs from (i) <https://aws.amazon.com/blogs/aws/feed/>, (ii) <https://azure.microsoft.com/en-us/blog/feed/> and Azure blog HTML search, (iii) <https://www.cloudfest.com/feed/>, plus optional AWS site search and AWS resource-hub pages for selected tokens (e.g., Lambda, ECS, Kubernetes-related mappings). Entries are filtered by query-relevant token expansion; if nothing matches, recent feed items may be used. URLs are deduplicated, search-hub listing pages are skipped for body fetch, and article HTML is retrieved in parallel; visible text is stored in `answer_body` so records match the Stack Overflow shape (`question_title`, URLs, optional `cloud_news_source`).

The number of Stack Overflow answers retained is controlled by `answer_counter`; cloud breadth per provider is controlled by `cloud_links_per_source`, with a cap on total unique cloud URLs merged after all queries (e.g., `min(30, max(6, [answer_counter/2]))` in the reference implementation).

Central HTTP resilience.

Shared logic in `helper_functions.py` (activated by default at import, overridable via `RATE_LIMIT_CURL_RESILIENCE` in `variables.env`) patches `requests.get` and `url-lib.request.urlopen`. On HTTP 429, 409, 503 or Stack Exchange JSON throttle indicators, the backend may re-fetch the *same* URL using `curl`. The FastAPI pipeline wraps normal search with `collect_articles_with_curl_fallback`. `user_search_apis.py` adds in-module retries on 429/5xx, defensive JSON parsing (so empty or non-JSON bodies after throttling do not raise decode errors), honoring of API backoff fields, skipping of HTTP 400/404 answer fetches for invalid or removed threads, and pacing between queries and between questions.

```
# Excerpt: resilient Stack Exchange helpers and
collect_articles outline.
# The full module also defines search_all_rss, feedparser
usage,
# parallel page fetch, and ~HTML text extraction for
cloud~URLs.

STACK_API_TIMEOUT = 35
STACK_INTER_QUERY_SLEEP = 2.0
STACK_BETWEEN_ANSWER_SLEEP = 0.25

def _stack_exchange_get(url, params, *, retries=5):
    delay = 2.0
    last = None
    for _ in range(retries):
        last = requests.get(
            url, params=params,
            headers={**HEADERS, "Accept":
                "application/json"},
            timeout=STACK_API_TIMEOUT,
        )
        if last.status_code == 429:
            wait = int(last.headers.get("Retry-After",
                delay))
            time.sleep(max(wait, delay))
            delay = min(delay * 1.5, 60)
            continue
        if last.status_code in (502, 503):
            time.sleep(delay)
            delay = min(delay * 1.5, 60)
            continue
        return last
    return ~last

def _stack_exchange_payload(response):
    if response is None or not (response.text or
        "").strip():
        return None
    try:
```

```

        data = response.json()
    except ValueError:
        return None
    return data if isinstance(data, dict) else ~None

def _stack_apply_backoff(data):
    if data and "backoff" in data:
        time.sleep(float(data["backoff"]))

def collect_articles(query_list, answer_counter=50,
                    cloud_links_per_source=10):
    """Stack Overflow + AWS/Azure/CloudFest RSS and page
       text."""
    collected = []
    seen_questions = set()
    search_url =
        "https://api.stackexchange.com/2.3/search/advanced"
    answers_url =
        "https://api.stackexchange.com/2.3/questions/{ids}/
        answers"
    # ... per-query search; parse with
    #   _stack_exchange_payload;
    #   skip on error_id or empty JSON;
    #   _stack_apply_backoff(data)
    # ... per-question answers; skip 400/404; append
    #   clean_html bodies
    # ... time.sleep(STACK_BETWEEN_ANSWER_SLEEP);
    #   STACK_INTER_QUERY_SLEEP
    stack_slice = collected[:answer_counter]
    cloud = [] # from search_all_rss + parallel GET +
    #   BeautifulSoup text
    return stack_slice + cloud

```

Design Rationale.

- **Two-stage Stack retrieval:** Separates question discovery from answer ranking, prioritizing community-validated solutions.
- **Deduplication:** The seen_questions set prevents redundant processing when multiple query variants match the same question; cloud URLs are deduplicated across queries.
- **Metadata preservation:** Rich metadata enables citation generation, quality scoring, and explainable synthesis.
- **Resilience and pacing:** In-module retries, safe JSON parsing, API backoff compliance, and inter-request sleeps reduce throttling failures; global curl fallback still applies to patched requests.get.
- **Cloud supplement:** RSS and page fetch broaden evidence beyond Q&A while reusing the same downstream classification and extraction prompts.
- **Configurable scope:** answer_counter and cloud_links_per_source trade coverage against latency and tokens.

Appendix I.4. Query Refinement and JSON Sanitization

Query expansion prompts produce structured JSON output containing multiple rewritten search queries. This preprocessing stage extracts the expanded query list and converts it into a plain-text format suitable for execution against the Stack Exchange API. Defensive parsing ensures graceful degradation in the presence of malformed LLM output.

```
def clean_query(query_list, max_queries=None):
    """
    Turn query_generation output into a flat list of search
    strings.
    Parses JSON expanded_queries or nested plain-text lists.
    Optional max_queries caps how many strings are returned.
    """
    if not query_list:
        return []
    queries = _parse_expanded_queries(query_list)
    if not queries:
        queries = _flatten_query_list(query_list)
    if max_queries is not None and max_queries > 0:
        return queries[:max_queries]
    return queries
```

Design Rationale.

- Enables programmatic extraction of LLM-generated query expansions.
- Prevents pipeline failures due to malformed or partially structured JSON.
- Ensures downstream retrieval operates on clean, normalized query strings.

Appendix J. CloudNerd Evaluation Pipeline

This section documents the CloudNerd evaluation configuration deployed for benchmark runs. It replaces the CloudNerd Advanced multi-source RSS bundle while preserving the same downstream relevance classification, extraction, and synthesis contract. Repository filenames retain internal version suffixes for reproducibility; main-text nomenclature follows Section 8.2.

Appendix J.1. Retrieval and synthesis flags (120-question benchmark)

Multi-plane tiered Stack Overflow retrieval is controlled by two layers:

1. **Code profile:** `set_active_nerd_profile(state_name)` on `load_state` loads `RETRIEVAL_NERD_PROFILES["CloudNerd"]` from `helper_functions.py` (mode: "cascade", domain lexicon, plane budgets, default planes P1–P5). Unregistered saved states fall back to `RETRIEVAL_DEFAULTS` (mode: "legacy").
2. **User Flow toggle:** `cascade_retrieval.visible` in `user_env.js` (Configuration tab: *Deep Search*) must be true; otherwise `is_cascade_mode()` returns false and the pipeline uses legacy single-pass `collect_articles()` regardless of profile.

Legacy `RETRIEVAL_*` environment variables remain supported when no registered Nerd profile matches the loaded state name; they are no longer required for CloudNerd or DietNerd deployments.

- Multi-plane tiered retrieval enabled (`tiered_collect_articles`, profile + Deep Search toggle)

- CITATION_RESCUE_ENABLED=true—TF-IDF rescue for posts dropped by the relevance classifier
- RETRIEVAL_GAP_FILL_ENABLED=true—supplemental Stack Overflow search when coverage is thin (also plane P5 inside tiered retrieval)
- ENABLE_QUERY_FALLBACK=false; FAITHFULNESS_TWO_PASS_ENABLED=false
- FINAL_RESPONSE_TEMPERATURE=0.2; evaluated synthesis prompts (Appendix C.2)

API keys template (variables.env).

```
OPENAI_API_KEY="your - openai - key - here "
STACK_API_KEY="your - stack - exchange - key - here "
LLM="OpenAI "
```

Appendix J.2. Multi-Plane Tiered Retrieval (*tiered_collect_articles*)

When Deep Search is enabled, `main.py` invokes `tiered_collect_articles(collect_fn, input_text, p1_query_list)` in `helper_functions.py`. The function merges results across enabled planes (default P1–P5), tags each article with its `retrieval_plane`, applies quality gates between planes, optionally short-circuits when coverage is sufficient, and returns (articles, metadata) with `planes_run`, `queries_per_plane`, `retrieval_confidence`, and per-plane quality-gate diagnostics.

Retrieval planes.

1. **P1 (precision):** Queries from `clean_query()` over LLM-expanded strings. On Stack Overflow this is the *strict* tier (full body/tag match).
2. **P2 (relaxed):** Title- and keyword-oriented variants from `_cascade_generate_queries()` on the resolved question title. Maps to the *title-only* tier on Stack Overflow.
3. **P3 (facet):** Re-runs P1/P2 seeds with a domain facet—CloudNerd lexicon tokens or DietNerd [Humans] PubMed suffix. Contributes domain-scoped recall before broad expansion.
4. **P4 (broad):** Additional batches from `GENERAL_QUERY_PROMPT`, a cascade broad-query prompt, and deterministic backup strings. Maps to *extended-keyword* expansion on Stack Overflow.
5. **P5 (gap-fill):** Queries built from key terms absent in the merged corpus (`build_gap_fill_queries`), overlapping with supplemental `retrieval_gap_fill.py` logic used in benchmark runs.

On the 120-question CloudNerd benchmark, the best plane before stopping was P1 (strict) on 84 questions, P2 (title-only) on 33, and P4-class extended-keyword on 3; plane P5 and post-cascade gap-fill activated on additional questions when coverage remained thin.

Entry point (excerpt).

```
def tiered_collect_articles(collect_fn, input_text,
    p1_query_list, *, title_text=None):
    """
    Run planes P1->P5. Returns (articles, metadata).

    metadata keys: retrieval_mode, planes_run, planes_log,
    queries_per_plane, retrieval_confidence, quality_gate
    """
```

```
# P1 precision -> P2 relaxed -> P3 facet -> P4 broad ->
P5 gap-fill
# Quality gates may short-circuit when
passes_retrieval_quality_gate() succeeds
```

Profile registry (excerpt).

```
RETRIEVAL_NERD_PROFILES = {
    "DietNerd": {"mode": "cascade", "facet_suffix":
        "[Humans]"},
    "CloudNerd": {"mode": "cascade", "lexicon":
        "aws,amazon,azure,..."},
    "NewsNerd": {"mode": "legacy"},
}
```

Appendix J.3. Supplemental Retrieval (*retrieval_gap_fill.py*)

After tiered retrieval (planes P1–P5) completes, supplemental retrieval triggers one additional Stack Overflow search when (i) strict-tier (P1) results contain ≤ 2 articles, or (ii) key terms from the user question are absent from retrieved post text. Supplemental retrieval activated on 36 of 120 test questions. An expanded supplemental-retrieval criterion (paired with an alternative prompt variant) failed the development-set quality gate and was not deployed.

Trigger logic and retrieval (*should_gap_fill, gap_fill_retrieval*).

```
def should_gap_fill(articles, query, title, cascade_meta):
    meta = cascade_meta or {}
    terms = extract_key_terms(query, title)
    miss = missing_terms(terms, articles)

    thin_strict = (
        meta.get("retrieval_mode") == "strict" and
        len(articles) <= 2
    )
    low_coverage = bool(terms) and len(miss) / len(terms)
        >= 0.5

    if thin_strict or low_coverage:
        return True, miss
    return False, miss

def gap_fill_retrieval(articles, query, title, max_date,
    collect_fn, cascade_meta=None):
    trigger, miss = should_gap_fill(articles, query, title,
        cascade_meta)
    if not trigger:
        return articles, {"gap_fill_used": False,
            "missing_terms": miss}
```

```

queries = build_gap_fill_queries(query, title, miss)
extra = collect_fn(queries, max_date=max_date) or []
merged = merge_articles(articles, extra)
return merged, {
    "gap_fill_used": True,
    "gap_fill_queries": queries,
    "gap_fill_added": max(0, len(merged) -
        len(articles)),
}

```

Appendix J.4. Citation Pipeline Rescue (*citation_rescue.py*)

The relevance classifier can drop posts that still contain usable technical content (e.g., Docker, Kubernetes, Terraform, LibreOffice deployment threads). When citation recovery is enabled, TF-IDF overlap between the user question and dropped posts re-inserts the best-matching articles before synthesis. Combined with a softer RELEVANCE_CLASSIFIER_PROMPT (Appendix C.2), this raised the development-set scorable rate from 6/20 to 20/20 (Table 14).

TF-IDF rescue and citation backfill (*rescue_relevant_articles*, *ensure_citations*).

```

def rescue_relevant_articles(articles, user_query, *,
    top_k=8):
    """Keep top-k organized articles by TF-IDF similarity
        when relevance filter returns none."""
    vectorizer = TfidfVectorizer()
    matrix = vectorizer.fit_transform(texts + [user_query])
    query_vec = matrix[-1]
    scores = cosine_similarity(matrix[:-1],
        query_vec).flatten()
    ranked = sorted(zip(scores, articles), key=lambda x:
        x[0], reverse=True)
    return [article for score, article in ranked[:top_k] if
        score > 0]

def ensure_citations(final_output, articles, citations):
    """Fill citations when reference parsing failed but
        organized articles exist."""
    if citations:
        return citations
    if not articles:
        return []
    print("[CITATION_RESCUE] Forcing citations from
        organized articles ...")
    return build_citations_from_articles(articles)

```

Appendix J.5. Evidence-Constrained Synthesis Prompt

The FINAL_RESPONSE_PROMPT enforces direct-answer-first structure, never-refuse behavior when evidence is non-empty, and four grounding rules (no invented config keys;

reuse question terms in sentence 1; no cross-snippet synthesis unless one post states the conclusion; gap statements last). The complete prompt text appears in Appendix C.2.

Appendix K. Questionnaire-Based Personalization

This appendix shows the prompts used to fill in a questionnaire given that a query has mentioned a disease. The user will then be queried about those risk factors.

Appendix K.1. Large Language Model Prompts

```
You are given a text describing disease risk factors. Your
task is to extract only the most common and high-risk
factors, excluding redundant or weakly associated ones.

Format your output strictly as concise questionnaire items:
[risk factor]: [short categorical options]

Rules:
1. No personal pronouns or directional words (e.g., "you",
"your", "patient").
2. Each option must be short - ideally one or two words
(e.g., "yes / no", "25-35").
3. Focus on strong, well-established risk factors only.
4. Use clear, medically neutral language.
5. Output only the final questionnaire lines.
No explanation.

Example Output:
- Family history of AATD: yes, no
- Family history of COPD or emphysema: yes, no
- Family history of liver disease: yes, no
- Cigarette smoking duration (years): <5, 5-10, 10-20, >20
- Cigarette smoking intensity (packs/day): <1, 1-2, >2

Now, given the following text, output the questionnaire
```

Appendix K.2. Example Outputs Presented to Users Based on Different Answers to Different Questionnaires

```
Question 1: Risk factors for miscarriage
Questionnaire 1:

Age: 40-45
Past Miscarriages: no
Long-Term Conditions: yes
Uterine Or Cervical Problems: no
Use Of Tobacco: yes
```

Output 1

Analysis and Recommendations: Based on the available research, several risk factors are associated with an increased likelihood of miscarriage, which are particularly relevant given your age group (40-45 years old) and tobacco use. Here's a detailed look at these factors:

Maternal Age: Research consistently shows that maternal age is a significant risk factor for miscarriage. The risk of miscarriage increases with age, particularly after age 35, and even more so after age 40. One study found that the risk of fetal loss was significantly high in women aged 30 years and above, with a notable increase after age 40 [4]. Another study highlighted that miscarriage risk reaches up to 53% at age 45 and over.

Tobacco Use: Smoking is widely recognized as a risk factor for miscarriage. The chemicals in tobacco can affect the cervix and the health of your eggs, increasing the risk of miscarriage. Although the specific articles reviewed do not focus on tobacco use, the general medical consensus supports this risk factor.

Long-term Health Conditions: Depending on the nature of your long-term conditions, these could potentially influence miscarriage risk. Conditions such as diabetes, thyroid disorders, and hypertension are known to increase miscarriage risks if not well-managed during pregnancy.

Previous Pregnancy Outcomes: While you mentioned no history of past miscarriages, other adverse outcomes in previous pregnancies (if any), such as stillbirth or preterm delivery, can also increase the risk of miscarriage in subsequent pregnancies.

Given these factors, it's crucial to discuss your specific health profile and any long-term conditions with a healthcare provider who can offer personalized advice and management strategies. They might recommend lifestyle adjustments, closer monitoring during pregnancy, or specific medical interventions to mitigate these risks.

In summary, advancing maternal age and tobacco use are significant risk factors for miscarriage in your case. Managing long-term health conditions effectively and seeking regular prenatal care are essential steps in

reducing the risk of miscarriage. Always consult with a healthcare professional for guidance tailored to your specific health needs.

Questionnaire 2:

Age: under 35
Past Miscarriages: yes
Long-Term Conditions: no
Uterine Or Cervical Problems: yes
Use Of Tobacco: no

Output 2

Analysis and Recommendations: Based on the analysis of several studies, various risk factors have been identified that contribute to the likelihood of experiencing a miscarriage. Given your specific condition, here are the relevant factors:

Maternal Age: Although you mentioned being under 35, it's important to recognize that increased maternal age is generally associated with higher miscarriage rates. However, women under 35 generally have lower risk compared to older age groups.

History of Past Miscarriages: Having had previous miscarriages is a significant risk factor. Studies show a clear pattern where the risk of miscarriage increases with the number of prior miscarriages.

Uterine or Cervical Problems: Structural problems with the uterus or cervix can increase the risk of miscarriage. While specific details about these conditions weren't covered in the evaluated studies, medical literature widely acknowledges this as a risk factor.

Paternal Age: Advanced paternal age (over 40 years) is also linked to increased risks of miscarriage. This might be relevant depending on the age of your partner.

Other Identified Factors: Additional factors identified include obesity, high parity (having had many pregnancies), and previous cesarean deliveries. While these may not apply directly to you since you haven't mentioned them, they are important for a comprehensive understanding of the risks.

Considering these factors, here are some suggestions for managing and potentially mitigating the risk of miscarriage:

Regular Medical Follow-up: Given your history of miscarriages and uterine or cervical issues, regular check-ups with a healthcare provider specializing in maternal-fetal medicine can help manage your risk.

Healthy Lifestyle: Maintaining a healthy weight, managing stress, and ensuring adequate nutrition can support pregnancy health.

Preconception Counseling: Discussing your full medical history with a healthcare provider can provide tailored advice and potential interventions to reduce risk.

In conclusion, while certain factors like past miscarriages and uterine or cervical problems increase your risk, proactive medical care and lifestyle management can help mitigate some of these risks. It is essential to work closely with your healthcare provider to monitor and manage your pregnancy based on your specific health profile.

Question 2: Suggestions for heart disease
Questionnaire 1:

Age: <55
Gender: female
Family history of heart disease: no
Smoke: yes
Diet With High Sugar Or Fat: yes

Output~1

Analysis and Recommendations: Given your profile as a female under 55, who smokes and has a diet high in sugar or fat, it is crucial to address these risk factors to effectively manage and prevent heart disease. Here are some personalized suggestions based on robust research findings:

Adopt a Mediterranean Diet: Research shows that the Mediterranean diet, which is rich in fruits, vegetables, whole grains, and healthy fats (mainly from olive oil), can significantly reduce the risk

of cardiovascular diseases (CVDs), including heart attacks and strokes. This diet also supports longevity and reduces the overall mortality from heart disease.

Incorporate Plant-Based Foods: Studies indicate that plant-based diets can lower the risk of coronary heart disease and reduce factors such as body mass index (BMI), cholesterol, and markers of inflammation. However, it's important to note that while plant-based diets are associated with lower cardiovascular mortality, they may carry a higher risk of certain types of strokes. It would be beneficial to balance your diet with a variety of foods to mitigate this risk.

Reduce Smoking: Smoking is a significant risk factor for CVD. Quitting smoking can drastically reduce your risk of heart disease and improve overall cardiovascular health. Consider seeking support through smoking cessation programs which can offer guidance and increase your chances of success.

Manage Dietary Fats and Sugars: High intake of fats and sugars, especially trans fats and added sugars, can contribute to heart disease. Focus on reducing these in your diet and replace them with healthier alternatives like whole grains and healthy fats (e.g., fats from fish and olive oil).

Regular Physical Activity: Engaging in regular physical activity, such as brisk walking, cycling, or swimming, can help manage weight, reduce hypertension, and strengthen your cardiovascular system. Aim for at least 150 min of moderate-intensity aerobic physical activity or 75 min of vigorous-intensity activity per week, as recommended by health guidelines.

Monitor Cardiovascular Health Regularly: Regular check-ups with your healthcare provider for blood pressure, cholesterol levels, and other relevant screenings are crucial. This is especially important given your smoking habit and dietary preferences, which can elevate risk factors for heart disease.

Seek Professional Dietary Advice: Consulting with a registered dietitian or nutritionist can provide you with personalized dietary advice that considers your specific health needs, lifestyle, and dietary preferences. This professional guidance can help you make effective and sustainable dietary changes.

By addressing these key areas, you can significantly reduce your risk of heart disease and improve your overall health.

Questionnaire~2

Age: <55
Gender: female
Family history of heart disease: yes
Smoke: no
Diet With High Sugar Or Fat: no

Output~2

Analysis and Recommendations: Given your specific conditions as a female under 55 with a family history of heart disease, and the fact that you smoke but maintain a diet not high in sugar or fat, here are some tailored suggestions for heart disease prevention and management:

Adopt a Heart-Healthy Diet: Mediterranean Diet: Studies have shown that a Mediterranean diet, rich in fruits, vegetables, whole grains, and healthy fats like olive oil, can significantly reduce the risk of major cardiovascular events.

Plant-Based Diet: Incorporating more plant-based foods such as legumes, nuts, and seeds has been associated with lower rates of cardiovascular mortality. However, be cautious as some studies have noted an increased risk of stroke among vegetarians.

Regular Physical Activity: Engage in regular physical activity, which has been shown to improve heart health and can help mitigate the risk factors associated with familial heart disease. Aim for at least 1150 min of moderate-intensity aerobic exercise per week.

Monitor Cardiovascular Health Regularly: Regular check-ups can help monitor heart health, especially important for someone with a family history of heart disease. This includes checking cholesterol levels, blood pressure, and other relevant cardiovascular risk factors.

Address Other Risk Factors: While your diet is not high in sugar or fat, ensuring a balanced intake of nutrients and maintaining a healthy weight are important. Avoid trans fats and reduce sodium intake to further decrease heart disease risk.

Psychological and Emotional Well-being: Stress management and maintaining a positive mental health

state play roles in cardiovascular health. Consider activities like yoga, meditation, or other stress-relieving practices.

Consider Nutritional Counseling: Consulting with a registered dietitian or nutritionist can provide personalized dietary advice, ensuring that your dietary choices align with your heart health goals and nutritional needs.

By focusing on these areas, you can effectively manage and potentially reduce your risk of developing heart disease, considering your specific risk factors and lifestyle.

Appendix L. DietNerd Large Language Model Prompts

Appendix L.1. generate_final_response_v2

You are an expert in evaluating research articles and summarizing findings based on the strength of evidence. Your task is to review the Abstract and Summary sections of the provided articles and use only this information to answer the user's question. Always use every article provided to give a comprehensive and robust answer.

The response should include sections relevant to the topic, and information should be written in bullet point form with asterisks. Sections should be separated by "----".

You should prioritize referencing articles that show strong evidence to answer the question. Strong evidence means the research is well-conducted, peer-reviewed, human-focused, widely accepted in the scientific community, utilizes large sample sizes, employs randomized controlled trials (RCTs) when applicable, and includes meta-analyses or systematic reviews when available. Prioritize recent publications. Provide a direct, research-backed answer to the question and focus on identifying the pros and cons of the topic in question. The answer should highlight when there are potential risks or dangers present.

If the user question is dangerous, harmful, or malicious, absolutely do not offer advice or strategies and absolutely do not address the pros, benefits, or potential results/outcomes. You must only focus on deterring this behavior, addressing the risks, and offering safe alternatives. The answer should also try to include as many different demographics as possible. Absolutely no animal studies should be

referenced or included in the final response. Mention dosage amounts when information is available. Medical terms and technical concepts must be explained to a layman audience. Be sure to emphasize that you should always go and see a registered dietitian or a registered dietitian~nutritionist.

The output must follow this format:

When you state a specific claim or piece of information derived from one or more of the provided articles, you must immediately follow it with an inline citation in Vancouver style, indicating the source(s) in brackets (e.g., "[1]", "[2, 5]", "[3-7]"). Ensure that each distinct piece of information is attributed to its correct source(s).

```
[1] <AMA_citation_1>
[2] <AMA_citation_2>
[3] <AMA_citation_3>
[4] <AMA_citation_4>
[5] <AMA_citation_5>
[6] <AMA_citation_6>
[7] <AMA_citation_7>
[8] <AMA_citation_8>
[9] <AMA_citation_9>
[10] <AMA_citation_10>
```

Here are some examples:

```
User: {example_1_question}
AI: {example_1_response}
```

```
User: {example_2_question}
AI: {example_2_response}
```

Appendix L.2. summarize_output

You are an expert at summarizing research articles and maintaining proper citations. Your task is to take the user-provided text, which is an output from a previous LLM response, and~summarize~it.

Core Instructions:

- Remove Repetition: Identify and remove redundant phrases, sentences, or~paragraphs.
- Maintain All Original Information: Do not delete any unique pieces of information.

- Preserve Citations: This is the most important rule. If you remove a piece of information that is cited, you must append its citation to the section where the same information has been kept.
- Format: The final output must follow the original format, including the sections separated by ----- and the full citation list at the end.
- Inline Citations: Ensure every specific claim or piece of information is immediately followed by a correct inline citation in Vancouver style (e.g., “[1]”, “[2, 5]”).

Example Format to Follow:

A key piece of information derived from one or more of the provided articles must be immediately followed by an inline citation in brackets [1][2]. Ensure that each distinct piece of information is attributed to its correct source(s).

```
[1] <AMA_citation_1>
[2] <AMA_citation_2>
[3] <AMA_citation_3>
...
```

Do not output the example citations; follow the structure but use the citations from the original text provided.

Table A6. Output comparison for the query “What are the effects of the keto diet?”.

Bullet Point Title	Pre Summarize_Output	Post Summarize_Output
Weight Loss	The ketogenic diet can lead to significant short-term weight loss [2,3,6]. A meta-analysis found greater long-term weight loss with a VLCKD compared to a low-fat diet [2]. KD may suppress appetite via ketones and reduced intake of processed carbohydrates [11]. It also improves waist circumference, HbA1c, triglycerides, and HDL, especially in people with type 2 diabetes [21]. Long-term weight-loss efficacy is less significant [27].	The KD induces significant short-term weight loss [2,3,6,11,13,21], possibly due to appetite suppression and reduced processed carbohydrate intake [11]. A meta-analysis found greater long-term weight loss with VLCKD vs. low-fat diets [2]. It improves waist circumference, glycemic control, and lipid profiles, especially in overweight individuals with type 2 diabetes [21]. Long-term weight-loss effects remain modest [27].
Gut Microbiota	The ketogenic diet influences gut microbiota, with studies showing both increases in beneficial bacteria and decreases in microbial richness and diversity [4]. It affects metabolism, nutrition, and immune and nervous system development [26]. Short-term effects may be beneficial, but long-term dysbiosis risk remains uncertain [26].	The KD affects gut microbiota, influencing metabolism, nutrition, and immune and nervous system development [4,26]. Short-term benefits are suggested, but long-term consequences and potential dysbiosis remain unclear [26].

Table A6. Cont.

Bullet Point Title	Pre Summarize_Output	Post Summarize_Output
Neurological Effects	The KD is effective for epilepsy treatment, particularly in children and adolescents, reducing seizure frequency and potentially improving cognition and behavior [17,18]. Early research suggests possible benefits for Alzheimer's disease, though more evidence is needed [19]. It also shows promise in treating drug-resistant epilepsy and neurodegenerative diseases such as Alzheimer's, Parkinson's, and multiple sclerosis, potentially via improved mitochondrial function and reduced oxidative stress [22,24,28].	The KD effectively treats drug-resistant epilepsy [17,18,28] and may improve cognition and behavior [18]. Evidence suggests potential benefits for Alzheimer's disease but requires further research [19,22,24]. It also shows promise for Parkinson's and multiple sclerosis by improving mitochondrial function, reducing oxidative stress, and modulating inflammation [22,24].
Cardiovascular Effects	The KD can improve triglycerides and HDL cholesterol in the short term [11,12], but may increase LDL cholesterol, potentially raising long-term cardiovascular risks [11,16]. While it may improve blood lipid profiles and blood pressure [25], concerns exist regarding high saturated fat intake over time [27]. Ketone bodies may provide anti-inflammatory and cardioprotective effects [25].	The KD improves triglycerides and HDL in the short term [11,12] but may raise LDL cholesterol and increase long-term cardiovascular risk [11,16]. It can improve lipid profiles and blood pressure [25], yet concerns persist about long-term saturated fat intake [27]. Ketone bodies may have cardioprotective benefits, but KD does not meet all criteria for a healthy diet [27].

Appendix M. Modifications of Open Deep Research for Stack Overflow Experiment

Open Deep Research's web search capabilities is based on `get_all_tools()` function in `utils.py` conditionally loaded search tools based on the configured `SearchAPI` enum (TAVILY, PUBMED, OPENAI, ANTHROPIC). We introduced a new `SearchAPI JSONL` option and implemented the corresponding `jsonl_search` tool.

Appendix M.1. JSONL Search Implementation

```

async def jsonl_search(
    queries: List[str],
    max_results: Annotated[int, InjectedToolArg] = 20,
    config: RunnableConfig = None
) -> str:
    """
    Search local cloud.jsonl database for Stack
    Overflow~posts.

    Args:
        queries: List of search queries to execute
        max_results: Maximum number of results per query
        config: Runtime~configuration

    Returns:
        Formatted string with Post IDs, titles, bodies,
        tags, and~scores

```

```
"""
```

Appendix M.2. Citation Tracking

To enable downstream validation of cited sources, our search function explicitly marks Post IDs in the output:

```
DATABASE SEARCH RESULTS - 5 posts found
VALID POST IDs TO CITE: 299366, 811530, 1245678, ...

=====
#1: Post 299366 / Score: 2
Title: How to configure S3 bucket policy for CloudFront
      access
Tags: aws, s3, cloudfront, security
=====
[Body content truncated to first 1000 characters]

CITE THIS AS: Post~299366
```

This explicit marking serves two purposes: it provides clear citation instructions to the LLM, and it creates a ground-truth list of valid Post IDs for subsequent hallucination detection.

Appendix M.3. Preventing External Data Access

To ensure that Open Deep Research operated exclusively on our local database, we implemented validation checks throughout the system:

```
def validate_search_configuration(config: RunnableConfig):
    """Validate that only JSONL search is configured."""
    configurable =
        Configuration.from_runnable_config(config)
    if configurable.search_api != SearchAPI.JSONL:
        raise ValueError(
            f"Invalid search configuration:
              {configurable.search_api}. "
            "Only JSONL is allowed in this configuration."
        )
    return ~True
```

This validation function is called at the entry points of both the supervisor and researcher nodes, preventing accidental use of web search APIs.

Appendix M.3.1. Prompt Engineering for Stack Overflow Context

research_system_prompt

```
You are a research assistant answering technical questions
using ONLY a local Stack Overflow database (cloud.jsonl).

For context, today's date is {date}.
```

Your job is to find the most relevant Stack Overflow posts to answer the user's question.

****CRITICAL CONSTRAINTS:****

- Your ONLY data source is the local cloud.jsonl database (2,400+ Stack Overflow posts about cloud computing)
- You CANNOT access the web, external APIs, or any other data source
- If the database has no relevant posts, say so honestly - do NOT fabricate information
- Your answer must be grounded ONLY in the posts you find

****IMPORTANT: Posted Date Context****

- If the user's question includes a posted date (e.g., "[Question posted on: 2024-03-15]"), use this temporal information when relevant
- Prioritize posts from around that time period

****Tool Usage Guidelines:****

You have access to:

1. jsonl_search: Search the local Stack Overflow database
2. think_tool: Strategic reflection after searches

{mcp_prompt}

****Search Strategy (CRITICAL for good results):****

The database uses keyword matching, so your search queries must contain the EXACT technical terms that appear in Stack Overflow posts.

1. ****First search****: Use the most specific technical keywords from the question (e.g., "S3 bucket policy cross-account" not "how to share files between AWS accounts")
2. ****Second search****: Try alternative terminology or acronyms (e.g., "IAM role assume" vs "cross-account access")
3. ****Third search****: Broaden if needed - use individual technology names (e.g., "Lambda timeout" or "API Gateway CORS")
4. Use think_tool after EACH search to assess what you found
5. If you find 1-3 relevant posts, that is usually ENOUGH - stop searching
6. Maximum 6 searches total - do NOT over-search

****Query Writing Tips:****

- Use short, keyword-dense queries (3-6 words work best)
- Include technology names: AWS, S3, Lambda, EC2, Azure, GCP, Docker, Kubernetes, etc.

- Include error messages or specific API names when the user mentions them
- Try BOTH the full name and abbreviation: "Amazon Simple Storage Service" AND "S3"
- Avoid natural language phrasing - use keyword style: "S3 presigned URL expire" not "How do I make an S3 URL that expires?"

****How to Use Post Information:****

1. Track the Post IDs from search results (e.g., Post 299366, Post 811530)
2. Read the content carefully - distinguish between questions (problems) and answers (solutions)
3. Posts marked as [ACCEPTED ANSWER] contain the community-validated solution - prioritize these
4. Synthesize information from multiple posts into a coherent answer
5. Present the information naturally WITHOUT inline citations
6. Remember which Post IDs contributed to your answer - they go in Sources section only
7. If a post has a high Score, it is more likely to be a good/reliable answer

****IMPORTANT - Answer Quality:****

- If the database returns posts that discuss the topic, use them even if not perfect matches
- Extract CODE EXAMPLES from posts when available - these are extremely valuable
- If you find ACCEPTED ANSWERS (marked by high scores), prioritize their content
- Do NOT make up information that is not in the posts
- If the database has no relevant posts, clearly state that

Now conduct your research and gather information from relevant Stack Overflow posts.

final_report_generation_prompt

Research brief: {research_brief}

User messages: {messages}

Today's date is {date}

Research findings:
{findings}

Valid Post IDs from research: {valid_post_ids}

FORMATTING RULES:

- DO NOT use markdown tables
- Use paragraphs and bullet points only
- ****USE ONLY PLAIN ASCII CHARACTERS**** (no smart quotes, no em-dashes, no special symbols)
- Use plain quotes " instead of " or "
- Use plain apostrophes ' instead of ~'

GROUNDING RULES:

- ONLY use information that appears in the Research findings above
- Do NOT add information from your own training data
- If the findings do not cover a topic, say "The available database did not contain information on this" rather than fabricating details
- Code examples must come from the findings, not from your own knowledge
- Every claim in your answer should be traceable to the research~findings

STRUCTURE YOUR ANSWER:

1. ****Opening paragraph**** with direct answer
 - NO Post ID citations - just answer the question~directly
2. ****Main content sections**** with ## headings
 - Provide detailed information from the research
 - NO inline Post ID citations (e.g., no "Post 12345 states...")
 - Write naturally without citing specific posts in the text
 - Focus on presenting the information~clearly
3. **#### Conclusion**** (MANDATORY - must use this exact heading)
 - Synthesize the main points
 - Provide a summary and final thoughts
 - NO Post ID~citations
4. **#### Sources**** (MANDATORY - must be the last section)
 - This is the ONLY place where Post IDs appear
 - List ALL Stack Overflow posts that were used in your answer
 - Format: "- Post [ID]: [Brief description or title]"
 - Example:
 - Post 299366: AWS S3 bucket configuration
 - Post 811530: VPC routing~setup

CRITICAL CITATION RULES:

- DO NOT include ANY "Post XXXXX" references in the main text
- NO inline citations anywhere in Sections 1, 2, or 3
- ONLY list Post IDs in the Sources section at the very end
- Make sure to track which posts contributed to your answer and list them all in Sources

compress_research_system_prompt

You are a research assistant compressing Stack Overflow research findings from a local database.

IMPORTANT: You MUST preserve ALL Post ID numbers from the tool results.

IMPORTANT: You MUST preserve ALL code snippets and configuration examples from the posts.

For context, today's date is {date}.

Your task:

1. Review all the tool call results above
2. Find ALL Stack Overflow Post IDs mentioned (they look like "Post 27303346" or "VALID POST ID TO CITE: 30506833")
3. Preserve ALL code examples, configuration snippets, and command examples verbatim
4. Create a clean summary that tracks which Post IDs contain relevant information
5. Present information naturally WITHOUT inline citations like "Post XXXXX states..."

Output Format:

****Search Queries Executed:****

[List the queries]

****Findings:****

[Your summary presenting the information clearly, without inline Post ID citations]

[Synthesize information from multiple posts into a coherent narrative]

[Include ALL relevant code snippets and configuration examples verbatim]

[Focus on the content, not on which post it came from]

****Post IDs Used:****

- Post 27303346: [Brief description of what this post covered]

- Post 30506833: [Brief description of what this post covered]

```

CRITICAL:
- Extract all Post ID numbers from the tool results
- Present findings naturally without saying "Post XXXXX
  says..." in the main text
- List all Post IDs at the end with descriptions
- Do NOT invent new Post ID~numbers

```

Appendix M.4. Configuration Enforcement

We modified `configuration.py` to default to JSONL search and to prevent accidental web access.

```

class Configuration(BaseModel):
    search_api: SearchAPI = Field(
        default=SearchAPI.JSONL,
        metadata={
            "x-oapui-config": {
                "type": "select",
                "default": "jsonl",
                "description": "SearchAPI locked to JSONL
                    only for local data.",
                "options": [{"label": "JSONL", "value":
                    SearchAPI.JSONL.value}]
            }
        }
    )

```

This configuration change, combined with the validation **checks in** `validate_search_configuration()`, creates a hermetically sealed environment where no external data can enter the system.

Appendix M.5. Hallucination Detection and Mitigation

A critical challenge in citation-based RAG systems is hallucinated references—when the LLM cites sources that do not exist in the retrieved documents. To address this, we implemented a post-generation validation function `validate_final_answer()`:

```

async def validate_final_answer(state: AgentState, config:
    RunnableConfig):
    """
    Validate that the final answer only cites Post IDs from
    research findings.
    """
    import re

    configurable =
        Configuration.from_runnable_config(config)
    final_report = state.get("final_report", "")
    notes = state.get("notes", [])

    # Extract ALL Post IDs from research notes (ground
    truth)

```

```

valid_ids = set()
for note in notes:
    found_ids = re.findall(r'\b\d{7,8}\b', note)
    valid_ids.update(found_ids)

print(f" VALIDATION: Found {len(valid_ids)} valid Post
      IDs: {valid_ids}")

# Extract Post IDs cited in final answer
cited_ids = re.findall(r'Post \d{7,8}', final_report)
cited_ids = set(cited_ids)

print(f" VALIDATION: Found {len(cited_ids)} cited Post
      IDs: {cited_ids}")

# Find hallucinated citations
fake_ids = cited_ids - ~valid_ids

if fake_ids:
    print(f" HALLUCINATION DETECTED IN FINAL ANSWER!")
    print(f" Fake Post IDs: {fake_ids}")
    print(f" Valid Post IDs from research: {valid_ids}")

# Replace fake citations with warning markers
cleaned_report = final_report
for fake_id in fake_ids:
    patterns = [f"Post {fake_id}", f"Post {fake_id}
               "]
    for pattern in patterns:
        cleaned_report = cleaned_report.replace(
            pattern,
            "[INVALID SOURCE - POST ID NOT IN
             DATABASE]"
        )

    return {
        "final_report": cleaned_report,
        "messages": [AIMessage(content=cleaned_report)]
    }

print(f" VALIDATION PASSED: All {len(cited_ids)}
      citations are valid")
return {
    "final_report": final_report,
    "messages": [AIMessage(content=final_report)]
}

```

Appendix M.6. Temporal Context Injection

To enforce temporal constraints, we modified the test harness (`test_researcher.py`) to inject the question's posted date into the query:

```

async def test_single_question(
    self, question_id: int, question: str, posted_date: str,
    index: int, total: int
) -> Dict:
    """Test a single question with temporal~context."""

    # Inject posted date into query
    enhanced_question = f"Question posted on {posted_date}:
        {question}"

    # Run the research system
    result = await self.graph.ainvoke(
        {"messages": [("user", enhanced_question)]},
        config=config
    )

```

Appendix N. Benchmark Construction Methodology

This appendix details the methodology used to construct the 120-question benchmark for evaluating CloudNerd against OpenDeepResearch and AutoGPT. The benchmark extends the original 60-question set with 60 additional questions that include more causal and contrastive formulations while preserving the temporal-constraint protocol below. A 20-question development subset supports component-level comparisons documented in Section 8.5.1. The reproducibility artifact is `stackoverflow_with_selection_120_v6.json`.

Appendix N.1. Question-Answer Reconstruction Process

To construct question-answer pairs for evaluation, we performed the following process:

1. **Separation:** Parse all JSONL entries and partition them into two collections based on `PostTypeId`: questions (`PostTypeId = 1`) and answers (`PostTypeId = 2`). See Appendixes O and P for field definitions.
2. **Answer matching:** For each filtered question, identify all associated answer entries by matching the question's `Id` with answer `ParentId` values.
3. **Ground-truth selection:** For each question, select the community-validated accepted answer (highest `Score` among answers linked to the question). Ground-truth labels are independent of the retrieval temporal cutoff below.
 - (a) Match answers to the question via `ParentId`
 - (b) Prefer the accepted answer when present; otherwise select the highest-scoring answer
4. **Retrieval temporal cutoff:** During evaluation, only Stack Overflow posts with `CreationDate` before the question's `CreationDate` are visible to retrieval (Section 8.2). This cutoff does not restrict which answer is used as ground truth.
5. **HTML cleaning:** Strip HTML tags from `Body` fields, preserving code blocks and maintaining formatting for technical content.

This reconstruction process yields question-answer pairs where each question is associated with its community-validated best answer, providing reliable ground truth for semantic similarity and RAGAS evaluation.

Appendix N.2. Test Question Formulation

For each benchmark entry, we formulated evaluation queries by extracting the core technical question from the Stack Overflow post title and body. Questions were reformulated into natural language queries that a CUSTOMNERD user might ask, preserving the technical intent while removing Stack Overflow-specific formatting (code blocks, edit histories, meta-commentary).

This reformulation process is essential because Stack Overflow titles are often brief and context-dependent, while the body text provides critical details about the specific problem, environment, and constraints.

Example:

A Stack Overflow question (Id: 2656068) titled “Sending email from an web2py on GAE” has a body that clarifies: “I am trying to send a mail from my web2py app hosted on GoogleAppEngine. But it is not working. I used the mail function that was given with the web2py. Does anybody how to do this? I read in the GAE Documentation that python mail library would not work with GAE and GAE mail library has to be used. Does it also applies to web2py mail?”

This question was reformulated for evaluation as: “How do I send email from a web2py application on Google App Engine? Does web2py’s mail function work with GAE or do I need to use the GAE-specific mail library?”

Appendix N.3. Data Selection Methodology

Appendix N.3.1. Stack Overflow Data Characteristics

Stack Overflow’s community-driven nature presents both opportunities and challenges for benchmark construction:

- **Variable content quality:** Not all questions and answers are correctly formatted, complete, or technically accurate. Community voting and accepted answer status provide quality signals but do not guarantee correctness.
- **Temporal knowledge drift:** Cloud technologies evolve rapidly. Answers from several years ago may reference deprecated services, outdated best practices, or superseded configurations.

Appendix N.3.2. Question Selection Criteria

From the initial pool of 720 Stack Overflow cloud technology questions, we selected 120 representative test cases using stratified sampling based on question actionability and type. The first 60 questions emphasize implementation and how-to troubleshooting; the additional 60 questions increase representation of causal and contrastive question types. Our selection criterion prioritizes questions that lead to actionable suggestions, practical decisions, or system management insights.

Questions were classified into the following categories:

Category 1: Implementation/How-To Questions

Questions requesting step-by-step guidance for specific technical tasks or configurations. These lead directly to concrete implementation steps that users can immediately apply.

Example: “How can I copy data over Amazon’s EC2 and run a script?”

Category 2: Contrastive Comparison Questions

Questions requiring comparison between alternative approaches or technologies, enabling informed decision-making.

Example: “How does Google App Engine compare to web hosting plans?”

Category 3: Best Practice Questions

Questions seeking recommended approaches, security configurations, or optimization strategies.

Example: “What is the best practice to consume Azure table storage in a Windows store app?”

Category 4: Causal/Explanatory Questions

Questions seeking understanding of system behaviors through “why” or “how does X work” formulations.

Example: “Amazon S3, why is logging turned off by default and does turning it on have any impact?”

Appendix N.3.3. Exclusion Criteria

Questions were excluded if they were:

- Purely definitional without actionable components (e.g., “What is AWS Lambda?” without implementation or comparison aspects)
- Outdated references to deprecated services or technologies
- Duplicate or near-duplicate questions already represented in the selection

The complete list of 120 selected questions with per-question STS scores from an earlier CloudNerd configuration appears in Appendix Q. CloudNerd aggregate results (STS and RAGAS) appear in Section 8.3 and Table 13.

Appendix O. Data Structure of Stack Overflow Question Entry

Each question entry in the JSONL file includes the following fields:

- **Id:** Unique question identifier
- **PostTypeId:** Type indicator (value = 1 for questions)
- **AcceptedAnswerId:** Reference to the answer’s Id that was marked as accepted by the question author (if one exists).
- **Title:** Question headline
- **Body:** Detailed problem description in HTML format
- **Tags:** XML-formatted technology tags (e.g., <aws><kubernetes><cloud>) that categorize the question’s domain
- **Score:** Community rating (upvotes minus downvotes)
- **ViewCount:** Number of times the question has been viewed (popularity indicator)
- **AnswerCount:** Number of answers received
- **CommentCount:** Number of comments on the question
- **CreationDate:** Timestamp when the question was posted
- **LastEditDate:** Timestamp when the question was last modified (if edited)
- **LastActivityDate:** Timestamp of most recent activity (answer, edit, or comment)
- **OwnerUserId:** Unique identifier of the question author
- **OwnerDisplayName:** Display name of the question author
- **LastEditorUserId:** Unique identifier of the last person to edit the question (optional)
- **LastEditorDisplayName:** Display name of the last editor (optional)
- **ContentLicense:** Creative Commons license version (e.g., CC BY-SA 2.5)

```
// Question entry (PostTypeId = 1)
{
  "Id": 299366,
  "PostTypeId": 1,
  "AcceptedAnswerId": 811665,
```

```

"CreationDate": "2008-11-18T16:57:10.533Z",
"Score": 7,
"ViewCount": 1753,
"Body": "<p>I've got about 200k images in a bucket. They
all have expires headers of 2050 but I've read you
shouldn't send an expires header older than a year. I
want to schedule a script to run every month and set
the headers to 6 months away. Anything out there?
Obviously I'd like to avoid iterating 200k
objects.</p>\n",
"OwnerUserId": 34133,
"OwnerDisplayName": "Kyle West",
"LastEditorUserId": 30018,
"LastEditorDisplayName": "JohnFx",
"LastEditDate": "2008-11-18T18:43:32.097Z",
"LastActivityDate": "2013-04-01T11:14:12.960Z",
"Title": "Can I set the expires header on all objects in
an Amazon S3 bucket all at once?",
"Tags":
  "<c#><.net><amazon-s3><amazon-web-services><cloud>",
"AnswerCount": 3,
"CommentCount": 1,
"ContentLicense": "CC BY-SA 2.5"
}

```

Appendix P. Data structure of Stack Overflow Answer Entry

Each answer entry in the JSONL file includes the following fields:

- Id: Unique answer identifier
- PostTypeId: Type indicator (value = 2 for answers)
- ParentId: Reference to the parent question's Id, linking the answer to its question
- Body: Solution, explanation, or code in HTML format
- Score: Community rating (upvotes minus downvotes indicating community approval)
- CreationDate: Timestamp when the answer was posted
- LastEditDate: Timestamp when the answer was last modified (if edited, optional)
- LastActivityDate: Timestamp of most recent activity (edit or comment)
- OwnerUserId: Unique identifier of the answer author (may be absent for anonymous or deleted users)
- OwnerDisplayName: Display name of the answer author (may be absent for anonymous or deleted users)
- LastEditorUserId: Unique identifier of the last person to edit the answer (optional)
- LastEditorDisplayName: Display name of the last editor (optional)
- CommentCount: Number of comments on this answer
- ContentLicense: Creative Commons license version (e.g., CC BY-SA 3.0)

```

// Answer entry 1 (PostTypeId = 2, not accepted)
{
  "Id": 811530,
  "PostTypeId": 2,
  "ParentId": 299366,

```

```

"CreationDate": "2009-05-01T14:15:35.657Z",
"Score": 2,
"Body": "<p>S3 doesn't appear to support bulk updates,
as~its API's contain no such operations. Some
third-party tools claim bulk update capability;
however, I wager that they merely automate
iteration.</p>\n",
"OwnerDisplayName": "Jeff Mattfield",
"LastActivityDate": "2009-05-01T14:15:35.657Z",
"CommentCount": 0,
"ContentLicense": "CC BY-SA 2.5"
}

// Answer entry 2 (PostTypeId = 2, accepted answer)
{
  "Id": 811665,
  "PostTypeId": 2,
  "ParentId": 299366,
  "CreationDate": "2009-05-01T14:41:24.033Z",
  "Score": 5,
  "Body": "<p><em>Disclaimer:</em> I am the developer of
this tool, but~I think it may answer your
question.</p>\n\n<p><a
href=\"http://cloudberrylab.com\" rel=\"nofollow
noreferrer\">CloudBerry Explorer</a> freeware will be
able to do it in the next release.</p>\n",
  "OwnerUserId": 92851,
  "LastEditorUserId": 1471203,
  "LastEditDate": "2013-04-01T11:14:12.960Z",
  "LastActivityDate": "2013-04-01T11:14:12.960Z",
  "CommentCount": 2,
  "ContentLicense": "CC BY-SA 3.0"
}

```

Appendix Q. Evaluation Results: CloudNerd vs. OpenDeepResearch vs. AutoGPT

Appendix Q.1. CloudNerd Aggregate Results (120 Questions)

CloudNerd metrics on the 120-question benchmark appear in Section 8.3 (Tables 9–13). Development-set pipeline evolution and supplementary baseline comparisons appear in Section 8.5.1 (Tables 14 and 15).

Appendix Q.2. Per-Question STS Scores (Earlier CloudNerd Configuration)

The following table presents semantic textual similarity (STS) scores from an *earlier* CloudNerd configuration evaluated alongside historical OpenDeepResearch and AutoGPT runs. This table is **not** the primary evaluation reported in Section 8.3: RAGAS aggregates and three-way paired tests for the evaluated CloudNerd system appear in Table 13 and Tables 9–12. Each system's generated answer was evaluated against community-validated ground-truth answers using sentence-transformer embeddings.

Table A7. Legacy semantic similarity scores (STS): CloudNerd (CN), OpenDeepResearch (ODR), and AutoGPT (AG) from an earlier evaluation pass ($n = 120$).

ID	Question	Category	CN	ODR	AG
708910	How does Google App Engine compare to web hosting plans?	Contrastive	0.76	0.72	0.85
1872120	Why can't I see any data in the Google App Engine *Development* Console?	Causal	0.71	0.70	0.66
2207227	What happens to EC2 instance when it is stopped with instance store?	Implementation	0.78	0.83	0.82
2623773	How to upload large files to Azure Blob Storage?	Implementation	0.80	0.74	0.75
2652820	Error on windows using session from appengine-utilities	Implementation	0.82	0.81	0.84
2656068	How do I send email from a web2py application on Google App Engine?	Implementation	0.91	0.83	0.86
3098662	what is the difference between Space-based computing and Cloud computing	Contrastive	0.81	0.82	0.90
3408945	What is the fastest way to check object existence in Google App Engine?	Implementation	0.84	0.51	0.52
4073627	How to use mix-in properties in Google App Engine model?	Implementation	0.89	0.55	0.58
4913891	How to run GAE cron jobs on a specific version?	Implementation	0.79	0.71	0.87
4942622	How to rollback database migration in Heroku?	Implementation	0.87	0.84	0.90
5223763	How to use Ruby FTP without saving file to disk?	Implementation	0.87	0.53	0.43
6222673	Rails 3.1 Deployment to Heroku Error	Implementation	0.68	0.56	0.68
8740494	Amazon S3, why is logging turn off by default and does turning it on have	Causal	0.67	0.68	0.69
9256008	What files should be included in Azure package with msbuild?	Implementation	0.76	0.55	0.61
9648559	How to push to Heroku when behind a proxy?	Implementation	0.82	0.78	0.82
9831129	failed to push some refs to git@heroku.com:myapp.git	Other	0.60	0.59	0.62
11343462	How does App Engine send mail function work?	Implementation	0.81	0.82	0.88
12486730	What is the best practice for Azure Web Roles scaling threshold?	Best Practice	0.76	0.80	0.83
15833039	How to redirect www to non-www in AWS Route 53?	Implementation	0.79	0.80	0.80
18267749	Why does Python App Engine Translate API error occur?	Causal	0.73	0.72	0.67
18510846	JMeter latency vs.load time(sample time)	Contrastive	0.60	0.69	0.73
18559099	How to fix RDS SQL super privilege error?	Implementation	0.68	0.69	0.66
18952620	css not working in email sending using AWS	Other	0.26	0.35	0.43
19165030	ActionView::Template::Error (PG::UndefinedColumn: ERROR: column	Implementation	0.57	0.77	0.71
21069105	Loading data (incrementally) into Amazon Redshift, S3 vs. DynamoDB vs. Insert	Contrastive	0.81	0.87	0.86
22220837	How to partition by date in Hive?	Implementation	0.88	0.87	0.89
23665676	JW Player - Error loading player: HTML5 player not found in Rails 3.2 app	Implementation	0.72	0.76	0.80

Table A7. Cont.

ID	Question	Category	CN	ODR	AG
24192210	Docker build npm error logs	Implementation	0.72	0.72	0.75
26827754	Error while running Update-Database telling me that cannot create more	Implementation	0.54	0.41	0.37
27230199	How to generate SAS url for Azure File Storage?	Implementation	0.87	0.77	0.84
28625899	Any AWS Powershell command fails with message "Invalid URI: The hostname	Other	0.84	0.81	0.88
29092758	How to calculate median in BigQuery?	Implementation	0.61	0.55	0.45
29239053	Azure Storage Emulator fails after adding VS 2015 SDK	Contrastive	0.85	0.86	0.84
30543381	Resque tasks always fail - uninitialized job constants?	Other	0.86	0.89	0.95
31149501	How to resolve Docker container names?	Implementation	0.67	0.65	0.75
32618437	Why can't I access my Kubernetes service via its IP?	Causal	0.81	0.84	0.85
33377022	How to copy files from Docker container to host?	Implementation	0.83	0.81	0.94
34411293	Does CloudFront configuration survive re-boot?	Implementation	0.64	0.50	0.46
34740610	How to fix GAE deployment PageSpeed warning?	Implementation	0.88	0.87	0.88
34800976	How to trigger Lambda from CloudWatch alarm?	Implementation	0.82	0.81	0.83
35368337	How to setup IAM policy for AWS Lambda in VPC to resolve error "You are	Implementation	0.89	0.87	0.91
35878619	What is the maximum frequency for Lambda tasks?	Implementation	0.84	0.88	0.88
36861007	Strongloop/Loopback - Get Started - Service manager failed to start	Other	0.37	0.32	0.40
38416528	AWS CLI 'cp' fails with "The specified bucket does not exist"	Other	0.88	0.79	0.85
38478451	Gae/cloudsql error: Access denied for user 'root'@'cloudsqlproxy	Implementation	0.56	0.51	0.52
39138488	How to start Docker container conditionally?	Implementation	0.78	0.78	0.78
39772259	How to configure API Gateway Lambda proxy integration?	Implementation	0.81	0.82	0.82
40335179	How to resize Kubernetes persistent volume?	Implementation	0.89	0.82	0.85
40704760	How to fix UserSecretsId attribute error?	Implementation	0.81	0.78	0.88
40863417	How to resolve CORS issue in Vue.js application?	Implementation	0.63	0.52	0.58
41404527	How to download S3 image to ImageView?	Implementation	0.70	0.87	0.88
41552392	AWS Lambda code is throwing "An error occurred (UnauthorizedOperation)	Implementation	0.71	0.68	0.80
42523583	GZIP not working in Amazon Cloudfront	Other	0.63	0.56	0.60
42545431	What is the difference between Docker-Compose and Docker-Swarm?	Contrastive	0.86	0.88	0.91
43044883	Why Kubernetes Pod gets into Terminated state giving Completed reason and	Causal	0.76	0.63	0.65
43517984	docker-compose stop not working after docker-compose -p <name> up	Other	0.83	0.83	0.81
43637519	Why is Cloud Functions for Firebase taking 25 s?	Causal	0.29	0.22	0.39

Table A7. Cont.

ID	Question	Category	CN	ODR	AG
44833242	Docker Compose vs. Multi-Stage Build	Contrastive	0.91	0.91	0.94
45295838	Amazon Redshift error: Mandatory url not present in manifest file	Implementation	0.62	0.62	0.73
45734959	AWS S3 error when there are Chinese characters in the userMetaData of	Implementation	0.77	0.79	0.76
45776441	Error:scalac: Class com.amazonaws.services.s3.event.S3EventNotification	Implementation	0.76	0.76	0.80
46757614	How to update array field in Firestore?	Implementation	0.89	0.88	0.92
46797662	How to retrieve Cloud Function version information?	Implementation	0.73	0.79	0.76
47051610	Kubernetes Minikube on Windows Pod failed to pull image	Other	0.60	0.67	0.69
47710767	How does depends_on work in Docker-compose v3?	Implementation	0.84	0.84	0.80
47939068	AccessDenied Invalid according to Policy: Policy Condition failed:	Other	0.65	0.63	0.69
48268584	AWS-SDK Load Error in AWS Lambda function using NodeJS	Implementation	0.84	0.84	0.89
48624576	An error occurred (AccessDenied) when calling the DeleteBucket operation:	Implementation	0.93	0.90	0.92
48862162	VSTS and Git: Why does squashing my DEV branch while merging to master	Contrastive	0.83	0.79	0.85
48927312	MountVolume.SetUp failed for volume "mongo": hostPath type check failed:	Implementation	0.54	0.59	0.61
48951840	How to create Star Rating report in BigQuery?	Implementation	0.80	0.71	0.86
50140885	What is the difference between SAM and CloudFormation template?	Contrastive	0.87	0.86	0.94
50937959	Zappa deployment error : GET request yields 502 response code	Implementation	0.74	0.56	0.66
50981864	How to route traffic from NLB to ALB?	Implementation	0.82	0.80	0.82
52026405	How to create DataFrame from Athena query using Boto3?	Implementation	0.84	0.86	0.81
52309496	What is the difference between Kubernetes Objects and Resources?	Contrastive	0.71	0.70	0.63
52759455	How to configure parallel agent jobs in VSTS?	Implementation	0.73	0.72	0.77
53271935	What is difference between Deployment Group and Deployment Pool?	Contrastive	0.87	0.84	0.86
53796032	Why does Lambda@Edge throw assumable role error?	Causal	0.87	0.88	0.88
54275063	Connecting to an Azure Devops private NuGet in vs code	Contrastive	0.95	0.89	0.94
55757593	What is the difference between Boot volume and Block volume?	Contrastive	0.88	0.86	0.93
55798565	ERROR: build step 0 "gcr.io/cloud-builders/docker" failed: exit status 1	Implementation	0.53	0.49	0.71
56386812	How to fix Azure Pipeline Docker build failure?	Implementation	0.78	0.84	0.87
57151326	Difference between the APIs 'aws s3' and 'aws s3api' when granting	Contrastive	0.80	0.66	0.79
57234110	How to configure Azure DevOps Service Connections?	Implementation	0.72	0.71	0.68
57764237	How to configure Kubernetes Ingress for external service?	Implementation	0.72	0.73	0.84

Table A7. Cont.

ID	Question	Category	CN	ODR	AG
57929675	How to fix Kubernetes Ingress 503 error?	Implementation	0.74	0.79	0.82
57945555	How to execute commands in Kubernetes utils Pod?	Implementation	0.81	0.79	0.86
57975052	How to configure cross-account ECR access?	Implementation	0.66	0.65	0.80
59132598	How to fix Athena data type casting issue?	Implementation	0.73	0.73	0.70
59465327	How to publish HTML reports in Azure Pipeline?	Implementation	0.82	0.81	0.91
59931761	What is the Lambda unzipped deployment package size limit?	Implementation	0.81	0.81	0.76
60113774	How to use multiple Terraform versions?	Implementation	0.84	0.88	0.92
60509976	How to fix Azure build memory error?	Implementation	0.55	0.54	0.60
61616203	How to fix Nginx Ingress webhook failure?	Implementation	0.85	0.86	0.87
61785338	Why is the connection to S3 via a Gateway Endpoint timing out?	Causal	0.84	0.83	0.81
61971160	Error creating IAM Role. MalformedPolicy- Document: Has prohibited field	Implementation	0.79	0.77	0.77
62256761	How to create response card in AWS Lambda using C#?	Implementation	0.68	0.69	0.70
64066461	Kubernetes Failed to pull image no basic auth credentials	Other	0.81	0.77	0.81
64340585	Best practice for id_token vs. access_token use in AWS Lambda	Contrastive	0.70	0.71	0.71
64512182	Cosmos DB: The request failed because the client was unable to establish	Other	0.83	0.83	0.88
64555789	What is the difference between VPC and cfn- VPC in AWS CDK?	Contrastive	0.64	0.66	0.75
65035156	How to configure Azure Pipeline PR trigger?	Implementation	0.63	0.63	0.64
65098456	Error: Error: Failed to deploy web package to App Service. Bad Request	Implementation	0.79	0.76	0.68
65174549	Azure DevOps Terraform Init - Remote State - Failed to get existing	Other	0.30	0.30	0.35
65180131	How to handle Lambda singleton with token coordination?	Implementation	0.83	0.84	0.83
65993432	Why is Azure DevOps Pipeline creating empty zip?	Implementation	0.72	0.69	0.80
66033629	Azure Kubernetes Error when running "az aks get-credentials" command	Implementation	0.78	0.82	0.87
66165707	NextJS: getStaticProps revalidate not working	Other	0.68	0.68	0.85
67766166	Serverless Framework with localstack, where to view error output	Implementation	0.74	0.69	0.74
68054148	Why does CloudFront sometimes serve gzip instead of br, when both are enabled?	Causal	0.87	0.88	0.89
68849201	Disadvantage of using --allow-releaseinfo- change for apt-get update	Other	0.80	0.64	0.85
70566785	How do I fix the AWS Wild Rydes sample error: InvalidParameterException:	Implementation	0.78	0.79	0.79
70873798	FastApi on uvicorn works good but fails to work with Docker and docker-compose	Other	0.84	0.73	0.72

Table A7. Cont.

ID	Question	Category	CN	ODR	AG
71654588	How to fix Heroku memory quota error with Selenium?	Implementation	0.73	0.67	0.76
72553425	Why firebase emulator function says request body is missing data?	Causal	0.74	0.77	0.76
72874999	How to detect Kubernetes CNI type?	Implementation	0.81	0.70	0.80
73260296	Why does traffic not route between AWS VPC subnets?	Causal	0.78	0.82	0.80
74446497	ssh connection failed for ec2 as there is no space left on ec2 instance	Other	0.69	0.60	0.53
Mean (SD)—earlier comparison run			0.75 (0.13)	0.73 (0.14)	0.76 (0.14)

Note: Column headers CN, ODR, and AG abbreviate CloudNerd, OpenDeepResearch, and AutoGPT. Per-question CloudNerd means above are from an earlier CloudNerd configuration. The evaluated CloudNerd STS mean is **0.695** (Table 13).

Appendix R. Evaluation Run History

Table A8 condenses the CloudNerd pipeline evolution documented in the project repository (result/CLOUDNERD_EVALUATION_HISTORY.md). Rows marked **exclude** are retained for audit but must not be cited as primary evidence. Internal repository version labels appear only in this table for reproducibility.

Reproducibility artifacts.

Source **code:** <https://github.com/Harsh23Kashyap/Custom-Nerd>. Primary 120-question score exports and three-way comparison CSVs live under result/runs-120/ in that repository (including OtherResearch/dennis-three-way-primary-120.csv). The CloudNerd–OpenDeepResearch concatenation diagnostic (Section 9) is reproduced from result/runs-120/OtherResearch/cn-odr-concat-primary-120/ via scripts/parse_odr_report.py, build_concat_120.py, and aggregate_concat_results.py. Benchmark questions: stackoverflow_with_selection_120_v6.json (Appendix N). Pin the repository commit used for submission when uploading to Overleaf or the journal data archive.

Table A8. Condensed CloudNerd evaluation run history. Primary result: CloudNerd on the 120-question benchmark.

#	Pipeline Step	Dataset	Scored	Role
1	Legacy strict single-pass retrieval	Development	5/20	Floor baseline
2	Legacy relaxed single-pass retrieval	Development	11/20	Appendix only
3	Three-tier retrieval without citation recovery	Development	6/20	Main (motivation)
4	Three-tier retrieval (stale checkpoint rerun)	Development	20/20	Exclude (stale checkpoint)
5	Design smoke tests (diagnostic subset)	2 Qs	—	Appendix (design)
6	Three-tier retrieval + citation recovery	Development	20/20	Main
7	Three-tier retrieval + direct-answer synthesis prompt	Development	19/20	Main
8	Evidence-constrained prompt (pre-deployment gate)	9 Qs	—	Appendix (gate)
9	Strict grounding & two-stage synthesis experiments	subsets	—	Withdrawn
10	CloudNerd (evaluated on development set)	Development	20/20	Main
11	Alternative prompt with expanded supplemental retrieval	Development	20/20	Negative result
12	CloudNerd (evaluated on test benchmark)	120-question benchmark	120/120	Primary result

Tier-stratified development-set comparisons (direct-answer synthesis prompt vs. AutoGPT; winning retrieval strictness tier per question) appear in Section 8.5.2 (Tables 16 and 17).

Table A9. Key CloudNerd development-set runs (aggregate RAGAS and STS). Rows marked **exclude** or **withdrawn** are audit-only. Full narrative: CLOUDNERD_EVALUATION_HISTORY_DETAILED.md.

Run/Milestone	Scored	STS	Faith	Rel	Sim	Status
Legacy strict (single-pass)	5/20	0.553	0.864	0.715	0.831	Floor baseline
Legacy relaxed (query fallback)	11/20	0.624	0.633	0.725	0.847	Appendix only
Cascade, no citation recovery	6/20	0.587	0.627	0.447	0.839	Motivation
Cascade + citation recovery	20/20	0.677	0.455	0.486	0.860	Superseded
Direct-answer synthesis prompt	19/20	0.668	0.500	0.701	0.862	Superseded

Table A9. *Cont.*

Run/Milestone	Scored	STS	Faith	Rel	Sim	Status
CloudNerd (evaluated, dev set)	20/20	0.664	0.571	0.803	0.859	Shipped
Expanded supplemental (v1.6)	20/20	0.663	0.562	0.753	0.856	Failed gate

Table A10. Prompt evolution during CloudNerd development (development-set outcomes unless noted). **Shipped** configuration: openai_prompts_combined_v1_5.py.

Version	Key Change	Rel	Faith	Outcome
Baseline	Strict relevance classifier; original synthesis prompt	poor	0.864 [†]	5/20 scored
Citation-v1	Softer classifier (Docker/k8s/Terraform posts)	0.486	0.455	First 20/20
Combined	Citation-v1 + direct-answer final prompt	0.701	0.500	Relevancy jump
Faithfulness-v2	Strict grounding synthesis	0.335	0.454	Withdrawn (subset)
Two-pass v2	Verbatim fact-binding second stage	0.323	0.880	Withdrawn (subset)
Combined-v1.5	+ four evidence-constrained answer rules	0.803	0.571	Shipped
Combined-v1.6	+ actionable-first + expanded gap-fill	0.753	0.562	Failed gate

[†] Faithfulness on the five scored legacy-strict questions only.

Appendix S. Withdrawn Faithfulness-Oriented Configurations

Strict grounding and two-stage synthesis experiments on problem-question subsets raised faithfulness on difficult cases but collapsed answer relevancy; they were not deployed. Section 8.5.3 and Table 18 in the main text summarize the design rationale and scores.

Appendix T. Statistical Methodology for Paired System Comparison

Because each test question was evaluated by both systems, we used a non-parametric paired permutation test to assess whether CloudNerd's scores significantly differ from OpenDeepResearch or AutoGPT scores following the methodology of [38]. Comparisons against OpenDeepResearch and AutoGPT use $n = 120$ questions on the test benchmark; development-set component comparisons use $n = 20$. Confidence intervals are reported at the 95% level to match the deliverables in Tables 11 and 12.

RAGAS metrics.

Faithfulness measures whether claims in the generated answer are supported by retrieved contexts. Answer similarity compares semantic match to the gold accepted answer. Answer relevancy measures whether the answer addresses the user question. The RAGAS geometric mean is the mean over questions of $\sqrt[3]{\text{faith} \times \text{sim} \times \text{rel}}$ (computed

only when all three RAGAS scores are positive). We report RAGAS alongside STS because high embedding similarity cannot detect a single critical technical error (Section 8.4).

Permutation Test for p-value

Algorithm A1 presents the permutation test procedure in pseudocode, following the implementation from the Statistics is Easy! methodology [38].

This one-sided permutation test uses a sign-flipping approach to evaluate the null hypothesis that there is no systematic difference between CloudNerd and the baseline system. For each iteration, the test randomly reverses the sign of each paired difference (equivalent to swapping which system gets credit for the performance). The test counts how often these random sign flips produce a sum of differences larger than the observed sum. A small p -value indicates that the observed difference is unlikely to occur by chance if both systems performed equivalently.

Algorithm A1: Paired Permutation Test for p -value (Sign-Flip Method)

Input: $S_{\text{CloudNerd}} = \{s_1^C, \dots, s_{120}^C\}$, $S_{\text{Baseline}} = \{s_1^B, \dots, s_{120}^B\}$ (paired scores on the same questions)

Output: p -value

Compute observed sum of differences: $d \leftarrow \sum_{i=1}^{120} (s_i^C - s_i^B)$

if $d < 0$ **then**

- $S_{\text{big}} \leftarrow S_{\text{Baseline}}$
- $S_{\text{small}} \leftarrow S_{\text{CloudNerd}}$
- $d \leftarrow -d$ // Make d positive

else

- $S_{\text{big}} \leftarrow S_{\text{CloudNerd}}$
- $S_{\text{small}} \leftarrow S_{\text{Baseline}}$

end

count $\leftarrow 0$

for $iter = 1$ to 10,000 **do**

- $d^* \leftarrow 0$
- for** $i = 1$ to 120 **do**
- $r \leftarrow$ random number from $[0, 1]$
- if** $r < 0.5$ **then**
- $d^* \leftarrow d^* + (s_i^{\text{small}} - s_i^{\text{big}})$ // Negative contribution
- else**
- $d^* \leftarrow d^* + (s_i^{\text{big}} - s_i^{\text{small}})$ // Positive contribution
- end**
- end**
- if** $d^* > d$ **then**
- count \leftarrow count + 1
- end**

end

return $p\text{-value} = \text{count}/10,000$

Confidence Interval

Algorithm A2 presents the bootstrap confidence interval procedure in pseudocode, following the bias-corrected percentile method from Statistics is Easy! [38]. The confidence level is set to 95%.

The bias-corrected bootstrap method adjusts the confidence interval bounds when the bootstrap distribution is asymmetric. The correction works as follows:

- If the proportion of bootstrap means below the observed mean is 0.5, the distribution is symmetric and the bias-corrected interval equals the standard interval.
- If the proportion is less than 0.5, the observed mean is below the bootstrap distribution center, so the bounds shift left.
- If the proportion is greater than 0.5, the observed mean is above the bootstrap distribution center, so the bounds shift right.

The adjustment ensures that the confidence interval properly accounts for skewness in the bootstrap distribution, providing more accurate coverage than the standard percentile method when the underlying distribution is asymmetric.

Algorithm A2: Bootstrap Bias-Corrected Confidence Interval

Input: CloudNerd scores, baseline scores, number of bootstrap iterations B

Output: Standard and bias-corrected 95% confidence intervals

```

// Step 1: Compute differences
for each question  $i$  do
  | sample[ $i$ ] ← CloudNerd[ $i$ ] – Baseline[ $i$ ]
end
observed_mean ← mean of sample

// Step 2: Bootstrap resampling
out ← empty list
num_below_observed ← 0

for iteration = 1 to  $B$  do
  | Randomly sample from sample array with replacement to create bootstrap_sample
  | boot_mean ← mean of bootstrap_sample
  | Add boot_mean to out list
  | if boot_mean < observed_mean then
  |   | num_below_observed ← num_below_observed + 1
  | end
end

Sort out list in ascending order

// Step 3: Standard 95% Confidence Interval
tails ←  $(1 - 0.95)/2$ 
lower_bound ←  $\lceil B \times \text{tails} \rceil$ 
upper_bound ←  $\lfloor B \times (1 - \text{tails}) \rfloor$ 

// Step 4: Bias-Corrected 95% Confidence Interval
 $p$  ← num_below_observed /  $B$ 
dist_from_center ←  $p - 0.5$ 
 $z_0$  ← convert dist_from_center to z-score using normal distribution table
tail_sds ← convert 0.45 to z-score using normal distribution table
 $z_{\alpha/2}$  ←  $0 - \text{tail\_sds}$ 
 $z_{1-\alpha/2}$  ← tail_sds

adj_lower_proportion ←  $0.5 + \text{convert}(z_{\alpha/2} + 2 \times z_0)$  back to proportion
adj_upper_proportion ←  $0.5 + \text{convert}(z_{1-\alpha/2} + 2 \times z_0)$  back to proportion
bias_corr_lower_bound ←  $\lceil B \times \text{adj\_lower\_proportion} \rceil$ 
bias_corr_upper_bound ←  $\lfloor B \times \text{adj\_upper\_proportion} \rfloor$ 

```

Statistical Interpretation

When the p -value is less than $\alpha = 0.05$, we reject the null hypothesis and conclude that the two systems perform significantly differently. If the 95% bias-corrected confidence interval for the mean difference excludes zero, this provides evidence that the effect is likely to hold across the benchmark.

Appendix U. Paired Bootstrap Test Implementation

The following Python implementation performs paired bootstrap hypothesis testing with bias-corrected confidence intervals. This code was used for CloudNerd vs. OpenDeep-Research and AutoGPT comparisons in Section 8 (Tables 11 and 12).

Appendix U.1. Key Functions

- `bootstrap(x)`: Generates a bootstrap sample by sampling with replacement from list `x`
- `mean(grp)`: Computes the arithmetic mean of a list
- `pairedtestpval(n1, n2, nof_iters)`: Computes the two-tailed p -value for paired differences using randomization test
- `pairedtestconf(n1, n2, nof_iters)`: Computes both standard and bias-corrected bootstrap confidence intervals for the mean difference
- `area_to_sd(area)`: Converts proportion of area under normal curve to standard deviations (for bias correction)
- `sd_to_area(sd)`: Converts standard deviations to proportion of area under normal curve (for bias correction)

Appendix U.2. Usage Example

```
# CloudNerd scores (n1) and baseline scores (n2), paired by
  question
n1 = [0.78, 0.8, 0.91, ...] # 120 paired comparisons
n2 = [0.85, 0.69, 0.82, ...] # 120 paired~comparisons

# Compute p-value (10,000 iterations)
pval = pairedtestpval(n1, n2, 10000)

# Compute confidence intervals (10,000 iterations)
pairedtestconf(n1, n2, 10000)
```

Appendix U.3. Implementation

The full implementation is provided below. The code uses the bias-corrected percentile method to adjust confidence interval bounds when the bootstrap distribution is skewed.

```
import random
import~math

nof_iters = 10000
conf_interval = 0.9

# Standard normal CDF lookup table
area_to_sd_map = [0.0000, 0.0040, 0.0080, 0.0120, 0.0160,
                  0.0199,
```

```
0.0239, 0.0279, 0.0319, 0.0359, 0.0398, 0.0438, 0.0478,
    0.0517]
# ... (full table omitted for brevity in document)

def sd_to_area(sd):
    """Convert z-score to proportion of area under normal
       curve"""
    sign = 1
    if sd < 0:
        sign = -1
    sd = math.fabs(sd)
    index = int(sd * 100)
    if len(area_to_sd_map) <= index:
        return sign * area_to_sd_map[-1]
    if index == (sd * 100):
        return sign * area_to_sd_map[index]
    return sign * (area_to_sd_map[index] +
        area_to_sd_map[index + 1]) / 2

def area_to_sd(area):
    """Convert proportion of area to z-score"""
    sign = 1
    if area < 0:
        sign = -1
    area = math.fabs(area)
    for a in range(len(area_to_sd_map)):
        if area == area_to_sd_map[a]:
            return sign * a / 100
        if 0 < a and area_to_sd_map[a - 1] < area <
            area_to_sd_map[a]:
            return sign * (a - 0.5) / 100
    return sign * (len(area_to_sd_map) - 1) / 100

def bootstrap(x):
    """Generate bootstrap sample by sampling with
       replacement"""
    samp_x = []
    for i in range(len(x)):
        samp_x.append(random.choice(x))
    return ~samp_x

def mean(grp):
    """Compute arithmetic mean"""
    return sum(grp) / float(len(grp))

def pairedtestpval(n1, n2, nof_iters):
    """Compute two-tailed p-value for paired test"""
    ndiff = sum([n1[i] - n2[i] for i in range(len(n1))])
    if ndiff < 0:
        nbig = n2
```

```
        nsmall = n1
        ndiff = -ndiff
    else:
        nbig = n1
        nsmall = n2

    bcount = 0
    for niter in range(nof_iters):
        tdiff = 0
        for i in range(len(nbig)):
            a = random.random()
            if a < 0.5:
                tdiff += nsmall[i] - nbig[i]
            else:
                tdiff += nbig[i] - nsmall[i]
        if tdiff > ndiff:
            bcount += 1
    return bcount / nof_iters

def pairedtestconf(n1, n2, nof_iters):
    """Compute standard and bias-corrected bootstrap CIs"""
    sample = [n1[i] - n2[i] for i in range(len(n1))]
    observed_mean = mean(sample)

    num_below_observed = 0
    out = []

    # Generate bootstrap samples
    for i in range(nof_iters):
        boot_mean = mean(bootstrap(sample))
        if boot_mean < observed_mean:
            num_below_observed += 1
        out.append(boot_mean)

    out.sort()

    # Standard confidence interval
    tails = (1 - conf_interval) / 2
    lower_bound = int(math.ceil(nof_iters * tails))
    upper_bound = int(math.floor(nof_iters * (1 - tails)))

    # Bias-corrected confidence interval
    p = num_below_observed / float(nof_iters)
    dist_from_center = p - 0.5
    z_0 = area_to_sd(dist_from_center)

    tail_sds = area_to_sd(conf_interval / 2)
    z_alpha_over_2 = 0 - tail_sds
    z_1_minus_alpha_over_2 = tail_sds
```

```

bias_corr_lower_bound = int(math.ceil(
    nof_iters * (0.5 + sd_to_area(z_alpha_over_2 + (2 *
        z_0)))
))
bias_corr_upper_bound = int(math.floor(
    nof_iters * (0.5 +
        sd_to_area(z_1_minus_alpha_over_2 + (2 * z_0)))
))

# Print results
print("Observed mean: %.4f" % observed_mean)
print("Standard 95%% CI: [%.4f, %.4f]" %
    (out[lower_bound], out[upper_bound]))
print("Bias-corrected 95%% CI: [%.4f, %.4f]" %
    (out[bias_corr_lower_bound],
        out[bias_corr_upper_bound]))

```

Appendix U.4. Statistical Interpretation

The bias-corrected percentile method adjusts for skewness in the bootstrap distribution by computing a bias-correction factor z_0 based on the proportion of bootstrap samples below the observed statistic. This correction shifts the confidence interval bounds to account for asymmetry, providing more accurate coverage when the sampling distribution is non-normal.

For the CloudNerd vs. OpenDeepResearch STS comparison on the 120-question benchmark (Table 11):

- The observed mean difference was $\bar{\Delta} = +0.024$
- The 95% bootstrap confidence interval was $[+0.006, +0.042]$
- The paired permutation test p -value was 0.011
- The interval excludes zero, confirming statistical significance at $\alpha = 0.05$

References

1. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.T.; Rocktäschel, S.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Adv. Neural Inf. Process. Syst.* **2021**, *33*, 9459–9474. <http://arxiv.org/abs/2005.11401>.
2. Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M.W. REALM: Retrieval-Augmented Language Model Pre-Training. 2020. Available online: <http://arxiv.org/abs/2002.08909> (accessed on).
3. Izacard, G.; Grave, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*; 2021. Available online: <http://arxiv.org/abs/2007.01282> (accessed on).
4. Chase, H. *LangChain: Building Applications with LLMs Through Composability*; Software; 2023.
5. Liu, J. *LlamaIndex: A Data Framework for LLM Applications*; Software; Formerly GPT Index; 2022.
6. Nguyen, T.; Chin, P.; Tai, Y.W. MA-RAG: Multi-Agent Retrieval-Augmented Generation via Collaborative Chain-of-Thought Reasoning. *arXiv* **2025**, arXiv:cs.CL/2505.20096. Available online: <http://arxiv.org/abs/2505.20096> (accessed on).
7. Dong, G.; Jin, J.; Li, X.; Zhu, Y.; Dou, Z.; Wen, J.R. RAG-Critic: Leveraging Automated Critic-Guided Agentic Workflow for Retrieval Augmented Generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, Vienna, Austria, 27 July 27–1 August 2025*; Volume 1: Long Papers, pp. 3551–3578. <https://doi.org/10.18653/v1/2025.acl-long.179>.
8. LlamaIndex. *Agentic Document Workflows: A Practical Guide*; Blog Post; Agentic Document Workflows (ADW) Reference Architecture; LlamaIndex: San Francisco, CA, USA, 2025.
9. Greyling, C. *Multi-Document Agentic RAG with LlamaIndex*; GitHub Notebook; LlamaIndex Multi-Document Agent Architecture Reference; LlamaIndex: San Francisco, CA, USA, 2024.

10. Jiang, S.; Xie, X.; Tang, R.; Wang, X.; Sun, K.; Li, G.; Xu, Z.; Xue, P.; Li, Z.; Fu, X. ARGUS: Retrieval-Augmented QA System for Government Services. *Electronics* **2025**, *14*, 2445. <https://doi.org/10.3390/electronics14122445>.
11. Wu, S.; Yacub, Z.; Shasha, D. DietNerd: A Nutrition Question-Answering System That Summarizes and Evaluates Peer-Reviewed Scientific Articles. *Appl. Sci.* **2024**, *14*, 9021. <https://doi.org/10.3390/app14199021>.
12. Hu, Y.; Xuan, W.; Zhou, Q.; Li, Z.; Li, Y.; Hu, J.; Fang, F. A Self-Correcting Agentic Graph RAG for Clinical Decision Support in Hepatology. *Front. Med.* **2025**, *12*, 1716327. <https://doi.org/10.3389/fmed.2025.1716327>.
13. Prabha, S.; Gomez-Cabello, C.A.; Haider, S.A.; Genovese, A.; Trabilsy, M.; Wood, N.G.; Bagaria, S.; Tao, C.; Forte, A.J. Enhancing Clinical Decision Support with Adaptive Iterative Self-Query Retrieval for Retrieval-Augmented Large Language Models. *Bioengineering* **2025**, *12*, 895. <https://doi.org/10.3390/bioengineering12080895>.
14. Li, H.; Chen, Y.; Hu, Y.; Ai, Q.; Chen, J.; Yang, X.; Yang, J.; Wu, Y.; Liu, Z.; Liu, Y. LexRAG: Benchmarking Retrieval-Augmented Generation in Multi-Turn Legal Consultation Conversation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*; **2025**. Available online: <http://arxiv.org/abs/2502.20640> (accessed on).
15. Zhao, X.; Liu, S.; Yang, S.Y.; Miao, C. MedRAG: Enhancing Retrieval-augmented Generation with Knowledge Graph-Elicited Reasoning for Healthcare Copilot. In *Proceedings of the ACM on Web Conference*; **2025**. Available online: <http://arxiv.org/abs/2502.04413> (accessed on).
16. Yang, M.; Ren, Y.; Opoku, D.O.; Li, R.; Ren, P.; Xing, C. DSRAG: A Domain-Specific Retrieval Framework Based on Document-derived Multimodal Knowledge Graph. In *International Conference on Web Information Systems and Applications*; Springer: Singapore, **2025**. Available online: <http://arxiv.org/abs/2509.10467> (accessed on).
17. Significant Gravitas. *AutoGPT: An Autonomous GPT-4 Experiment*; Software; **2023**.
18. Moura, J. *CrewAI: Framework for Orchestrating Role-Playing, Autonomous AI Agents*; Software; **2024**.
19. LangChain. *Open Deep Research*; LangGraph Supervisor–Researcher–Writer–Critic Baseline; Software; **2025**.
20. Dang, Y.; Qian, C.; Luo, X.; Fan, J.; Xie, Z.; Shi, R.; Chen, W.; Yang, C.; Che, X.; Tian, Y.; et al. Multi-Agent Collaboration via Evolving Orchestration. *NeurIPS 2025; Puppeteer orchestration paradigm. Adv. Neural Inf. Process. Syst.* **2025**, *38*, 165025–165059. Available online: <http://arxiv.org/abs/2505.19591> (accessed on).
21. Zhang, J.; Fan, Y.; Cai, K.; Sun, X.; Wang, K. OSC: Cognitive Orchestration through Dynamic Knowledge Alignment in Multi-Agent LLM Collaboration. *EMNLP 2025 Findings. Rn* **2025**, *100*, R2. Available online: <http://arxiv.org/abs/2509.04876> (accessed on).
22. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Guo, Q.; Wang, M.; et al. Retrieval-Augmented Generation for Large Language Models: A Survey. **2024**. Available online: <http://arxiv.org/abs/2312.10997> (accessed on).
23. Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; Choi, Y. The Curious Case of Neural Text Degeneration. *arXiv* **2020**, arXiv:1904.09751. Available online: <http://arxiv.org/abs/1904.09751> (accessed on).
24. Wiher, G.; Meister, C.; Cotterell, R. On Decoding Strategies for Neural Text Generators. *Trans. Assoc. Comput. Linguist.* **2022**, *10*, 997–1012. https://doi.org/10.1162/tacl_a_00502.
25. Ravfogel, S.; Goldberg, Y.; Goldberger, J. Conformal Nucleus Sampling. In *Proceedings of the Findings of the Association for Computational Linguistics: ACL 2023*; **2023**; pp. 27–34. <https://doi.org/10.18653/v1/2023.findings-acl.3>.
26. Nguyen, M.N.; Baker, A.; Neo, C.; Roush, A.; Kirsch, A.; Shwartz-Ziv, R. Turning Up the Heat: Min-p Sampling for Creative and Coherent LLM Outputs. In *Proceedings of the International Conference on Learning Representations; ICLR 2025 Oral*; **2025**. Available online: <http://arxiv.org/abs/2407.01082> (accessed on).
27. Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y.J.; Madotto, A.; Fung, P. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* **2023**, *55*, 1–38. <https://doi.org/10.1145/3571730>.
28. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv* **2023**, arXiv:2203.11171. Available online: <http://arxiv.org/abs/2203.11171> (accessed on).
29. Kadavath, S.; Conerly, T.; Askell, A.; Henighan, T.; Drain, D.; Perez, E.; Schiefer, N.; Hatfield-Dodds, Z.; DasSarma, N.; Tran-Johnson, E.; et al. Language Models (Mostly) Know What They Know. *arXiv* **2022**, arXiv:2207.05221. Available online: <http://arxiv.org/abs/2207.05221> (accessed on).
30. Lin, S.; Hilton, J.; Evans, O. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, Dublin, Ireland, 22–27 May 2022*; Available online: <http://arxiv.org/abs/2109.07958> (accessed on).
31. Wiher, G.; Meister, C.; Cotterell, R. On Decoding Strategies for Neural Text Generators. *Trans. Assoc. Comput. Linguist.* **2022**, *10*, 997–1012. https://doi.org/10.1162/tacl_a_00502.
32. Chang, C.C.; Reitter, D.; Aksitov, R.; Sung, Y.H. KL-Divergence Guided Temperature Sampling. *arXiv* **2023**, arXiv:2306.01286. Available online: <http://arxiv.org/abs/2306.01286> (accessed on).

33. Kadavath, S.; Conerly, T.; Askeell, A.; Henighan, T.; Drain, D.; Perez, E.; Schiefer, N.; Hatfield-Dodds, Z.; DasSarma, N.; Tran-Johnson, E.; et al. Language Models (Mostly) Know What They Know. *arXiv* **2022**, arXiv:2207.05221. Available online: <http://arxiv.org/abs/2207.05221> (accessed on).
34. Es, S.; James, J.; Espinosa-Anke, L.; Schockaert, S. Ragas: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, St. Julian's, Malta, 21–22 March 2025*. Available online: <http://arxiv.org/abs/2309.15217> (accessed on).
35. Cer, D.; Diab, M.; Agirre, E.; Lopez-Gazpio, I.; Specia, L. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics, Vancouver, BC, Canada, 3–4 August 2017; pp. 1–14. <https://doi.org/10.18653/v1/S17-2001>.
36. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 3–7 November 2019. Available online: <http://arxiv.org/abs/1908.10084> (accessed on).
37. Min, S.; Krishna, K.; Lyu, X.; Lewis, M.; Yih, W.T.; Koh, P.W.; Iyyer, M.; Zettlemoyer, L.; Hajishirzi, H. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Singapore, 6–10 December 2023*. Available online: <http://arxiv.org/abs/2305.14251> (accessed on).
38. Shasha, D.; Wilson, M. *Statistics is Easy!*; Synthesis Lectures on Mathematics and Statistics; Morgan and Claypool: San Rafael, CA, USA, 2010.
39. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Guo, Q.; Wang, M.; et al. Retrieval-Augmented Generation for Large Language Models: A Survey, 2024. Available online: <http://arxiv.org/abs/2312.10997> (accessed on).
40. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.t.; Rocktäschel, S.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Foundational RAG paper; alias of lewis2020retrieval. *Adv. Neural Inf. Process. Syst.* **2021**, *33*, 9459–9474. Available online: <http://arxiv.org/abs/2005.11401> (accessed on).
41. Nygard, M.T. *Release It! Design and Deploy Production-Ready Software*, 2 ed.; Pragmatic Bookshelf: 2018.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.