Evidence-Grounded Tree-Based Hypothesis Generation for Scientific Discovery

Anonymous Authors[1]

Abstract

AI scientists are advancing rapidly, yet most entangle evidence management with reasoning, limiting hypothesis generation to narrow, prior-driven contexts. To enable the sustained reasoning chains required for scientific ideation, we propose the Evidence-Grounded AI Scientist (EGAS), which strictly separates evidence curation from reasoning. A Context Optimizer curates and grounds evidence without interpretation, enabling a Tree-of-Thoughts Hypothesis Generator to explore diverse reasoning paths and produce novel, falsifiable hypotheses. Experimental outcomes feed back into the system, with a Diagnostician that flags anomalous signals before they corrupt downstream interpretation. This growing body of validated evidence improves subsequent ideation cycles. As a proof of concept, we demonstrate this closed-loop architecture as an integrated member of a drug discovery team, from target suggestion through wet-lab validation.

1. Introduction

An AI Scientist should operate as an independent yet collaborative member of the lab. Recent systems (Du et al., 2025; Lu et al., 2026; Swanson et al., 2025; Sui et al., 2026; Huang et al., 2025), have automated individual research stages with impressive results, but they remain stronger at well-scoped tasks than at ideation where genuine scientific discovery begins. Creative ideation arises when a researcher connects known and newly encountered evidence in unexpected ways. Yet the LLM prior works against this: trained on published literature, models favor likely continuations of existing knowledge over novel connections. (Shahhosseini et al., 2026; Liu et al., 2024). Retrieval augmentation inherits this bias rather than removing it, since the same prior shapes both what is retrieved (Abe et al., 2025; Scialom et al., 2021) and how it is weighted (Wu et al., 2024). Recent works indicate this limitation is better mitigated through structural design rather than prompting alone. Just as decoupling retrieval from generation reduces reliance on parametric priors(Asai et al., 2024; Sun et al., 2025), and separating planning from execution minimizes cognitive load(Erdogan et al., 2025; Liu et al., 2026), we apply a similar structural separation targeted directly at the upstream ideation failure.

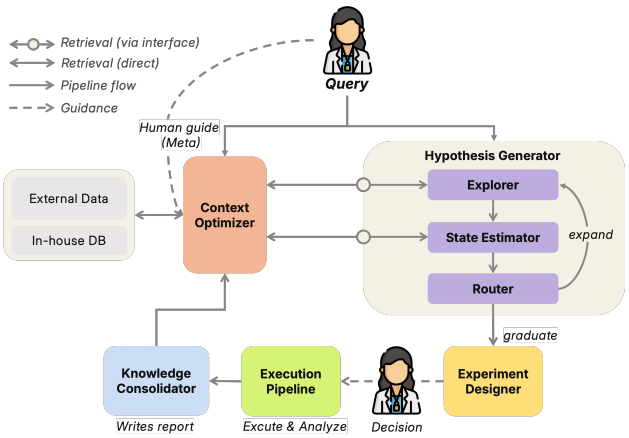


Figure 1. Overall pipeline of EGAS. Evidence curation (Context Optimizer) is strictly separated from reasoning (Hypothesis Generator). Human guidance is supported in multiple spots.

We propose the **Evidence-Grounded AI Scientist** (EGAS, Figure 1), built on two core principles. First, evidence curation is strictly separated from reasoning, preventing the LLM's prior from biasing the evidence base. Second, ideation is treated as a grounded search rather than single-pass generation: empirical results, including failures, feed back continuously to extend the team's reasoning capability over time. EGAS integrates ideation, experimental planning, and data analysis into a closed-loop, with human oversight embedded at every cognitive decision point (A.2).

2. Evidence-Grounded AI Scientist

EGAS supports creative ideation through three interconnected stages. First, the Context Optimizer (CO) curates evidence without bias. Second, the Hypothesis Generator (HG) explores grounded ideation across diverse reasoning

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Submitted to the AI for Science workshop (ICML 2026).

paths. Third, a collaborative pipeline designs and executes experiments, depositing all outcomes back as a compounding knowledge asset.

2.1. Context Optimizer

CO acts as a Librarian: it collects, grounds, and indexes evidence but never interprets, summarizes, or judges it(A.2). This separation of curation from interpretation yields two benefits. First, during reasoning, HG operates over an unnarrowed inference space because CO imposes no interpretive framing from the LLM's prior, preserving room for creative leaps. Second, after the fact, when a hypothesis seems implausible or unsound, its source can be traced to either the context or the reasoning, because the two responsibilities live in different modules. To preserve this separation, HG accesses evidence exclusively through CO's stateless retrieval, with all higher-order reasoning kept inside HG. CO also serves as the system's long-term memory. Every modification is append-only, so accumulated evidence compounds over time into a reusable scientific asset (B.1).

2.2. Hypothesis Generator

HG treats scientific ideation as a grounded Tree-of-Thoughts search, driven by three cognitive modules. The Explorer generates and expands hypotheses, the State Estimator evaluates them, and the Router decides the next search action. Building on evidence curated by CO, HG follows two complementary design principles. (A.2, B.2). The first is structurally induced diversity. Instead of relying on stochastic sampling, HG partitions retrieved evidence into distinct subsets and explores each through different reasoning modes. Sibling hypotheses differ along interpretable dimensions, not by random variation. The second is grounding-aware evaluation. Each hypothesis undergoes a two-level assessment to identify different sources of failure. Node-level evaluation ensures the hypothesis is evidence-backed and testable, while trajectory-level evaluation checks whether the hypothesis-producing step itself was justified given the evidence, reasoning mode, and query scope. This two-level evaluation enables the Router to optimize its actions by pruning unsound claims (node-level), expanding flawed but promising hypotheses (trajectory-level), or graduating ones that are sound on both levels. Through this framework, diversity drives the search beyond familiar priors, while rigorous evaluation grounds every creative step in retrieved evidence.

2.3. Collaborative End-to-End Discovery

A graduated hypothesis enters a verification pipeline where the Experiment Designer outlines the logical experiment and selects the appropriate verification module (In Silico or Wet-Lab), leaving specific software and protocol choices to the assigned agent. The In Silico Scientist executes computational pipelines, while the Wet-Lab Assistant generates reproducible, version-controlled protocols for human or robotic execution. Subsequently, the Wet-Lab Analyzer interprets the raw data. Before database integration, a Diagnostician intercepts anomalous signals, traces their root causes, and proposes protocol revisions to prevent corrupted measurements from distorting downstream hypotheses. Finally, the Knowledge Consolidator synthesizes all findings into standardized records, including negative results and diagnostic reports. This transforms the lab's experience into a growing asset that CO makes available to future reasoning. Throughout the cycle, human oversight remains central. Domain experts steer the flow of evidence by curating retained data and directing the acquisition of new evidence by choosing which hypothesis to test. Together, these interventions let human control govern *what the AI sees* independently of *how it reasons* (D).

3. Evaluation Roadmap

The primary evaluation uses a knowledge cutoff, testing whether EGAS generates novel hypotheses rather than regressing toward its prior. We restrict CO's retrieval to pre-cutoff evidence and ask whether EGAS can reproduce discoveries established in the literature afterward. (Narganes-Carlón et al., 2023). Beyond final hypothesis quality, we also evaluate the search process itself. We assess CO's retrieval and grounding quality and HG's reasoning-step quality using existing biomedical benchmarks (Miller et al., 2025; Gao et al., 2025). To evaluate the Diagnostician, we use custom datasets to measure its accuracy in anomaly detection and root-cause attribution. Wet-lab validation can expand progressively from narrow candidate verification to end-to-end therapeutic discovery. This sequential validation tests long-horizon reasoning, determining if accumulating evidence enhances subsequent hypothesis generation (E).

4. Discussions

EGAS targets the ideation bottleneck by strictly decoupling evidence curation from reasoning, enabling domain experts to steer discovery directly at the evidence layer. Beyond this, EGAS operates as a closed-loop system that transforms all empirical outcomes, including negative results and troubleshooting reports, into reusable scientific assets. Over the cycles, this compounding knowledge mitigates recurrent errors and drives increasingly robust hypothesis exploration. However, certain limitations remain. The CO creates a bottleneck dependent on accurate entity linking, and the HG's final hypothesis quality is inherently bounded by the LLM's reasoning capacity. Detailed module specifications, drug discovery case study, governance protocols, and cross-domain extension are provided in the Appendix.

References

Abe, K., Takeoka, K., Kato, M. P., and Oyamada, M. LLM-based query expansion fails for unfamiliar and ambiguous queries. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25)*, pp. 3035–3039, 2025. doi: 10.1145/3726302.3730222.

Arora, R. K., Wei, J., Hicks, R. S., Bowman, P., Quiñonero-Candela, J., Tsimpourlas, F., Sharman, M., Shah, M., Vallone, A., Beutel, A., Heidecke, J., and Singhal, K. HealthBench: Evaluating large language models towards improved human health. arXiv preprint arXiv:2505.08775, 2025.

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Bard, J., Rhee, S. Y., and Ashburner, M. An ontology for cell types. *Genome Biology*, 6(2):R21, 2005. doi: 10.1186/gb-2005-6-2-r21.

Bran, A. M., Cox, S., Schilter, O., Baldassari, C., White, A. D., and Schwaller, P. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6 (5):525–535, 2024. doi: 10.1038/s42256-024-00832-8.

Burley, S. K., Bhatt, R., Bhikadiya, C., Bi, C., Biester, A., Biswas, P., Bittrich, S., Blaumann, S., Brown, R., Chao, H., et al. Updated resources for exploring experimentally-determined PDB structures and Computed Structure Models at the RCSB Protein Data Bank. *Nucleic Acids Research*, 53(D1):D564–D574, 2025. doi: 10.1093/nar/gkae1091.

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2318–2335, 2024a. doi: 10.18653/v1/2024.findings-acl.137.

Chen, T., Lin, B., Yuan, Z., Zou, Q., He, H., Goyal, A., Ong, Y., and Liu, D. HypoSpace: Evaluating LLM creativity as set-valued hypothesis generators under underdetermination. arXiv preprint arXiv:2510.15614, 2025.

Chen, X., Wang, T., Guo, T., Guo, K., Zhou, J., Li, H., Zhuge, M., Schmidhuber, J., Gao, X., and Zhang, X. ScholarChemQA: Unveiling the power of language models in chemical research question answering. arXiv preprint arXiv:2407.16931, 2024b.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Du, Y., Yu, B., Liu, T., Shen, T., Chen, J., Rittig, J. G., Sun, K., Zhang, Y., Song, Z., Zhou, B., Masschelein, C., Wang, Y., Wang, H., Jia, H., Zhang, C., Zhao, H., Ester, M., Head-Gordon, T., Gomes, C. P., Sun, H., Duan, C., Schwaller, P., and Jin, W. Accelerating scientific discovery with autonomous goal-evolving agents. arXiv preprint arXiv:2512.21782, 2025.

Erdogan, L. E., Lee, N., Kim, S., Moon, S., Furuta, H., Anumanchipalli, G., Keutzer, K., and Gholami, A. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Gao, L., Ma, X., Lin, J., and Callan, J. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1762–1777, 2023. doi: 10.18653/v1/2023.acl-long.99.

Gao, S., Zhu, R., Kong, Z., Su, X., Ginder, C., Aldogom, S., Das, I., Evans, T., Tsiligkaridis, T., and Zitnik, M. CURE-Bench: Competition on reasoning models for drug decision-making in precision therapeutics, 2025. URL https://openreview.net/forum?id=rD9YGynuT8. NeurIPS 2025 Competition Track.

Garda, S., Weber-Genzel, L., Martin, R., and Leser, U. BELB: A biomedical entity linking benchmark. *Bioinformatics*, 39(11):btad698, 2023. doi: 10.1093/bioinformatics/btad698.

Gero, K., Liu, V., and Chilton, L. Sparks: Inspiration for science writing using language models. In *Proceedings of the First Workshop on Intelligent and Interactive Writing Assistants (In2Writing 2022)*, 2022.

Gottweis, J., Weng, W.-H., Daryin, A., Tu, T., Palepu, A., Sirkovic, P., Myaskovsky, A., Weissenberger, F., Rong, K., Tanno, R., Saab, K., Popovici, D., Blum, J., Zhang, F., Chou, K., Hassidim, A., Gokturk, B., Vahdat, A., Kohli, P., Matias, Y., Carroll, A., Kulkarni, K., Tomasev, N., Guan, Y., Dhillon, V., Vaishnav, E. D., Lee, B., Costa, T. R. D., Penadés, J. R., Peltz, G., Xu, Y., Pawlosky, A., Karthikesalingam, A., and Natarajan, V. Towards an AI co-scientist. arXiv preprint arXiv:2502.18864, 2025.

Guo, S., Shariatmadari, A. H., Xiong, G., Huang, A., Xie, E., Bekiranov, S., and Zhang, A. IdeaBench: Benchmarking large language models for research idea generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.

Guo, T., Guo, K., Nan, B., Liang, Z., Guo, Z., Chawla, N. V., Wiest, O., and Zhang, X. What indeed can GPT models do in chemistry? a comprehensive benchmark on eight tasks. arXiv preprint arXiv:2305.18365, 2023.

Huang, K., Zhang, S., Wang, H., Qu, Y., Lu, Y., Roohani, Y., Li, R., Qiu, L., Li, G., Zhang, J., Yin, D., Marwaha, S., Carter, J. N., Zhou, X., Wheeler, M., Bernstein, J. A., Wang, M., He, P., Zhou, J., Snyder, M., Cong, L., Regev, A., and Leskovec, J. Biomni: A general-purpose biomedical AI agent. *bioRxiv*, 2025. doi: 10.1101/2025.05.30.656746.

Jin, Q., Dhingra, B., Liu, Z., Cohen, W. W., and Lu, X. PubMedQA: A dataset for biomedical research question answering. 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)*, pp. 2567–2577, 2019. doi: 10.18653/v1/D19-1259.

Kanehisa, M., Furumichi, M., Sato, Y., Matsuura, Y., and Ishiguro-Watanabe, M. KEGG: Biological systems database as a model of the real world. *Nucleic Acids Research*, 53(D1):D672–D677, 2025. doi: 10.1093/nar/gkae909.

Knox, C., Wilson, M., Klinger, C. M., Franklin, M., Oler, E., Wilson, A., Pon, A., Cox, J., Chin, N. E. L., Strawbridge, S. A., Garcia-Patino, M., Kruger, R., Sivakumaran, A., Sanford, S., Doshi, R., Khetarpal, N., Fatokun, O., Doucet, D., Zubkowski, A., Rayat, D. Y., et al. DrugBank 6.0: The DrugBank knowledgebase for 2024. *Nucleic Acids Research*, 52(D1):D1265–D1275, 2024. doi: 10.1093/nar/gkad976.

Krithara, A., Nentidis, A., Bougiatiotis, K., and Paliouras, G. BioASQ-QA: A manually curated corpus for biomedical question answering. *Scientific Data*, 10(170), 2023. doi: 10.1038/s41597-023-02068-4.

Laurent, J. M., Janizek, J. D., Ruzo, M., Hinks, M. M., Hammerling, M. J., Narayanan, S., Ponnapati, M., White, A. D., and Rodriques, S. G. LAB-Bench: Measuring capabilities of language models for biology research. arXiv preprint arXiv:2407.10362, 2024.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020. doi: 10.1093/bioinformatics/btz682.

Leung, H., Duan, C., Gou, W., Chen, J., Xin, Y., Zheng, Z., Naumov, V., Gennert, D., Zhang, M., Aliper, A., Ren, F., Izumchenko, E., Pun, F. W., and Zhavoronkov, A. Advancing target discovery through disease-specific integration of multi-modal target identification models and comprehensive target benchmarking system. *Scientific Reports*, 2026. doi: 10.1038/s41598-026-47765-3.

Li, Y., Yue, X., Liao, Z., and Sun, H. AttributionBench: How hard is automatic attribution evaluation? In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14919–14935, 2024. doi: 10.18653/v1/2024.findings-acl.886.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Liu, H., Huang, S., Hu, J., Zhou, Y., and Tan, C. HypoBench: Towards systematic and principled benchmarking for hypothesis generation. arXiv preprint arXiv:2504.11524, 2025.

Liu, Z., Yu, D., and He, H. On the role of model prior in real-world inductive reasoning. arXiv preprint arXiv:2412.13645, 2024.

Liu, Z., Zhou, Z., Li, Y., Hu, M., Pan, Y., Xu, Z., and Chen, Y. SEVADE: Self-evolving multi-agent analysis with decoupled evaluation for hallucination-resistant irony detection. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

Lu, C., Lu, C., Lange, R. T., Yamada, Y., Hu, S., Foerster, J., Ha, D., and Clune, J. Towards end-to-end automation of AI research. *Nature*, 651:914–919, 2026. doi: 10.1038/s41586-026-10265-5.

Milacic, M., Beavers, D., Conley, P., Gong, C., Gillespie, M., Griss, J., Haw, R., Jassal, B., Matthews, L., May, B., Petryszak, R., Ragueneau, E., Rothfels, K., Sevilla, C., Shamovsky, V., Stephan, R., Tiwari, K., Varusai, T., Weiser, J., Wright, A., Wu, G., Stein, L., Hermjakob, H., and D'Eustachio, P. The Reactome pathway knowledgebase 2024. *Nucleic Acids Research*, 52(D1):D672–D678, 2024. doi: 10.1093/nar/gkad1025.

Miller, H. E., Greenig, M., Tenmann, B., and Wang, B. BioML-bench: Evaluation of AI agents for end-to-end biomedical ML. *bioRxiv*, 2025. doi: 10.1101/2025.09.01.673319.

Min, S., Krishna, K., Lyu, X., Lewis, M., t. Yih, W., Koh, P. W., Iyyer, M., Zettlemoyer, L., and 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 (EMNLP)*, pp. 12076–12100, 2023. doi: 10.18653/v1/2023.emnlp-main.741.

Mirza, A. et al. A framework for evaluating the chemical knowledge and reasoning abilities of large language models against the expertise of chemists. *Nature Chemistry*, 17:1027–1034, 2025. doi: 10.1038/s41557-025-01815-x.

Mitchener, L., Laurent, J. M., Tenmann, B., Narayanan, S., Wellawatte, G. P., White, A. D., Sani, L., and Rodriques, S. G. BixBench: A comprehensive benchmark for LLM-based agents in computational biology. arXiv preprint arXiv:2503.00096, 2025.

Mungall, C. J., Torniai, C., Gkoutos, G. V., Lewis, S. E., and Haendel, M. A. Uberon, an integrative multi-species anatomy ontology. *Genome Biology*, 13(1):R5, 2012. doi: 10.1186/gb-2012-13-1-r5.

Narganes-Carlón, D., Crowther, D. J., and Pearson, E. R. A publication-wide association study (PWAS), historical language models to prioritise novel therapeutic drug targets. *Scientific Reports*, 13:8366, 2023. doi: 10.1038/s41598-023-35597-4.

Neumann, M., King, D., Beltagy, I., and Ammar, W. ScispaCy: Fast and robust models for biomedical natural language processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pp. 319–327, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5034.

Phan, L., Gatti, A., Han, Z., Li, N., Hu, J., Zhang, H., Zhang, C. B. C., Shaaban, M., Ling, J., Shi, S., et al. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. Originally titled "Humanity's Last Exam," arXiv:2501.14249, 2025.

Phylo Bio. Evaluating AI agents in biology. Blog post, 2026. URL https://phylo.bio/blog/evaluating-ai-agents-in-biology.

Qi, B., Zhang, K., Tian, K., Li, H., Chen, Z.-R., Zeng, S., Hua, E., Hu, J., and Zhou, B. Large language models as biomedical hypothesis generators: A comprehensive evaluation. arXiv preprint arXiv:2407.08940, 2024.

Santhanam, K., Khattab, O., Saad-Falcon, J., Potts, C., and Zaharia, M. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 3715–3734, 2022. doi: 10.18653/v1/2022.naacl-main.272.

Sayers, E. W., Beck, J., Bolton, E. E., Brister, J. R., Chan, J., Connor, R., Feldgarden, M., Fine, A. M., Funk, K., Hoffman, J., Kannan, S., Kelly, C., Klimke, W., Kim, S., Lathrop, S., Marchler-Bauer, A., Murphy, T. D., O'Sullivan, C., Schmieder, E., Skripchenko, Y., Stine, A., Thibaud-Nissen, F., Wang, J., Ye, J., Zellers, E., Schneider, V. A., and Pruitt, K. D. Database resources of the National Center for Biotechnology Information in 2025. *Nucleic Acids Research*, 53(D1):D20–D29, 2025. doi: 10.1093/nar/gkae979.

Sciavolino, C., Zhong, Z., Lee, J., and Chen, D. Simple entity-centric questions challenge dense retrievers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6138–6148, 2021. doi: 10.18653/v1/2021.emnlp-main.496.

Seal, R. L., Braschi, B., Gray, K., Jones, T. E. M., Tweedie, S., Haim-Vilmovsky, L., and Bruford, E. A. Genenames.org: The HGNC resources in 2023. *Nucleic Acids Research*, 51(D1):D1003–D1009, 2023. doi: 10.1093/nar/gkac888.

Shahhosseini, F., Marioriyad, A., Momen, A., Baghshah, M. S., Rohban, M. H., and Javanmard, S. H. Large language models for scientific idea generation: A creativity-centered survey. arXiv preprint arXiv:2511.07448, 2026.

Sprueill, H. W., Edwards, C., Olarte, M. V., Sanyal, U., Ji, H., and Choudhury, S. Monte Carlo thought search: Large language model querying for complex scientific reasoning in catalyst design. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. doi: 10.18653/v1/2023.findings-emnlp.560.

Sprueill, H. W., Edwards, C. N., Agarwal, K., Olarte, M. V., Sanyal, U., Johnston, C., Liu, H., Ji, H., and Choudhury, S. ChemReasoner: Heuristic search over a large language model's knowledge space using quantum-chemical feedback. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Sui, P., Li, M. M., Gao, S., Shen, W., Giunchiglia, V., Shen, A., Huang, Y., Kong, Z., and Zitnik, M. Medea: An omics AI agent for therapeutic discovery. *bioRxiv*, 2026. doi: 10.64898/2026.01.16.696667.

Sun, Z., Zang, X., Zheng, K., Song, Y., Xu, J., Zhang, X., Yu, W., Song, Y., and Li, H. ReDeEP: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Swanson, K., Wu, W., Bulaong, N. L., Pak, J. E., and Zou, J. The virtual lab of AI agents designs new SARS-CoV-2 nanobodies. *Nature*, 646:716–723, 2025. doi: 10.1038/s41586-025-09442-9.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In

Advances in Neural Information Processing Systems 34 (NeurIPS 2021) Datasets and Benchmarks Track, 2021.

The UniProt Consortium. UniProt: The universal protein knowledgebase in 2025. *Nucleic Acids Research*, 53(D1): D609–D617, 2025. doi: 10.1093/nar/gkae1010.

Tyagin, I. and Safro, I. Dyport: Dynamic importance-based biomedical hypothesis generation benchmarking technique. *BMC Bioinformatics*, 25(213), 2024. doi: 10.1186/s12859-024-05812-8.

Varadi, M., Bertoni, D., Magana, P., Paramval, U., Pidruchna, I., Radhakrishnan, M., Tsenkov, M., Nair, S., Mirdita, M., Yeo, J., Kovalevskiy, O., Tunyasuvunakool, K., Laydon, A., Žídek, A., Tomlinson, H., Hariharan, D., Abrahamson, J., Green, T., Jumper, J., Birney, E., Steinegger, M., Hassabis, D., and Velankar, S. AlphaFold Protein Structure Database in 2024: Providing structure coverage for over 214 million protein sequences. *Nucleic Acids Research*, 52(D1):D368–D375, 2024. doi: 10.1093/nar/gkad1011.

Vasilevsky, N. A., Matentzoglu, N. A., Toro, S., Flack, J. E., Hegde, H., Unni, D. R., Alyea, G. F., Amberger, J. S., Babb, L., Balhoff, J. P., et al. Mondo: Unifying diseases for the world, by the world. *medRxiv*, 2022. doi: 10.1101/2022.04.13.22273750. Preprint.

Wang, L., Yang, N., Huang, X., Jiao, B., Yang, L., Jiang, D., Majumder, R., and Wei, F. Text embeddings by weakly-supervised contrastive pre-training. arXiv preprint arXiv:2212.03533, 2022.

Wei, C.-H., Allot, A., Leaman, R., and Lu, Z. PubTator central: Automated concept annotation for biomedical full text articles. *Nucleic Acids Research*, 47(W1):W587–W593, 2019. doi: 10.1093/nar/gkz389.

Weir, J. A., Krebs, Y., and Chen, F. Sample barcoding-associated technical variation in probe-based single-cell RNA sequencing. *bioRxiv*, 2026. doi: 10.64898/2026.04.06.716804.

Wu, K., Wu, E., and Zou, J. ClashEval: Quantifying the tug-of-war between an LLM's internal prior and external evidence. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024) Datasets and Benchmarks Track*, 2024. doi: 10.52202/079017-1053.

Yang, F., Ye, C., Ma, M. D., Xiao, Y., Yang, M., and Wang, W. BioVerge: A comprehensive benchmark and study of self-evaluating agents for biomedical hypothesis generation. arXiv preprint arXiv:2511.08866, 2025.

Yang, Z., Liu, W., Gao, B., Xie, T., Li, Y., Ouyang, W., Poria, S., Cambria, E., and Zhou, D. MOOSE-Chem: Large language models for rediscovering unseen chemistry scientific hypotheses. arXiv preprint arXiv:2410.07076, 2024.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.

Zdrazil, B., Felix, E., Hunter, F., Manners, E. J., Blackshaw, J., Corbett, S., de Veij, M., Ioannidis, H., Lopez, D. M., Mosquera, J. F., Magarinos, M. P., Bosc, N., Arcila, R., Kizilören, T., Gaulton, A., Bento, A. P., Adasme, M. F., Monecke, P., Landrum, G. A., and Leach, A. R. The ChEMBL database in 2023: A drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic Acids Research*, 52(D1):D1180–D1192, 2024. doi: 10.1093/nar/gkad1004.

Zhong, X., Jin, B., Ouyang, S., Shen, Y., Jin, Q., Fang, Y., Lu, Z., and Han, J. Benchmarking retrieval-augmented generation for chemistry. arXiv preprint arXiv:2505.07671, 2025.

Appendix Contents

A. Framework Overview

A.1. Framework Design

The Evidence-Grounded AI Scientist (EGAS) separates evidence curation from reasoning so that each can be independently examined, corrected, and improved. Evidence curation and hypothesis reasoning are handled by two distinct modules, a Context Optimizer and a Hypothesis Generator, connected through stateless retrieval APIs. **Context Optimizer** acts as a "Librarian": it gathers, indexes, and exposes evidence from external literature, public databases, and in-house experimental results. However, it never interprets, summarizes, or judges any piece of evidence (B.1). **Hypothesis Generator** treats scientific ideation as a grounded Tree-of-Thoughts search, combining, contrasting, and reframing retrieved evidence across diverse reasoning paths to produce hypotheses that are novel yet falsifiable (B.2).

A cycle proceeds as follows. Hypothesis Generator generates evidence-grounded hypotheses through its internal branch search. When Hypothesis Generator evaluates a hypothesis as specific, grounded, and experimentally actionable, it is forwarded to the Experiment Designer (B.3). A human reviewer then selects which graduated hypotheses to verify, and each selected hypothesis proceeds through an *experimental episode*, comprising experiment design, execution, and analysis. If an experimental module surfaces an anomaly, the Diagnostician (C.1) is invoked. All outcomes are recorded by the Knowledge Consolidator (B.7) and incorporated into the Internal branch as new evidence for subsequent cycles. By default, the process continues across cycles as long as retained hypotheses remain; at each cycle boundary, the human reviewer may also decide to terminate the search. Modules are connected by explicit interfaces and share several pieces of common infrastructure (the Diagnostician, Internal branch, and State Memory), along with a uniform provenance convention that each module follows when producing records (C).

Inputs.

- **User query.** High-level goals aiming for open-ended discovery are the default input, though narrowly scoped queries are also accepted when the human already has a prior commitment (e.g., target and modality fixed). *Examples*: "design a small molecule binder for target TCF7"; "propose a rejuvenation target capable of reversing T cell exhaustion and design a drug for that target."
- **Human guide (optional).** Domain-expert constraints, preferences, and prior commitments that persist across cycles as Meta evidence (B.1.3). A human guide may intervene during context optimization (e.g., directing Context Optimizer to prioritize retrieval along a particular entity, or deprecating an evidence record), shaping the evidence available to Hypothesis Generator (B.1).
- **External databases.** Standard biomedical databases and structural resources, public literature, and clinical trial records, reached through source-appropriate queries orchestrated by the Context Optimizer (B.1).
- **In-house database.** Prior experimental data, protocol variants, and diagnostic reports, including both pre-existing archives (e.g., prior experiments from the same or related projects) and reports written by earlier cycles of the pipeline via the Knowledge Consolidator (B.7).

Outputs.

- **Tested hypotheses.** All hypotheses that underwent experimental verification, each with its verdict (support, refute, or inconclusive) and full provenance chain.
- **Hypothesis search history.** The complete search trajectory archived in Context Optimizer's StateMemoryArchive (B.1.1, B.2.6): every expanded, pruned, and terminated branch across all cycles, including the evidence and reasoning mode that produced each node. Failed and discarded directions are preserved alongside successful ones, making the full exploratory process a retrievable asset for future campaigns.
- **Experimental records.** The structured records produced by the Knowledge Consolidator (B.7) and stored in the in-house database and Internal branch: experiment plans, results, diagnostic reports, and troubleshooting episodes for every experimental episode across all cycles. Negative results and failure case studies are indexed under the same schema as external evidence and remain retrievable in future research campaigns.

A.2. Key Concepts and Rationale

The framework's design follows one central commitment, which has two major payoffs.

> **Central commitment.** Evidence curation and hypothesis reasoning are handled by separate modules connected only through stateless retrieval APIs, so that each can be examined, corrected, and improved independently.

When evidence retrieval and hypothesis reasoning share the same module, the LLM's prior shapes both what evidence is selected and how it is interpreted, narrowing the space of hypotheses the system can reach. Open-ended scientific inquiry, which requires sustained chains of reasoning across unfamiliar territory, is especially vulnerable to this narrowing.

EGAS assigns retrieval and reasoning to two separate modules (A.1). The query Hypothesis Generator sends to Context Optimizer, the evidence Context Optimizer returns, and the judgment Hypothesis Generator draws from it are all explicit, recorded artifacts. A weak hypothesis can therefore be traced: was the query poorly formed, was the retrieved evidence insufficient, or did the reasoner misuse what it had? Retrieval and reasoning can then be improved independently. Any output can be traced back through an unambiguous chain of decisions, making the system both auditable and improvable, whether by self-correction or human intervention.

Cognition is further separated from execution under a similar philosophy: reasoning about what to do (hypothesis generation, experiment selection, diagnosis) is structurally separated from doing it (running computational tools, executing protocols, processing assay outputs). The human guide, when engaged, acts on the cognition side: their input changes what is reasoned about, not how execution is performed.

Payoff 1: decomposition of aspirational goals for open-ended discovery. Because Context Optimizer and Hypothesis Generator are decoupled, Hypothesis Generator can accept high-level, aspirational goals and expand each one into a tree of progressively more specific and falsifiable claims. This is a capability prior AI scientists largely lack: most focus on sub-problems a human has already planned out, such as retrieval-style question answering or problems with a fixed, optimizable objective like drug design conditioned on a specified target; autonomous decomposition of open-ended goals is flagged as one of the main future directions the field should pursue (Du et al., 2025).

The root of the tree is the original goal; early expansions produce goal-level claims (a candidate target, a candidate modality, a selectivity or developability constraint), each framed as a sub-hypothesis rather than a task to be solved; later expansions refine these into more specific claims until one is concrete and testable enough to hand off for verification. Sub-hypotheses are not limited to the therapeutic target itself: Hypothesis Generator can also branch into meta-hypotheses about the evidence it uses, such as whether a particular dataset carries a platform-specific artifact that would confound downstream analyses. Resolving such meta-hypotheses through ordinary verification cycles lets the outcome persist in the in-house database and inform all future cycles that touch the same data source.

Within this tree search, Hypothesis Generator is organized around two complementary design principles. The first is *structurally induced diversity*: the Explorer partitions each evidence pool into distinct subsets and assigns each a different reasoning mode (causal, analogical, mechanism proposal, or gap-filling), so that sibling hypotheses differ along interpretable axes rather than by stochastic variation (B.2.2). The second is *grounding-aware evaluation*: the State Estimator decomposes each hypothesis into semantic units, checks their evidential support, and produces node-level scores (is the claim sound?) alongside trajectory-level attribution (did the reasoning step that produced it go wrong?). These signals let the search prune unsupported claims, re-expand promising but flawed directions, and graduate well-grounded hypotheses for verification (B.2.3).

Payoff 2: evidence accumulation for long-horizon discovery. Experimental verification of a single graduated hypothesis may take days to weeks; answering an open-ended goal typically requires many such cycles. Each cycle's findings, including negative results and diagnostic reports, are recorded by the Knowledge Consolidator (B.7) and incorporated into the Internal branch as structured, append-only records; subsequent cycles retrieve these through Context Optimizer on the same terms as external branch (B.1.3, C.3, C.4). Cross-cycle continuity is carried by the deposited records and their provenance chains rather than by any persistent reasoning state. Over successive cycles, this compounding body of validated evidence improves subsequent ideation, transforming the lab's accumulated experience into a growing scientific asset.

Experimental integrity is ensured by the Diagnostician (C.1), which assesses whether an observation is reliable enough to be used as evidence, flagging process-level anomalies such as failed controls or platform-level anomalies such as known model

biases. The verdict on whether the hypothesis is supported, refuted or inconclusive is issued by the assigned experimental module against the criteria pre-specified by the Experiment Designer (B.3), with the Diagnostician's report as one input. Where relevant, the troubleshooting episode enters the Internal branch as a failure case study, enabling the system to recognize recurrent failure patterns across future experiments.

B. Modules

B.1. Context Optimizer

Role. The Context Optimizer is a retrieval-only "librarian" module. It gathers, indexes, and serves evidence from both external databases and the in-house databases without interpreting it, ranking its relevance, or assigning polarity. All higher-order reasoning tasks, such as combining evidence or resolving contradictions, are strictly delegated to the Hypothesis Generator. By decoupling these processes, we can easily determine whether an unexpected hypothesis stems from incomplete evidence or flawed reasoning.

Context Optimizer organizes this information into a four-branch context tree consisting of External, Internal, StateMemoryArchive, and Meta branches. Each piece of evidence is stored as an atomic claim tied to a single raw data source. We also use Named Entity Recognition (NER) and ontology-linking to extract entities, allowing for entity-centric search (B.1.2). To maintain complete temporal provenance, the system treats all context modifications as append-only operations and never deletes existing records (B.1.3). Furthermore, Hypothesis Generator accesses this tree strictly through seven stateless APIs. Because Context Optimizer does not track prior interactions, it remains completely agnostic to Hypothesis Generator's search strategy (B.1.4).

Trigger. Context Optimizer activates in three specific situations. First, it builds an initial context tree when a new user query arrives. Second, it activates during cycle-time when the Knowledge Consolidator (B.7) submits experimental results. Context Optimizer deposits the raw data and extracts the relevant evidence into the Internal branch. Third, it triggers when a human guide steps in to remove, modify, or refocus evidence, or to reset the context entirely. Further details for each trigger are provided in B.1.3.

Process. When a query comes in, Context Optimizer breaks it down into distinct entities and concepts. This is the only step where Context Optimizer performs any interpretive work, and it can be subject to human review. The results are logged in the Meta branch to guide the retrieval plan. From there, Context Optimizer translates the concepts into queries tailored to specific sources, such as PubMed, clinical databases, or internal schemas. It then collects the raw data into a shared store and extracts the evidence records. For text artifacts, an LLM scans for factual and causal claims while preserving their exact source pointers. Structured database records, on the other hand, are converted using deterministic templates without any LLM involvement. Once the extraction is complete, Context Optimizer builds an entity index and checks for any remaining gaps in coverage. If it finds new entities appearing frequently, it triggers a second retrieval round. To keep temporal traceability intact, the system limits these iterations to two or three rounds using a `cycle_stage` counter.

Output. Context Optimizer outputs a populated context tree along with an entity index. Hypothesis Generator's Action module then accesses this data entirely through pull-based requests via the seven stateless APIs.

B.1.1. Context Schema

The context tree is organized into four top-level branches:

- **External.** Published literature, public databases, and clinical trial records, relying on peer review or community curation for credibility.
- **Internal.** Evidence records extracted from in-house experimental results submitted by the Knowledge Consolidator (B.7), keeping them securely accessible only to the research group.
- **StateMemoryArchive.** Per-cycle snapshots of Hypothesis Generator's State Memory, capturing the complete hypothesis tree and global search state. Unlike the other branches, this stores reasoning artifacts rather than empirical claims, acting as an immutable archive for cross-cycle warm-starts.
- **Meta.** The query decomposition, human-guide directives, and system-level annotations. Rather than carrying empirical claims, this branch records managerial decisions about how the evidence should be handled.

Context Schema

query: The original user query and its entity decomposition.
branches: Four top-level branches organized by evidence authority
- **External**: Published literature, public databases, clinical trials
- **Internal**: In-house experimental results and prior pipeline outputs
- **StateMemoryArchive**: State Memory Snapshots, comprising the global state and node records
- **Meta**: Query decomposition, human-guide directives, system annotations

raw_data_store: Shared database holding original artifacts, outside the tree.
entity_index: Projection mapping each canonical entity ID to its evidence records
cycle_stage: Counter tracking retrieval iteration for temporal filtering

Figure 2. Context schema of the Context Optimizer.

Evidence Schema

evidence_id: Unique identifier for this evidence record
content: The atomic claim extracted from a single raw data item
entities:
- canonical_id: Ontology identifier. Null if unresolved
- surface: Surface form as it appears in the source text
- type: Biomedical type(Gene, Protein, Disease, etc)

source: Provenance pointer to the originating raw data
- raw_data_id: Identifier of the raw data item
- span: Exact location within the raw data

originating_hypothesis_node_id: Pointer to the hypothesis node that prompted the experiment.
verdict: Analyzer-assigned outcome (support / refute / inconclusive).
branch_path: Position in the context tree
cycle_stage: Retrieval iteration in which this evidence was added
status: Whether this record is currently valid.
superseded_by: ID of the replacing evidence record if any
extracted_at: When this record was created.
deprecated_at: When this record was deprecated, if applicable

Figure 3. Evidence record schema used in the Context Optimizer.

Below these main branches, a secondary layer categorizes the evidence records into flat lists based on source types, such as literature, clinical data, or assay results. Importantly, raw data itself is not stored as a node within the tree. Instead, it resides in a separate shared database. Each evidence record simply points to its originating raw data item using an identifier and a specific span. This architectural separation keeps the context tree shallow, prevents the duplication of large artifacts, and allows multiple evidence records to reference the same raw data without redundancy.

Evidence record. Each record contains several key fields, including the core atomic claim, extracted entities, source pointers, branch path, cycle stage, operational status, and relevant timestamps. For records in the Internal branch extracted from experimental episodes, the system includes two additional fields: an `originating_hypothesis_node_id`, which points back to the specific hypothesis node that initially prompted the experiment, and a verdict, which preserves the Analyzer-assigned outcome (support / refute / inconclusive) for that hypothesis. Both fields remain null or absent for records in the External and Meta branches. A strict rule governs all these records: every piece of evidence must be grounded in exactly one raw data item. Consequently, a synthesized claim that merges multiple sources cannot qualify as a standalone evidence record. When a piece of evidence becomes useless, the system deprecates it rather than deleting it. It changes a `superseded_by` field to link directly to the newer, replacing record. The system then appends the reason for deprecation as new Meta evidence. While Hypothesis Generator retrieves active evidence by default, it can still query these deprecated records if necessary.

State Memory Snapshot. Because the StateMemoryArchive branch stores reasoning artifacts rather than empirical claims, it relies on a specialized schema. Each snapshot functions as an immutable archive entry, uniquely identified by its tree and cycle IDs. Once the system writes this snapshot at the end of a cycle, the record remains permanently unchanged.

The snapshot itself is divided into two primary sections: a global state and a map of node records. The global state captures the overarching context of the search. It records the original user query, the seed hypothesis, the current cycle ID, and a list of node IDs retained for the next cycle, along with a simple flag indicating whether any nodes remain expandable.

Meanwhile, the node records map organizes individual nodes using their specific hypothesis IDs. Each node begins with standard identifiers that define its structural position in the tree, such as its parent pointer, children, depth, and creation cycle. Beyond these basic identifiers, the node holds a comprehensive, multi-part history of its lifecycle, organized into specific traces.

Different modules write these traces at various stages of the pipeline. The Explorer module contributes a creation trace—logging the actual hypothesis claim, its reasoning mode, and the supporting evidence—as well as an expansion trace that tracks the specific queries and retrieved evidence pools used to generate child nodes. Following this, the State Estimator appends an extensive evaluation trace. This detailed section captures verification data, node-level state scores (like validity, expand gain, and experiment readiness), and path-level attribution signals that assess factors such as evidence adequacy and distortion risk. Finally, a router trace logs the exact search-control decision made during that cycle.

To maintain continuity across multiple cycles, the system carries over successful "graduated" leaves into the subsequent cycle's State Memory. Furthermore, whenever a new experiment produces internal evidence, the system uses the `originating_hypothesis_node_id` to link those empirical findings directly back to the historical reasoning snapshot that originally inspired them.

B.1.2. Entity and Entity Index

Entities exist as attributes of evidence records, not as nodes in the context tree. Treating entities as nodes would create significant structural problems. For instance, an evidence record mentioning multiple entities would either be forced under a single parent or duplicated across several. Furthermore, every new entity discovered during a cycle would require a structural update to the tree. Ultimately, defining entities as attributes is the most effective way to maintain a scalable and stable data architecture.

Entity extraction Every entity extraction unfolds in two stages. First, **Named Entity Recognition (NER)** identifies entity mentions within the text and categorizes them into biomedical schemas, such as genes, proteins, or diseases. This step relies on domain-specific tools like BioBERT (Lee et al., 2020), SciSpacy (Neumann et al., 2019), or PubTator (Wei et al., 2019). Second, the system performs **Ontology linking** to map each mention to a canonical ID in established databases. For example, genes and proteins map to HGNC and UniProt, while diseases link to MONDO (refer to C.2 for a complete list). If a mention cannot be resolved, the system assigns a null ID, leaving it searchable only by its surface form. However, human guides can manually register these mappings if necessary.

The entity index updates automatically whenever new evidence is generated. Because it is completely rebuildable from the existing evidence records, it contains no extraneous information. We intentionally designed the index to exclude inter-entity relationships, summaries, and relevance rankings. Relationships are treated as standalone claims that must exist as evidence, while synthesis and ranking are delegated to Hypothesis Generator since relevance is highly context-dependent. Additionally, the index tracks deprecated evidence separately and returns only active evidence by default.

B.1.3. Context Lifecycle

Initial construction The context lifecycle begins with initial construction, which triggers as soon as a user submits a query. As detailed in the B.1 process section, this phase involves breaking down the query, retrieving source-appropriate data, extracting evidence, building the entity index, and performing iterative coverage checks.

Cycle-time accumulation As the pipeline executes, cycle-time accumulation begins. During this phase, the Knowledge Consolidator submits experimental results containing both a human-readable report and a machine-queryable summary. The Context Optimizer then deposits this pair into the in-house database as raw data and extracts the relevant evidence into the Internal branch. Because these pipeline-generated claims share the same format as external branch, Hypothesis

Generator can retrieve Internal branch using the exact same API it uses for published findings. If new evidence contradicts an existing record, the system creates a supersession edge. It appends the new evidence and updates the older record's status to deprecated.

To maintain proper provenance tracking, two additional write operations occur at the cycle boundaries. First, at the end of its cycle, Hypothesis Generator saves its complete State Memory snapshot, encompassing both the global state and the node records. The system stores this snapshot as an immutable entry within the StateMemoryArchive branch. Second, when Context Optimizer extracts new evidence from experimental summaries, it stamps each record in the Internal branch with two fields pulled directly from the summary: an `originating_hypothesis_node_id`, which establishes a clear, metadata-level link that connects the newly generated empirical evidence back to the specific hypothesis node that originally prompted the experiment, and a verdict, which preserves the Wet-Lab Analyzer(B.6) assigned outcome for that hypothesis.

Human guide actions Human guides can intervene at any point during the context's lifecycle using four specific actions:

- **Remove.** When an evidence record is deemed invalid, the system marks it as deprecated and logs the reason for removal as new Meta evidence.
- **Modify.** If a record requires correction, the system deprecates the original and appends the corrected version, linking the two through a supersession relation.
- **Focus directive.** Guides can instruct Context Optimizer to prioritize a specific retrieval direction. This directive is saved in the Meta branch, allowing Context Optimizer to use it for future retrieval and helping Hypothesis Generator understand the human's intended focus.
- **Context reset.** If necessary, guides can discard and rebuild the entire context from scratch. The system archives the existing tree rather than deleting it, records the reset reason in the Meta branch, and leaves the raw data store unaffected.

B.1.4. Context Retrieval API

Context Optimizer exposes seven stateless APIs: four for retrieval across the three evidence branches, two for archive operations targeting the StateMemoryArchive branch (one write, one read), and one dedicated to the Knowledge Consolidator for submitting experimental episodes.

`get_evidence` As the most frequently utilized API, this function accepts the widest range of parameters and combines them strictly using AND conditions. The `entities` parameter takes a list of canonical IDs, while `entities_mode` dictates the matching logic. Setting this mode to "all" requires every listed entity to be present, making it ideal for exact verification. Conversely, setting it to "any" requires only a single match to support broader exploration. Users can apply additional filters such as branch paths, raw data IDs, deprecation status, hypothesis node IDs, and exclusion IDs. Crucially, the system sorts results using only neutral criteria like extraction time or cycle stage, completely avoiding relevance-based ranking. By default, the API returns only active records.

`get_raw_data` This function returns the complete raw data item unless a specific span is provided. When a span is specified, it retrieves only that exact segment. This targeted approach allows the State Estimator to inspect a particular passage from a paper or a specific database field without having to load the entire artifact.

`cooccurring_entities` This function identifies entities that appear alongside a target entity within the same evidence records. The sorting direction plays a key strategic role here. Sorting in descending order reveals frequent co-occurrences to highlight mainstream associations. In contrast, sorting in ascending order surfaces rare pairings that often serve as novel candidates for combination.

`archive_state_memory` This function handles persistence payloads. Its metadata field captures cycle-end timestamps along with any operator annotations logged at the cycle boundary. The operation is strictly atomic at the snapshot level, meaning a snapshot must succeed or fail in its entirety. The system does not support the partial archival of selected nodes.

`get_archived_state_memory` This function retrieves archived node records, requested trace fields, and the resolved ancestor chain directly from Context Optimizer's StateMemoryArchive branch. Callers provide specific hypothesis node IDs and use optional parameters to define the exact trace limits and chain depth. Since these snapshots are permanently immutable, repeated queries for the same node always guarantee identical results.

Table 1. API specifications for the Context Optimizer module.

API_name	Purpose	Input	Output
search_entities	Resolve natural language queries to canonical IDs.	query, type	canonical_id, surface_form, evidence_count
get_evidence	Retrieve evidence records based on filter conditions.	entities, entities_mode, branch_path, hypothesis_node_ids, order	List of evidence records
get_raw_data	Access original source materials or specific segments.	raw_data_id, span	Raw data item or specified segment
cooccurring_entities	Surface entity co-occurrence patterns.	entity_id, order, limit	Co-occurring entities with counts
archive_state_memory	Persist an immutable State Memory snapshot at cycle end.	tree_id, cycle_id, global_state, node_records, archive_meta	Write status confirmation
get_archived_state_memory	Retrieve historical node records from archived State Memory snapshots.	hypothesis_node_ids, fields, depth	Node records with requested trace fields and resolved ancestor chain.
make_experiment_record_evidence	Submit experimental episodes to trigger evidence extraction.	report, summary, episode_meta	None

make_experiment_record_evidence The Knowledge Consolidator uses this API as its sole entry point to submit experimental episodes to Context Optimizer. It accepts a paired report, a summary, and episode metadata to trigger evidence extraction directly into the Internal branch. The function returns no value. Section B.1.7 provides full processing details.

B.1.5. Query spec translation

The cognitive modules within Hypothesis Generator (the Explorer, State Estimator, and Router) never call these APIs directly. Instead, they draft query specifications that outline their retrieval intent, detailing target entities, desired scope (wide or narrow), and tilt (rare or mainstream). The Action module then translates these high-level specs into concrete API calls. For example, a wide scope translates into a cooccurring_entities call with a large limit, followed by get_evidence set to "any" mode. A narrow scope simply maps to get_evidence using the "all" mode over a defined set of entities. To handle tilt, a rare focus triggers an ascending sort to prioritize infrequent connections, whereas a mainstream focus triggers a descending sort. For exact verification, the system uses the "all" mode across the full combination of entities in a claim. Throughout this process, Context Optimizer remains entirely oblivious to these strategic concepts. It simply returns the evidence that matches the applied filters. The cognitive modules bear full responsibility for the search strategy, while the Action module merely executes the resulting mechanical steps. The same applies to get_archived_state_memory: when Explorer or State Estimator requires additional ancestor context from a prior cycle, they specify the relevant node IDs and depth in a query spec, and the Action module executes the call.

B.1.6. StateMemoryArchive access patterns

The archive branch supports one write API and one read API. At the end of each cycle, Hypothesis Generator persists its complete State Memory snapshot using the archive_state_memory API. When the next cycle initiates a

warm-start, the system reads ancestor reasoning paths directly from State Memory. Because graduated leaves and their associated metadata remain in memory across cycle boundaries, there is no need to query Context Optimizer to reconstruct these paths. If the Explorer or State Estimator requires additional ancestor context, it may optionally recover it via `get_archived_state_memory`. If the system needs to retrieve experiment-derived evidence from previous cycles, it uses `get_evidence` filtered by specific hypothesis node IDs. This filter matches the originating IDs stamped onto each Internal-branch record during the initial extraction phase.

B.1.7. Experimental results Ingestion

The Knowledge Consolidator submits experimental episodes through a single external API named `make_experiment_record_evidence`. This function accepts the report, the summary, and the episode metadata. Context Optimizer then deposits the report and summary into the in-house database as paired raw data, linking them under a shared record ID. From there, it extracts evidence records from the summary and places them into the Internal branch. During extraction, the system stamps each new record with two crucial pieces of metadata derived directly from the summary. The first is the `originating_hypothesis_node_id`, which explicitly links the new evidence back to the hypothesis that inspired the experiment. The second is the `verdict`, which records whether the Analyzer determined the outcome to support, refute, or remain inconclusive regarding the original hypothesis. This API operation is unidirectional and returns no value to the Knowledge Consolidator. Internally, this process relies on a hidden helper function called `deposit_experiment_record`. This function manages the actual writing of the paired artifacts to the database and returns the shared record ID necessary for the subsequent extraction step. Keeping this helper strictly internal ensures that Context Optimizer remains the sole authority over database writes and evidence-to-node linking.

B.2. Hypothesis Generator

The Hypothesis Generator performs hypothesis generation as a Tree-of-Thoughts search over a hypothesis tree maintained in State Memory. Hypothesis Generator is designed to produce hypotheses that are structurally diverse yet evidentially grounded: diversity is induced along two axes (evidence subset and reasoning mode), while each hypothesis is evaluated at both node and path levels to maintain auditability of the search process.

Hypothesis Generator consists of three cognitive modules — **Explorer** (generation and expansion), **State Estimator** (evaluation and reasoning diagnosis), and **Router** (search control) — together with an **Action module**, a shared execution layer that handles Context-Optimizer-facing retrieval and outcome-access calls for the Explorer and State Estimator, and **State Memory**, a shared data structure that stores the hypothesis tree (see Fig. 4).

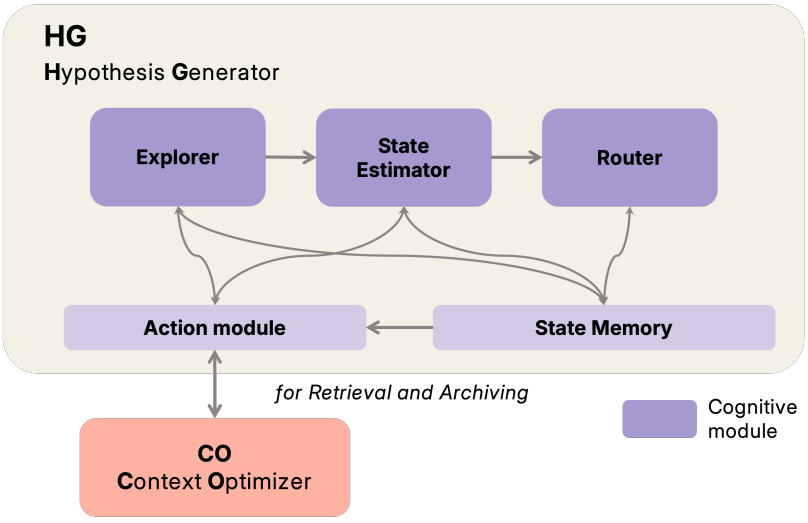


Figure 4. Architecture of the Hypothesis Generator, showing the interaction among Explorer, State Estimator, and Router, with State Memory as a shared internal component and the Action module interfacing with the Context Optimizer for retrieval and archiving.

B.2.1. HYPOTHESIS GENERATOR EXECUTION PIPELINE

Hypothesis Generator operates as an iterative Tree-of-Thoughts search over a hypothesis tree stored in State Memory, proceeding in cycles that begin from either a cold start or a warm start.

In cold start cycles, starting from an empty hypothesis tree, the Explorer rewrites the human query into a seed hypothesis, which defines the global objective of the search. This objective is stored in State Memory and is not subject to State Estimator or Router decisions. The Explorer then generates child hypotheses guided by this objective, which are evaluated by the State Estimator, and the Router assigns actions (expand, prune, graduate, or terminate) to guide the search.

In warm start cycles, the global objective remains fixed, while previously graduated nodes are reused as candidate nodes. These nodes are re-evaluated by the State Estimator using newly available evidence, after which the Router determines their actions and selects expansion targets.

The search proceeds iteratively through interactions among the Explorer, State Estimator, and Router. Candidate nodes are evaluated and acted upon in parallel across the frontier. A cycle terminates when no expandable nodes remain, at which point the current search tree is complete. Upon cycle termination, the State Memory snapshot is archived to Context Optimizer, and the graduated nodes are retained as candidates for the next cycle. Experimental validation is then performed on the retained graduated nodes. At the query level, the process continues across cycles as long as retained graduated nodes remain; otherwise, the search terminates. At each cycle boundary, a human operator may optionally intervene to select which hypotheses to advance for experimental validation and to decide whether to continue or terminate the search.

B.2.2. EXPLORER (IN COGNITIVE MODULE)

Role. The Explorer is the reasoning module responsible for generating hypotheses that populate the Tree-of-Thoughts. It writes retrieval specifications, partitions returned evidence into diverse subsets, assigns distinct reasoning modes to each subset, and produces the resulting hypotheses as nodes in the tree. During cold-start initialization, the Explorer creates a seed node from the rewritten user query and records it in State Memory as an initialization anchor that defines the starting point of the search. The Explorer then generates k child hypothesis nodes from this seed node as the first candidate hypotheses to be evaluated by the State Estimator. On each subsequent expansion step, it generates k child nodes from the current expansion target.

Trigger. The Explorer is invoked in two situations:

- **Cold start initialization:** when a new tree is generated for a query that has no prior exploration history. Activated at seed node creation.
- **Expansion step:** after the Router assigns an expand decision to a node, the Explorer is invoked on that node as the current expansion target.

Input. The input depends on the invocation mode:

- **Cold start** (tree initialization, external input): human query
- **Expansion step** (mid-tree, internal): current node identity and ancestor reasoning path, read from State Memory.

Process. The Explorer proceeds through sub-steps depending on the invocation mode. Sub-0 runs only during cold-start initialization, while Sub-1 through Sub-5 run during both initial expansion from the seed node and ordinary expansion from a Router-selected expansion target.

- **Sub-0. Seed Hypothesis Initialization.** In cold-start mode, the Explorer rewrites the human query into a seed hypothesis and instantiates a seed node containing the rewritten hypothesis. No evidence retrieval is performed at this stage, deferring evidence construction to subsequent expansion steps. This seed node serves as an initialization anchor for the first cycle but is not maintained as the structural root of the hypothesis tree. Instead, the human query and the seed hypothesis are retained as a global objective in State Memory and made accessible to all nodes. The Explorer then generates k child nodes from this seed node as the first candidate hypotheses.
- **Sub-1. Query spec authoring.** The Explorer reads the current node context and available reasoning path from State Memory, and authors a query specification for expansion. Scope parameters are set dynamically: nodes requiring

creative exploration receive wide radius with rare-entity tilt; nodes requiring feasibility refinement receive narrow radius with mainstream tilt.

- **Sub-2. Action module call and evidence pool retrieval.** The query spec is passed to the Action module, which returns an evidence pool. The pool is a set of evidence records accompanied by summary statistics.
- **Sub-3. Evidence subset partitioning.** The returned pool is partitioned into k distinct subsets through a hybrid of deterministic clustering and LLM-driven pattern identification:
  - **Deterministic partitioning:** pool entities are clustered by frequency and semantic embeddings, without requiring LLM invocation.
  - **LLM-driven partitioning:** the LLM reads the evidence (or a summarized view of the pool) and identifies semantically meaningful groupings, ideally capturing patterns that deterministic partitioning misses.
- **Sub-4. Reasoning mode assignment.** For each of the k subsets, the Explorer assigns a reasoning mode to construct the corresponding prompt. Reasoning modes include causal (cause-effect chains), analogical (structural similarity across domains), mechanism proposal (proposing missing links), and gap-filling (bridging gaps between evidence pieces).
- **Sub-5. Child hypothesis generation.** The k prompts are issued to the LLM in parallel, producing k child hypotheses. Each child consists of a claim, the supporting evidence subset with provenance, and creation metadata, and is recorded as a new node in State Memory.

Output.

- **Cold start:** One seed node and k child hypothesis nodes. The seed node contains the rewritten hypothesis derived from the user query, and serves as the starting point for the search. The k child nodes generated from this seed node are the first candidate hypotheses evaluated by the State Estimator.
- **Expansion step:** k child hypothesis nodes, each containing a natural-language claim, the supporting evidence subset with provenance, and metadata (parent pointer, depth, subset strategy, reasoning mode, and scope parameters applied).

All output is written to State Memory.

B.2.3. State Estimator (in Cognitive module)

Role. The State Estimator is Hypothesis Generator's evaluation module. It evaluates each child hypothesis node through a grounded two-level assessment:

- **Node-level estimation**, which evaluates the intrinsic quality and search status of the hypothesis
- **Path-level attribution**, which identifies how the reasoning path that produced the node may have succeeded or failed

The State Estimator does not select the next search action. It produces structured evaluation signals that are passed to the Router for search control.

Trigger. The State Estimator is invoked in two situations:

- **Expansion evaluation**: after the Explorer generates child hypothesis nodes, the State Estimator evaluates each node.
- **Warm start evaluation**: at the beginning of each warm-start cycle, the State Estimator re-evaluates retained nodes from the prior cycle using newly available evidence.

Input. For each node under evaluation, the State Estimator receives:

- the child hypothesis node and its natural-language claim
- the supporting evidence subset used during generation
- the reasoning mode and query scope parameters used by the Explorer
- the parent pointer and ancestor reasoning path retrieved from State Memory

Process. The State Estimator proceeds through four sub-steps. These sub-steps produce two parallel streams of signals: **node-level state scores** and **path-level attribution signals**.

- **Sub-1. Verification evidence retrieval.** For each child hypothesis, the State Estimator formulates a verification-oriented query specification. Unlike the Explorer's expansion query, this query is narrower and is designed to test the current claim. The query is passed to the Action module, which returns a verification evidence pool from Context Optimizer. Each evidence item is associated with a stance toward the current claim: support, refute, or inconclusive. For literature or database-derived evidence, this stance is assigned by the State Estimator. For experiment-derived evidence, the State Estimator does not re-adjudicate the experimental verdict; it imports the Analyzer-assigned verdict stored in Context Optimizer and integrates it with literature/database evidence at the claim and semantic-unit levels. Each evidence item retains a provenance pointer to the raw-data id and span, and the stance annotations are aggregated into support, refute, and inconclusive ratios.
- **Sub-2. Fine-grained grounding analysis.** The hypothesis is decomposed into semantic units, and each unit is aligned against the retrieved evidence. The estimator assigns each unit a grounding status and records which evidence items support or refute it. This step produces a grounding map that identifies:
  - supported, weakly supported, and not_supported parts of the hypothesis
  - over-claims or unsupported semantic units
  - potential reasoning gaps introduced during generation

  The grounding map is used both for node-level scoring and for path-level failure localization.
- **Sub-3. Two-level evaluation.** Using the verification evidence and grounding map, the State Estimator computes two categories of signals.

  First, it computes **node-level state scores**:
  - **Validity**: whether the hypothesis is contradictory, nonsensical, or implausible given the available evidence
  - **Expand Gain**: whether further expansion is likely to improve the hypothesis by resolving ambiguity or refining an incomplete but promising claim
  - **Experiment Readiness**: whether the hypothesis is sufficiently specific, testable, and operationalizable for experimental validation

  Second, it computes **path-level attribution signals**:
  - **Evidence Adequacy**: whether the selected evidence subset was sufficient and appropriate for supporting the claim
  - **Reasoning Mode Fit**: whether the chosen reasoning mode was aligned with the structure and quality of the evidence
  - **Scope Fitness**: whether the query scope was appropriately calibrated
  - **Claim Support Strength**: how strongly the core semantic units are grounded in the retrieved evidence
  - **Distortion Risk**: whether the transition introduced unsupported inference, over-specification, or semantic distortion

  The estimator also assigns explicit failure types, such as evidence selection error, reasoning mode mismatch, scope error, or over-claim, and outputs a localization confidence score as a reliability proxy for the attribution(see Fig. 5).
- **Sub-4. Terminal signal estimation.** Finally, the State Estimator estimates whether the node should remain on the active search frontier or be treated as a terminal candidate. A node receives a terminal-state signal when it is sufficiently valid, has low expected gain from further expansion, is experiment-ready, and has no major unresolved reasoning gaps. This is an evaluation signal, not a search-control decision. The Router consumes this signal together with the other node-level and path-level outputs to decide whether to expand, prune, graduate, or terminate the branch. The terminal signal is recorded in `state_scores` alongside the other node-level scores.

Output. For each child node, the State Estimator returns:

- **Verification and grounding results**

  - verification query specification (in verification_trace)
  - evidence references (in verification_trace)
  - support / refute / inconclusive ratios (in verification_summary)
  - semantic-unit grounding map (in grounding trace)
  - over-claim indicators and reasoning-gap units (in grounding_trace)
- **Node-level state scores (in state_scores)**
  - validity
  - expand gain
  - experiment readiness
- **Path-level attribution signals (in attribution)**
  - evidence adequacy
  - reasoning mode fit
  - scope fitness
  - claim support strength
  - distortion risk
  - failure types
  - localization confidence
- **Terminal-state signal (in state_scores)**
  - whether the node is likely to remain useful for further search or should be treated as a terminal candidate

All outputs are stored under evaluation_trace in the node schema (see Fig. 5).

Rationale. Generating diverse hypotheses is insufficient if their evidential grounding cannot be inspected and their reasoning failures cannot be localized. The State Estimator addresses this by combining fine-grained grounding analysis with two-level evaluation: node-level evaluation estimates the current quality and search status of a hypothesis, while path-level attribution explains how that state arose from evidence selection, query scope, reasoning mode, and the parent-to-child transition. These interpretable signals provide the Router with a principled basis for search control.

B.2.4. ROUTER (IN COGNITIVE MODULE)

Role. The Router converts the State Estimator's outputs into explicit search-control actions. Given the current node's evaluation results together with branch and frontier context, it decides whether to expand, graduate, prune, or terminate the current branch. Here, `graduate` closes the branch for the current cycle while leaving the hypothesis epistemically unresolved and awaiting experimental verification, whereas `terminate` closes the branch because the hypothesis is sufficiently resolved for the current search objective. The Router also triggers tree archival once no open frontier remains.

Trigger. The Router is invoked immediately after the State Estimator evaluates the current node.

Input.

- current-node evaluation outputs
  - validity, expand gain, experiment readiness
  - grounding_trace
  - path-level attribution signals
  - terminal_signal
- branch history
- frontier state (derived from State Memory, indicating whether any nodes remain expandable)

Process. The Router performs control in two layers.

- **Sub-1. Node-level action selection.** For the currently evaluated node, the Router selects one of the following actions, each corresponding to a distinct epistemic status of the node:
  - **expand:** the hypothesis is promising but underspecified. Return the node to the Explorer for further refinement.
  - **graduate:** the hypothesis is specific, grounded, and experimentally actionable, but its truth cannot be settled from currently available evidence alone. The node is handed to the Experiment Designer (B.3) for verification. `Graduate` closes the current branch for the present cycle, but the hypothesis remains epistemically unresolved and is retained as a warm-start anchor for the next cycle.
  - **prune:** the hypothesis is invalid, contradictory, or unsupported.
  - **terminate:** the hypothesis is sufficiently resolved for the current search objective, either because additional expansion is unnecessary given convergent evidence, or because a completed experimental episode provides a definitive verdict. `Terminate` closes the branch without retaining the node as a warm-start anchor.

  The action is chosen according to a fixed priority: **(1) Terminal Check**, **(2) Prune Check**, **(3) Graduate Check**, and **(4) Expand Check**. This priority prevents invalid, resolved, or experiment-ready nodes from being repeatedly expanded.
- **Sub-2. Tree-level completion control.** After assigning node-level actions, the Router checks whether any open frontier remains. If no node is eligible for further expansion, then every leaf must be in one of the states: `graduate`, `prune`, or `terminate`. The Router then triggers the tree-level action:
  - **Archive:** trigger State Memory's cycle-end lifecycle when no expandable frontier remains. The Router does not directly persist the tree to Context Optimizer; State Memory constructs the persistence payload, invokes the Action module to write the complete snapshot to Context Optimizer's StateMemoryArchive branch via `archive_state_memory`, and compacts active memory. Among the closed leaves, only graduated leaves are retained as root-level warm-start anchors for the next cycle.

  Here, `terminate` closes a single branch, whereas `archive` closes the entire tree.

Output.

- Node-level action : expand, graduate, prune, or terminate.
- Tree-level action : archive, if no open frontier remains, meaning that all leaf nodes have been assigned a closed action: graduate, prune, or terminate.

B.2.5. Action module

Role. The Action module is the shared execution layer that handles Context Optimizer-facing retrieval and outcome-access calls for the Explorer and State Estimator. It translates query specifications authored by these modules into Context Optimizer API calls, performs the calls, and returns structured evidence or outcome records. The Action module carries no interpretive authority. Decisions about what to retrieve, how to judge retrieved evidence, and how to modify hypotheses remain with the calling Explorer and State Estimator. This separation keeps these modules free from retrieval-protocol concerns and ensures that all cognitive-to-Context Optimizer traffic flows through a single, auditable channel.

Trigger. The Action module is not a fixed pipeline stage. It is invoked on demand when the Explorer or State Estimator requires external context from Context Optimizer:

- **Explorer expansion (Sub-2):** during hypothesis generation, when the Explorer needs an evidence pool matching its expansion query specification.
- **State Estimator evaluation (Sub-2):** during hypothesis evaluation or warm-start initialization, when the State Estimator needs to retrieve evidence or experiment outcomes from Context Optimizer.
- **State Memory archive persistence :** during cycle-end lifecycle, when State Memory needs to persist the complete State Memory snapshot into Context Optimizer's StateMemoryArchive branch via `archive_state_memory`.
- **StateMemoryArchive read (Explorer / State Estimator):** when either module requires additional ancestor context from a prior cycle, the Action module calls `get_archived_state_memory` with the relevant node IDs and depth.

Invocations are independent; the Action module holds no persistent state between calls.

Input. The input is either a query specification or a persistence payload.

- **Retrieval and outcome-access call.** The input is a query specification authored by the calling Explorer or State Estimator, containing:
  - entity set
  - scope parameters
  - expected pool size
  - retrieval purpose tag
- **Archive-persistence call.** The input is a persistence payload constructed by State Memory, containing:
  - tree id and cycle id
  - global state and node records to be persisted
  - archive metadata
  - provenance and parent/child relations required for later StateMemoryArchive retrieval
- **Archive-read call.** The input is an archive-query specification containing:
  - hypothesis_node_ids
  - requested fields
  - depth

Process.

- **Sub-1. Translation.** For archive-read calls, the node ID and depth specification are translated into Context Optimizer read calls to the StateMemoryArchive branch.
- **Sub-2. Execution.** the translated call is issued to Context Optimizer. Retrieval calls return evidence or outcome records. Archive-persistence calls write snapshot records to Context Optimizer, typically via `archive_state_memory`. Archive-read calls return archived node records from Context Optimizer via `get_archived_state_memory`. If the request fails or is rate-limited, the Action module handles retries transparently to the caller.
- **Sub-3. Parsing.** For retrieval calls, the raw Context Optimizer response is parsed into structured evidence records, each carrying content, entity tags, and provenance. For archive-persistence calls, the Action module parses write acknowledgments and error metadata. For archive-read calls, the response is parsed into structured node records with the requested trace fields and resolved ancestor chains.
- **Sub-4. Post-processing.** For retrieval calls, the Action module performs lightweight operations that do not involve interpretation: deduplication across paginated responses, merging multi-page pools into a single evidence set, and basic sanity filtering. For archive-persistence calls, it reports the write status to State Memory. For archive-read calls, it merges paginated node records and filters to the requested fields and depth. It does not decide which nodes to retain, does not compact State Memory, and applies no content-level judgment.

Output.

- **Retrieval and outcome-access call.** The output is a result set returned to the calling Explorer or State Estimator, consisting of structured records with provenance and pool-level summary statistics.
- **Archive-persistence call.** The output is a write-status record returned to State Memory, indicating successful writes, failed writes, retry metadata, and any Context Optimizer-side errors.
- **Archive-read call.** The output is a set of archived node records, requested trace fields, and resolved ancestor chains returned to the calling Explorer or State Estimator.

Node Schema

identifier: Unique node identity and tree-relation metadata, including hypothesis_node_id, node_type, parent_pointer, children, depth, and cycle_created.
creation_trace: Explorer-written metadata.
- **generation_stage:** cold_start_seed for the seed node; expansion_child for ordinary hypothesis nodes.
- **reasoning_mode:** reasoning mode used to generate the node.
- **hypothesis_claim:** natural-language hypothesis represented by the node.

expansion_trace: Explorer-written metadata.
- **expansion_query_spec:** query specification for further expansion.
- **retrieved_evidence_pool:** evidence references and pool-level summary statistics.
- **generated_child_ids:** child nodes produced from the expansion.

evaluation_trace: State Estimator outputs for the current cycle. It contains:
- **verification_trace:** verification_trace and verification_summary.
- **grounding_trace:** semantic-unit grounding map.
- **state_scores:** node-level scores such as validity, expand_gain, experiment_readiness, and terminal_signal.
- **attribution:** path-level attribution signals.

router_trace: Router-written search-control decision for the current cycle.

Figure 5. Node schema used in State Memory, including structural metadata and module-specific traces for creation, expansion, evaluation, and routing.

B.2.6. State Memory

Role. State Memory is the shared working memory that holds both the global search state and the hypothesis tree for the current cycle in Hypothesis Generator, including nodes in all states (see Fig. 6). It is read and written exclusively by the three cognitive modules above—Explorer, State Estimator, and Router—and no module outside Hypothesis Generator accesses it directly. The global state stores cycle-level information such as the seed hypothesis, original user query, current cycle ID, expandable-node status, and retained node IDs. These fields are dynamically updated throughout the search process and across cycles.

Node records are stored as a map from `hypothesis_node_id` to `Node` and contain creation, expansion, evaluation, and routing traces (see Fig. 5). At cycle end, the complete snapshot (global_state and node_records) is persisted into Context Optimizer's StateMemoryArchive branch (B.1.1) via `archive_state_memory`, while State Memory is compacted for the next cycle. Retained graduated leaves remain in memory for warm-start evaluation, and the global state preserves the query-level objective across cycles. As a result, State Memory achieves the practical continuity of a single persistent tree without carrying its full memory cost.

Trigger. State Memory is a passive data structure with no autonomous action. It is written by the Explorer for new nodes, by the State Estimator for evaluation signals, and by the Router for action decisions. It is read by all three cognitive modules during their operations. Cycle-end lifecycle behavior is triggered by the Router's archive action.

Input. The primary input units are node records (see Fig. 5) and the global state. The global state is updated by the Router, including expandable-node status and retained node IDs, while the global objective is maintained across cycles.

Each cognitive module writes to specific fields:

- **Explorer:** writes node identity, tree-relation fields, and `creation_trace` at node creation, and writes `expansion_trace` when a node is refined.

State memory Schema

global_state:

- **seed_hypothesis:** Rewritten hypothesis defining the search goal
- **original_user_query:** The original natural language query provided by the user
- **tree_id:** Identifier of the query-level hypothesis tree across cycles
- **cycle_id:** Identifier of the current search cycle within the tree
- **has_expandable_nodes:** Boolean flag indicating whether any nodes remain expandable

retained_node_ids: List of node ids preserved after archive for warm-start initialization

- **node_records: Map [hypothesis_node_id, Node]**
- **Collection of Node objects** (See Node Schema for details)

Figure 6. State Memory schema, including the global search state and the collection of node records, along with their lifecycle across cycles.

- **State Estimator:** writes `verification_trace`, semantic-unit grounding records, node-level state scores, and path-level attribution signals after evaluation.
- **Router:** writes decision fields after action selection.

Process. At Router's archive trigger, State Memory executes its cycle-end lifecycle. It constructs a persistence payload for the complete in-memory tree and requests the Action module to write the snapshot to Context Optimizer's StateMemoryArchive branch via `archive_state_memory`. The archived tree is treated as an immutable archive entry. After successful persistence, State Memory is compacted: all non-retained node records are cleared from active memory, while retained graduated leaves and the global state are preserved for the next cycle. The retained graduated leaves serve as warm-start anchors for the next cycle. Their ancestor paths are not kept in active memory, but remain recoverable from Context Optimizer's StateMemoryArchive through the Action module via `get_archived_state_memory` when Explorer or State Estimator requires additional ancestor context. Newly accumulated internal evidence is retrieved at the context level, either through entity-based filters in `get_evidence`, or by the `hypothesis_node_ids` filter when warm-starting a specific retained leaf. Therefore an experiment performed on one retained graduated leaf can update the evaluation context of other retained warm-start anchors. No LLM invocation occurs within these lifecycle operations.

Output. State Memory exposes module-specific read interfaces:

- **Explorer.** State Memory provides the current expansion target, its `seed_trace`, and the available ancestor reasoning path.
  - The ancestor reasoning path includes ancestor hypotheses, `creation_trace`, `expansion_trace`, prior scope parameters, and retrieval outcomes.
  - The `seed_trace` preserves the original user query and seed hypothesis, while the reasoning path provides local expansion context.
  - These records are used to construct the next `expansion_query_spec` and generate child hypotheses.
- **State Estimator.** State Memory provides the current node's `hypothesis_claim`, `seed_trace`, and `creation_trace`.
  - It also provides the parent and ancestor reasoning path and their previous `evaluation_trace` records when available.
  - These records are used to perform verification, semantic-unit grounding, node-level scoring, and path-level attribution.
  - The resulting `verification_trace`, `grounding_trace`, `state_scores`, and `attribution` are written into the current node's `evaluation_trace`.
- **Router.** State Memory provides the current node's `evaluation_trace`, branch history, and frontier state.

- Branch history is derived from prior `evaluation_trace` and `router_trace` records and may include score trends, low-scoring segments, unresolved reasoning gaps, and improvement trajectories.
- The frontier state indicates whether any nodes remain expandable.
- These signals are used to select `node_action`, which is written into the current node's `router_trace`.
- When no expandable nodes remain, the cycle-end archive lifecycle of State Memory is triggered.

B.3. Experiment Designer

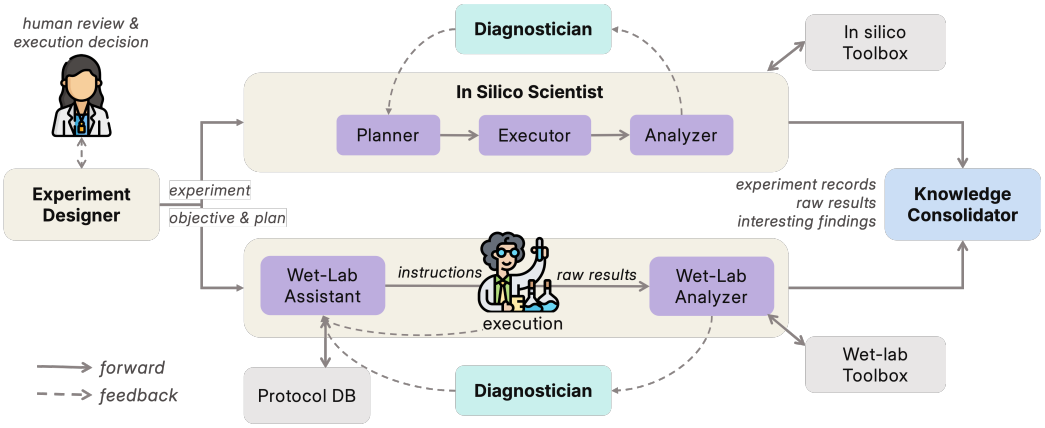


Figure 7. Experimental pipeline, composed of Experimental Designer (B.3), In Silico Scientist (B.4), Wet-Lab Assistant and Analyzer (B.4, B.5). Record of each experimental episode written by the Knowledge Consolidator (B.7) is deposited into the in-house database via the Context Optimizer (B.1.4). Diagnostician (C.1) is a plug-in module that assesses observation trustworthiness, either reactively when acceptance criteria fail or proactively when a quality scan is requested.

Role. The Experiment Designer constructs a rigorous experimental strategy for verifying each graduated hypothesis received from the Hypothesis Generator, and assigns it to the In Silico Scientist (B.4) or the Wet-Lab Assistant(B.5). The Experiment Designer does not execute experiments; it designs them and **a human reviewer decides whether to execute or not**. Its primary responsibilities are maximizing the hypothesis's falsifiability, minimizing wasted experimental resources, and ensuring that the resulting evidence is interpretable and unambiguous.

Trigger. Whenever the Hypothesis Generator graduates a hypothesis during its tree-of-thought search, the graduated hypothesis is forwarded to the Experiment Designer.

Position in the cycle. The Experiment Designer is invoked after the Hypothesis Generator completes its full tree-of-thought search, that is, when every leaf node has reached a *terminate*, *prune*, or *graduate* state and no further expansion is possible. At this point, the State Estimator ranks the graduate, and the Experiment Designer composes an experiment plan for each graduated hypothesis on the priority list. A human reviewer decides which experiments to execute in the current cycle by jointly evaluating three components: the hypothesis (from the Hypothesis Generator), its confidence and quality assessment (from the State Estimator), and the experimental plan with expected cost (from the Experiment Designer). The reviewer may also attach optional directives to selected experiments, such as requesting a proactive quality scan of the results by the Diagnostician. The selected experiments are forwarded to the In Silico Scientist or the Wet-Lab Assistant for execution, and their results are passed to the corresponding Analyzer.

Process. Based on the hypothesis, it defines the objectives to verify hypothesis. After defining objectives including what to observe, which types of tools to use, and how to control variables, it assigns the experimental task to In Silico Scientist or Wet-Lab Assistant according to the following criteria.

In Silico Scientist is selected when:

- The hypothesis is computationally verifiable (structure prediction, affinity scoring, distogram analysis)
- Large-scale screening is required before committing to wet-lab resources. The screening results will be passed to Hypothesis Generator through Context Optimizer, then Experiment Designer.

Wet-Lab Assistant is selected when:

- The hypothesis requires direct experimental measurement (e.g., target-ligand binding signals, gene expression level, gene knockout response)
- A prior in silico result requires orthogonal experimental validation

Acceptance and verdict criteria. As part of the experiment plan, the Experiment Designer pre-specifies two distinct sets of criteria, both fixed before execution and forwarded to the assigned module.

- **Acceptance criteria** gate whether the resulting observation is trustworthy enough to interpret. They specify conditions that must hold for the data to be considered valid, independent of what the data says about the hypothesis. For wet-lab experiments, these include positive and negative control thresholds and replicate-variance limits. For in silico experiments, these include minimum model confidence (e.g., pLDDT thresholds for structural claims), required sanity checks (e.g., recapitulation of a known reference structure within a specified RMSD), and replicate-variance limits across stochastic runs. If any acceptance criterion fails, the Diagnostician (C.1) is invoked in reactive mode before any verdict is assigned.
- **Verdict criteria** specify, given trustworthy data, the conditions under which the results are interpreted as support, refute, or inconclusive. Inconclusive criteria include cases where the effect size falls between the support and refutation thresholds, or where off-target confounds cannot be ruled out by the data alone. Verdict assignment is performed by the assigned module's Analyzer only after acceptance criteria are satisfied.

Output. The Experiment Designer returns three outputs:

- **Module assignment**: whether the In Silico Scientist or the Wet-Lab Assistant will conduct the experiment.
- **Experiment plan**: the logical structure of the experiment to be executed by the assigned module, carrying the originating `hypothesis_node_id` for traceability back to StateMemoryArchive, and any optional directives attached by the human reviewer (such as a request for the Diagnostician's proactive quality scan).
- **Verdict criteria**: the pre-specified conditions under which results are interpreted as support, refute, or inconclusive.

The Experiment Designer does not specify which software or protocols to use; these are determined by the assigned module according to its own expertise. The Experiment Designer is solely responsible for designing the logical structure of the experiment.

B.4. In Silico Scientist

Role. The In Silico Scientist is responsible for executing computational experiments assigned by the Experiment Designer. Given a logical experiment plan, it independently determines the appropriate software, tools, and pipelines, runs the experiments, and interprets the results in the context of the research objective.

Trigger. The In Silico Scientist activates upon receiving an experiment plan from the Experiment Designer. The plan specifies the objectives of the experiment (e.g., the hypothesis, the comparison design, the metrics to be measured) but leaves all implementation details to the In Silico Scientist.

Process. The In Silico Scientist consists of three modules:

- **Planner** translates the experiment plan into a concrete computational workflow. This includes selecting software and tools, preparing input data, defining the pipeline architecture, and specifying execution parameters.
- **Executor** runs the pipeline and resolves runtime-level issues internally, including transient I/O failures, missing dependencies, and recoverable execution errors. The internal loop continues until the pipeline produces a final output. Note that producing an output does not imply the output is trustworthy; trust assessment occurs in the next step.
- **Analyzer** checks the output against the acceptance criteria specified by the Experiment Designer (B.3).

  (1) *Acceptance fails* → The Diagnostician (C.1) is invoked in reactive mode. Its report is forwarded to the Planner, which decides whether to re-execute the same workflow or compose a revised one. No interpretation is generated for failed runs; each such run is retained in the experiment's accumulating history and forwarded to the Knowledge Consolidator (B.7) when the experiment concludes.

  (2) *Acceptance passes* → The Analyzer proceeds to interpret the results against the verdict criteria, applying any necessary statistical analyses and producing a structured summary with a verdict (support, refute, or inconclusive)

on the hypothesis. If the human guide has requested a proactive quality scan, the Diagnostician is then invoked between acceptance and verdict assignment, following the same handling described in B.6.

Tool Usage. The In Silico Scientist has access to a curated toolbox of bioinformatics and computational biology software, covering tasks such as structure prediction, sequence analysis, molecular dynamics, and statistical testing. When a required tool is not available in the current registry, the agent can invoke an external tool discovery and validation workflow, including search over repositories such as ToolUniverse and Biomni, as well as literature databases, to identify, evaluate, and register new tools on demand. The details of this discovery-validation-registration loop, along with the debugging infrastructure, are handled by dedicated external agents and are not specific to the In Silico Scientist; they apply uniformly across all computational modules in the system.

Output. The In Silico Scientist returns three outputs:

- **experimental record**, which documents the tools and their versions, input data, and pipeline used to produce the results
- **raw output files**, including their storage locations for reproducibility and downstream access
- **interpreted findings**, which summarize the results in the context of the hypothesis and the research objective, including any statistical analyses applied and a clear verdict on whether the results support, refute, or are inconclusive with respect to the hypothesis. This bundle, which carries the originating `hypothesis_node_id` alongside the records, is forwarded to the Knowledge Consolidator for final documentation.

B.5. Wet-Lab Assistant

Role. The Wet-Lab Assistant translates the experiment plan into concrete, reproducible experimental protocols for human or robotic executors. It acts as the interface between the AI scientist and physical experimentation, ensuring that every proposed experiment is grounded in standardized, version-controlled procedures.

Trigger. As with the In Silico Scientist, the Wet-Lab Assistant activates upon receiving an experiment plan from the Experiment Designer. Distinctively, the Wet-Lab Assistant may also be triggered mid-experiment by the Diagnostician, when a control failure or assay integrity issue requires protocol-level intervention rather than analytical troubleshooting.

Process

- **Protocol design**. The Wet-Lab Assistant translates the logical experiment plan into a fully specified experimental protocol, including reagents, concentrations, equipment settings, step-by-step procedures, and timing. It draws from a curated protocol database and selects or adapts the most appropriate procedure for the given assay type. All protocols are version-controlled and traceable to their source.
- **Protocol review and finalization**. The Wet-Lab Assistant presents the drafted protocol to the human executor for review. If modifications are needed, the human provides feedback to the Wet-Lab Assistant, which revises the protocol accordingly. This review cycle repeats until the protocol is approved, at which point it is finalized and dispatched for execution.
- **Resource check**. Before dispatching the protocol, the Wet-Lab Assistant verifies that all required reagents, cell lines, instruments, and consumables are available. If any resource is missing, it flags the gap and proposes alternatives or procurement steps before proceeding.

Output. The Wet-Lab Assistant returns the experimental protocol, which provides a complete, version-controlled, executor-ready procedure specifying all reagents, equipment, steps, and controls.

B.6. Wet-Lab Analyzer

Trigger. Activated when a human or robotic executor submits raw experimental data, accompanied by a pointer to the relevant experiment specification (protocol version, control definitions, expected output schema).

Process.

- **Data heterogeneity handling**. Wet-lab data spans a wide range of modalities: fluorescence microscopy images, flow cytometry FCS files, western blot densitometry values, ELISA plate reader outputs, sequencing FASTQ files, and free-text observational notes from the experimenter. For each modality, the Wet-Lab Analyzer maintains a curated pipeline registry (wet-lab tool box), a mapping from data type to a versioned analysis workflow. If an incoming data

type is not covered by the existing tool, the agent queries an external tool registry for candidate pipelines, downloads and validates them in a sandboxed environment, and registers the selected pipeline for future reuse. All analysis code is stored in the wet-lab toolbox with its run configuration, ensuring reproducibility.

- **Diagnostician interaction**. Before proceeding to biological interpretation, the Wet-Lab Analyzer checks whether the protocol's acceptance criteria are satisfied (B.3).

  (1) *Acceptance fails* → The Diagnostician (C.1) is invoked in reactive mode. Its report is forwarded to the Wet-Lab Assistant, which decides whether to re-execute the same protocol or compose a revised one. No biological interpretation is generated for failed runs; each such run is retained in the experiment's accumulating history and forwarded to the Knowledge Consolidator (B.7) when the experiment concludes.

  (2) *Acceptance passes* → The Analyzer proceeds to quantitative analysis. If the human guide has requested a proactive quality scan, the Diagnostician is then invoked to check for confounds not covered by the acceptance criteria (e.g., batch effects, covariate correlations, platform artifacts). If the scan raises a flag, interpretation is held and the report is forwarded to the human guide, who may direct re-execution or proceed with the flag recorded as a caveat in the Knowledge Consolidator report.

Output. The Wet-Lab Analyzer returns three outputs:

- **experimental record**, which documents the protocol version as actually executed, acceptance validation status, and the complete analysis code with versioned configuration used to process the results.
- **raw and processed data files**, including their storage locations for reproducibility and downstream access
- **interpreted findings**, which summarize the results in the context of the hypothesis and the research objective, including any statistical analyses applied and a verdict on whether the results support, refute, or are inconclusive with respect to the hypothesis, evaluated against the verdict criteria pre-specified by the Experiment Designer. This bundle, which carries the originating `hypothesis_node_id` alongside the records, is forwarded to the Knowledge Consolidator for final documentation.

B.7. Knowledge Consolidator

Role. The Knowledge Consolidator produces a structured experimental record after each experimental episode and deposits it into the in-house database through Context Optimizer's `make_experiment_record_evidence` API in two formats: a human-readable report for review and knowledge accumulation, and a machine-queryable summary that the Context Optimizer reads to extract evidence records into the context tree. Evidence record creation remains exclusively within the Context Optimizer, preserving single-source evidence creation across the pipeline.

Trigger. Activated upon a completed experimental episode.

Input.

(1) the hypothesis(hypothesis and `hypothesis_node_id`), propagated from Hypothesis Generator through the dispatch chain,

(2) the experiment plan and direction from the Experiment Designer,

(3) the output of the In Silico Scientist (for in silico experiments), or

(4) the experiment protocol from the Wet-Lab Assistant paired with the result from the Wet-Lab Analyzer (for wet-lab experiments).

(5) All experiment history, if there is a series of experiments that the Diagnostician modified during troubleshooting.

Knowledge Consolidator submits both formats to Context Optimizer via `make_experiment_record_evidence`. Context Optimizer deposits them as paired raw data, extracts evidence from the summary into the Internal branch, and stamps each new evidence record with the `originating_hypothesis_node_id` read from the summary. The call returns no value to Knowledge Consolidator (B.1.4).

Output. Each deposit consists of two formats stored together in the in-house database:

(1) **Human-readable report**. Full documentation of the experimental episode, intended for human review and as a permanent knowledge asset.(see Table 2).

(2) **Machine-queryable summary**. A compact structured representation containing only the fields needed for evidence extraction (see Table 3). Methodological detail (protocols, tool configurations) is excluded; the summary focuses on the hypothesis tested, the result, and its interpretation.

Table 2. Document structure of the human-readable experimental report.

Section	Source	Content
Background & Rationale	Hypothesis Generator	Justification for the experiment, including supporting evidence and citations.
Objective	Hypothesis Generator	The hypothesis to be tested during the cycle.
Methods (wet)	Experiment Designer & Wet-Lab Assistant	Experimental plan and verdict criteria (Designer); version-controlled protocol details (Assistant).
Results (wet)	Wet-Lab Analyzer	Interpreted findings with a verdict (support/refute/inconclusive), acceptance validation status via the Diagnostician, and links to raw/processed data files.
Methods (in silico)	Experiment Designer & In Silico Scientist	Experimental plan and verdict criteria (Designer); computational workflow details (Scientist).
Results (in silico)	In Silico Scientist	Interpreted findings with a verdict, acceptance-criteria validation status via the Diagnostician where applicable, model-level metadata, and output file location.

Table 3. Fields of the machine-queryable summary passed to the Context Optimizer.

Field	Source	Content
`hypothesis_node_id`	Hypothesis Generator	Pointer to the originating node.
`claim`	Hypothesis Generator	The hypothesis to be tested during the cycle.
`experiment_summary`	Experiment Designer	Experimental plan, acceptance criteria, and verdict criteria.
`results`	In Silico Scientist or Wet-Lab Analyzer	Interpreted findings.
`verdict`	In Silico Scientist or Wet-Lab Analyzer	Support / refute / inconclusive label.

C. Common Infrastructure

C.1. Diagnostician (plug-in)

Role. The Diagnostician is a shared plug-in module that assesses whether experimental observations are reliable enough to be used as evidence. It operates in two modes distinguished by trigger type, not by module: *reactive mode* performs root-cause analysis when a pre-specified acceptance criterion fails, and *proactive mode* screens for confounds that the acceptance criteria did not explicitly test for. Both modes apply to both wet-lab and in silico experiments. The Diagnostician does not issue scientific verdicts (support, refute, or inconclusive); that responsibility belongs to the assigned experimental module's Analyzer. Its sole concern is whether the data can be trusted before interpretation proceeds.

Invocation. The Diagnostician is not a pipeline stage. It is invoked by experimental modules when needed:

- **Reactive mode:** called when a pre-specified acceptance criterion fails. In wet-lab experiments, this is triggered by the Wet-Lab Analyzer when controls fail against the protocol's acceptance criteria (B.6). In in silico experiments, this is triggered by the In Silico Scientist's Analyzer when model confidence, sanity checks, or replicate-variance criteria specified by the Experiment Designer (B.3) fail (B.4).

- **Proactive mode:** called by the Wet-Lab Analyzer or the In Silico Scientist's Analyzer after quantitative analysis but before verdict assignment, when the human guide has requested it via the experiment plan (B.3).

Input. Both modes receive a diagnosis request from the calling module containing:

- **Observation:** the data that triggered the call. For reactive mode, the failing data and the full attempt history (current run plus all prior failed runs). For proactive mode, the quantitative analysis results.
- **Expectation:** what should have held. For reactive mode, the acceptance criteria from the experiment plan or protocol. For proactive mode, normal operating ranges derived from platform metadata or prior experiments.
- **Execution context:** protocol version (wet-lab) or pipeline configuration and model metadata (in silico), along with any relevant technical variables (batch, plate, lane, barcode, operator, reagent lot for wet-lab; input distribution, model version, hyperparameters, random seed for in silico).

Process. Reactive mode addresses explicit acceptance failures, while proactive mode screens for hidden confounds that may compromise results even when acceptance criteria pass.

- **Reactive mode.** The Diagnostician compares the failing observation against the acceptance criteria, examines the attempt history for recurrent patterns, and performs root-cause reasoning. Root causes are module-specific. For wet-lab failures: protocol deviations, target-specific idiosyncrasies (e.g., low-expressing proteins, unstable ligands), reagent lot inconsistencies, or equipment calibration drift. For in silico failures: input out-of-distribution conditions, model misapplication outside its valid domain, training-data domain mismatch, or configuration errors that produce finite-but-untrustworthy outputs. In both cases, the Diagnostician retrieves prior diagnostic reports from Context Optimizer's Internal branch via `get_evidence` to check whether the same failure pattern has been encountered before.
- **Proactive mode.** The Diagnostician screens for confounds through two layers. A generic confound check tests whether technical metadata variables in the execution context correlate with the biological signal rather than the intended experimental variable. A template-based check retrieves prior diagnostic reports from the Internal branch of Context Optimzer to see whether the current platform has known artifact patterns from prior cycles. For example, probe barcode identity in single-cell RNA sequencing has been shown to drive substantial spurious differential expression between biologically identical cells (Weir et al., 2026); if a prior cycle had diagnosed this artifact, the Diagnostician would flag the same pattern in subsequent experiments on the same platform. Without prior case studies or specific guidance from the human guide, the proactive mode is limited to the generic confound check, and novel platform artifacts may go undetected.

Output. Both modes produce a diagnostic report containing:

- **Anomaly class:** the type of problem identified (control failure, acceptance-criterion failure, batch effect, model bias, reagent inconsistency, training-data overlap, etc.). In proactive mode, this field may be `clean` if no confound is detected, in which case the Analyzer proceeds to verdict assignment without further intervention.
- **Diagnostic findings:** the analysis results supporting the diagnosis (e.g., correlation statistics between technical variables and biological signal, QC metric comparisons across batches, overlap with known artifact patterns from prior case studies, input-distribution distance from training data).
- **Candidate root causes:** a ranked list of candidate causes with references to the diagnostic findings and any matching prior case studies retrieved from Context Optimizer.

The report is forwarded to the appropriate decision-maker depending on mode and module. In reactive mode, the report goes to the module responsible for re-execution: the Wet-Lab Assistant (B.5) for wet-lab failures, which decides whether to re-execute or revise the protocol; or the In Silico Scientist's Planner (B.4) for in silico failures, which decides whether to re-execute or revise the workflow. In proactive mode, the report goes to the human guide, who decides whether to direct re-execution or proceed with the flag recorded as a caveat in the Knowledge Consolidator report. In all cases, the diagnostic report is eventually bundled into the Knowledge Consolidator's experimental record (B.7) and deposited into the Internal branch, where it becomes a retrievable failure case study for future cycles. Over time, these deposited reports expand the template library available to the proactive mode, enabling the Diagnostician to catch a progressively wider range of

Table 4. Entity types and their canonical resources.

Entity type	Resource	Identifier prefix	Scope
Gene	HGNC (Seal et al., 2023)	`HGNC:`	Human gene symbols and nomenclature
Protein	UniProt (The UniProt Consortium, 2025)	`UniProt:`	Protein sequences and identifiers across species
Disease	MONDO (Vasilevsky et al., 2022)	`MONDO:`	Unified disease ontology integrating OMIM, Orphanet, DOID, NCIt, ICD
Chemical	ChEMBL (Zdrazil et al., 2024)	`CHEMBL`	Bioactive small molecules and drug-like compounds
Drug	DrugBank (Knox et al., 2024)	`DB`	Approved and investigational drugs with clinical and pharmacological annotation
Anatomical structure	UBERON (Mungall et al., 2012)	`UBERON:`	Cross-species anatomy (organs, tissues)
Cell type	Cell Ontology (Bard et al., 2005)	`CL:`	Canonical cell types across organisms
Pathway	Reactome (Milacic et al., 2024)	`R-HSA-`	Manually curated reaction-level biological pathways

platform-specific artifacts.

C.2. Database

The pipeline relies on two distinct types of databases based on access scope and authority. First, the Context Optimizer (B.1) pulls information from external databases during context construction. Second, the Knowledge Consolidator (B.7) writes experimental results directly to a secure in-house database. We can further divide the external resources by their functional roles. Some act as reference identifiers that allow the Context Optimizer to normalize entity mentions into canonical IDs (B.1.2). The rest function as the primary data sources for extracting the actual evidence records.

C.2.1. Reference Identifiers

During the ontology-linking step, the system maps each NER-detected mention to a canonical ID from a recognized community registry. We use these resources strictly for identifier normalization, meaning the bulk of the actual evidence content comes from the separate data sources discussed in the next section. To align with standard biomedical informatics practices, we selected the most widely accepted resource for each entity type. Occasionally, multiple ontologies overlap for the same category, such as MONDO for diseases or Reactome for pathways. In these situations, we establish one resource as the primary standard to prevent evidence from fragmenting across duplicate identifiers. Any alternative ontologies simply remain accessible via cross-reference mappings provided by the primary resource.

When the system encounters a mention that it cannot resolve to a standard ID, it retains the entity with a null value for its canonical ID. These unresolved mentions still remain fully searchable using their original surface forms (B.1.2). Additionally, human guides can step in and manually register new canonical mappings if the automated linker struggles with highly specialized or novel domain terminology.

Table 5. External data sources and access methods.

Category	Resource	Data Type	Access
Literature	PubMed (Sayers et al., 2025)	Citations and abstracts of biomedical literature	E-utilities API
Clinical	ClinicalTrials.gov (Sayers et al., 2025)	Clinical trial protocols, outcomes, status	API
Protein	UniProt (The UniProt Consortium, 2025)	Protein sequence, function, domains, PTM, variants	REST API
Chemical	ChEMBL (Zdrazil et al., 2024)	Compound–target bioactivity measurements	REST API
Drug	DrugBank (Knox et al., 2024)	Approved/investigational drugs, mechanisms, interactions	API
Pathway	Reactome (Milacic et al., 2024)	Reaction-level curated pathways	REST API
Pathway	KEGG (Kanehisa et al., 2025)	Pathway maps, drug–disease links, metabolism	REST API
Protein Structure	RCSB PDB (Burley et al., 2025)	Experimental 3D structures	RCSB API
Protein Structure	AlphaFold DB (Varadi et al., 2024)	AI-predicted protein structures	EMBL-EBI API

C.2.2. External Database

The Context Optimizer pulls raw data from various public sources, categorizing them by the type of evidence they provide. While the specific content varies, the extraction process remains consistent across all sources. The system deposits raw items into a shared data store, extracts evidence records, and links entity mentions to canonical IDs (C.2.1). Text-heavy sources rely on LLMs to scan for artifacts, whereas structured databases use deterministic templates. To prevent redundant records, we designate a single primary source for any information that overlaps across multiple databases.

Literature PubMed serves as our primary source for biomedical literature, offering access to millions of citations and abstracts (Sayers et al., 2025). The system uses PubMed to execute natural-language queries based on the user's initial query decomposition. Because the data here consists of unstructured prose, an LLM must scan the retrieved abstracts and full-text articles to parse out factual and causal claims with verified span pointers.

Clinical For clinical research, the system relies on ClinicalTrials.gov (Sayers et al., 2025). Context Optimizer retrieves trial records relevant to the queried diseases and interventions, extracting evidence through deterministic templates. These records capture critical structured data, including trial identity, target indications, study phases, and outcome verdicts. This information is especially vital for hypotheses that evaluate therapeutic translatability.

Protein UniProt acts as the definitive reference for protein-level data (The UniProt Consortium, 2025). The system retrieves UniProt entries using resolved accession IDs and extracts functional annotations, localization data, and sequence variants via deterministic templates. If a UniProt entry lists pathway memberships, Context Optimizer treats these merely as cross-references and extracts the actual pathway evidence from Reactome to avoid duplication.

Chemical and Drug For compound and drug data, we utilize both ChEMBL (Zdrazil et al., 2024) and DrugBank (Knox et al., 2024) in a complementary manner. ChEMBL provides manually curated, quantitative bioactivity metrics (such

as IC50 values) for small molecules against biological targets. In contrast, DrugBank supplies qualitative, clinical-stage information, including mechanisms of action, approved indications, and pharmacokinetic properties. The system extracts structured evidence from both sources using deterministic templates, covering both research-stage bioactivity and clinical positioning.

Pathway Reactome provides the core reaction-level pathway models (Milacic et al., 2024). The system extracts individual evidence records for each molecular event, capturing the specific roles of substrates, catalysts, and regulators. This reaction-level decomposition is crucial when Hypothesis Generator needs candidate intermediate steps for mechanistic reasoning. We supplement Reactome with KEGG (Kanehisa et al., 2025) only in areas where Reactome's coverage is sparse, such as specific metabolic compound networks. Any signaling pathway data that overlaps with Reactome is deliberately excluded during KEGG extraction.

Protein Structure It is important to note that Context Optimizer does not ingest 3D atomic coordinates directly as evidence. The In Silico Scientist (B.4) handles the actual structural files for tasks like docking and conformational simulation. Instead, Context Optimizer extracts structural metadata to be used as ordinary atomic claims. From the Protein Data Bank (PDB) (Burley et al., 2025), the system extracts evidence regarding experimentally determined structures, bound ligands, and complex compositions. Similarly, from the AlphaFold Protein Structure Database (Varadi et al., 2024), it captures metadata on AI-predicted structures and their associated confidence scores (pLDDT). This allows Hypothesis Generator to reason about structural availability and reliability without handling the heavy structural files.

Extensibility The sources listed above represent the current pipeline integrations. However, the system's architecture allows for the seamless addition of other data categories, such as genomic variation or expression databases, using the exact same retrieval and extraction patterns whenever new query domains require them.

C.2.3. In-House Database

Role. The in-house database serves as a private, persistent storage system for the specific research group operating the pipeline. It holds two main types of content. First, it stores pre-existing archives that the group already possesses, such as previous experimental data, internal protocols, and historical notes. Second, it captures new records generated by the pipeline during active operation. These include computational and wet-lab outcomes, negative results, diagnostic reports, and the report-summary pairs assembled by the Knowledge Consolidator. Unlike public external sources, this database remains strictly confidential to the research team. It represents the group's unique experimental history rather than general community knowledge.

Deposition. Teams load their pre-existing archives into this database through a one-time ingestion process before the pipeline even starts. Once the system is running, all new records enter exclusively through the Knowledge Consolidator. When an experimental episode concludes, this module submits the results to the Context Optimizer using the `make_experiment_record_evidence` API. This submission pairs a human-readable report with a machine-queryable summary under a single shared record ID. The report is saved as raw data, while the summary functions as its metadata. The Context Optimizer then uses this summary to extract evidence directly into the Internal branch of the context tree. To ensure strict provenance, every piece of evidence retains a pointer back to the original database entry, matching the tracking method used for external sources. Records in the Internal branch also receive an originating hypothesis node ID and a verdict, both of which are permanently stamped during the initial extraction phase.

Retrieval. During subsequent cycles, the system retrieves Internal branch using the exact same APIs it uses for external data. At the content level, the Human Guide makes no distinction between a published external citation and an internal experimental result. Instead, the API separates them structurally using a branch path filter. The system can also use a hypothesis node ID filter to specifically target experiment-derived evidence based on the exact hypothesis that originated it. This unified approach directly enables continuous cross-cycle evidence accumulation. It ensures that the system processes a failed wet-lab experiment from yesterday and a newly published paper from today on completely equal footing, evaluating them through the same reasoning module and the identical interface.

C.3. State Memory (cross-pipeline view)

As a data structure exclusive to the Hypothesis Generator (B.2), State Memory is never traversed directly by external modules. When a hypothesis graduates, only its individual node reference and a compact provenance summary cross pipeline boundaries to reach the Experiment Designer and downstream execution stages. At the end of each cycle, the Knowledge Consolidator writes all experimental records to the Context Optimizer instead of modifying State Memory directly. Subsequent cycles retrieve both the archived tree traces and the fresh experimental records through the Context Optimizer to reconstruct the relevant node context during warm-start initialization. This structural separation ensures that the tree tracks hypothesis resolution while the Internal branch preserves the actual findings. As a result, active State Memory can be safely compacted after each cycle without losing any reasoning history or experimental outcomes.

C.4. Provenance Tracking

Every record generated by the pipeline maintains a provenance pointer indicating its direct origin. Rather than relying on a centralized provenance module, the system distributes this tracking responsibility. Each module independently attaches a provenance pointer to its own output at the exact moment of creation, securely linking the new record back to the specific inputs it used. This uniform convention guarantees that any final hypothesis can be thoroughly audited by traversing its chain backward through every prior decision, retrieval, and experimental outcome.

Provenance naturally accumulates across four main stages without requiring redundant data entry. First, the Context Optimizer embeds span pointers referencing the exact location in the raw data source when extracting evidence. Second, the Hypothesis Generator records the developmental context of each node, securely linking it to its parent hypothesis and the retrieved evidence. Third, experimental specifications point back to their graduated hypotheses, while subsequent result records and diagnostic reports link back to their originating specifications. Finally, the Knowledge Consolidator compiles this seamlessly connected chain and deposits it alongside the record into the in-house database via Context Optimizer.

This distributed architecture enforces strict single-source writes, completely avoiding the coordination bottlenecks of a centralized store. Furthermore, by using a unified retrieval interface, the Hypothesis Generator accesses in-house experimental results exactly like published literature. Since the Knowledge Consolidator pre-packages these outcomes with their complete provenance, the reasoning module immediately inherits the full historical context of any internal finding. This frictionless integration directly enables deep cross-cycle reasoning.

D. Worked Example

This section provides a virtual scenario workflow (Figure 8) demonstrating how EGAS operates across multiple cycles to achieve an open-ended scientific goal.

User Query: *"Analyze the characteristics of T cell exhaustion, find a target to block it, and design a drug. Specifically, generate time-series T cell expression data, reveal the T cell exhaustion signature, determine the time point of irreversibility, analyze the optimal reversal timing and target protein, and repurpose an approved drug capable of inhibiting this target."*

Cycle 1

Phase 1. Initialization (Cold Start) The cycle begins with a cold start since there is no prior exploration history for this query. The Hypothesis Generator's Explorer module rewrites the user query into a seed hypothesis, which serves as a global objective guiding the search: "There exists a target whose modulation can reverse T cell exhaustion, and a drug can be designed or repurposed to achieve this." This seed node is initialized in the State Memory with no assigned evidence. Simultaneously, the Context Optimizer decomposes the human query to construct the initial context tree. For example, Context Optimizer creates the initial query "T cell exhaustion" and extracts concepts such as "target identification" and "drug discovery". These parsing decisions and extracted entities are logged in the Meta branch. Human guides can monitor this Meta branch log to see what information Context Optimizer retrieves and provide direct adjustments if necessary.

Phase 2. Hypothesis Generation The Explorer accesses the root hypothesis and authors a specific query specification (e.g., "T cell exhaustion signatures and irreversible timelines") to call the Action module, which retrieves a relevant evidence pool from Context Optimizer. The Explorer partitions this evidence pool into subsets and assigns distinct reasoning modes

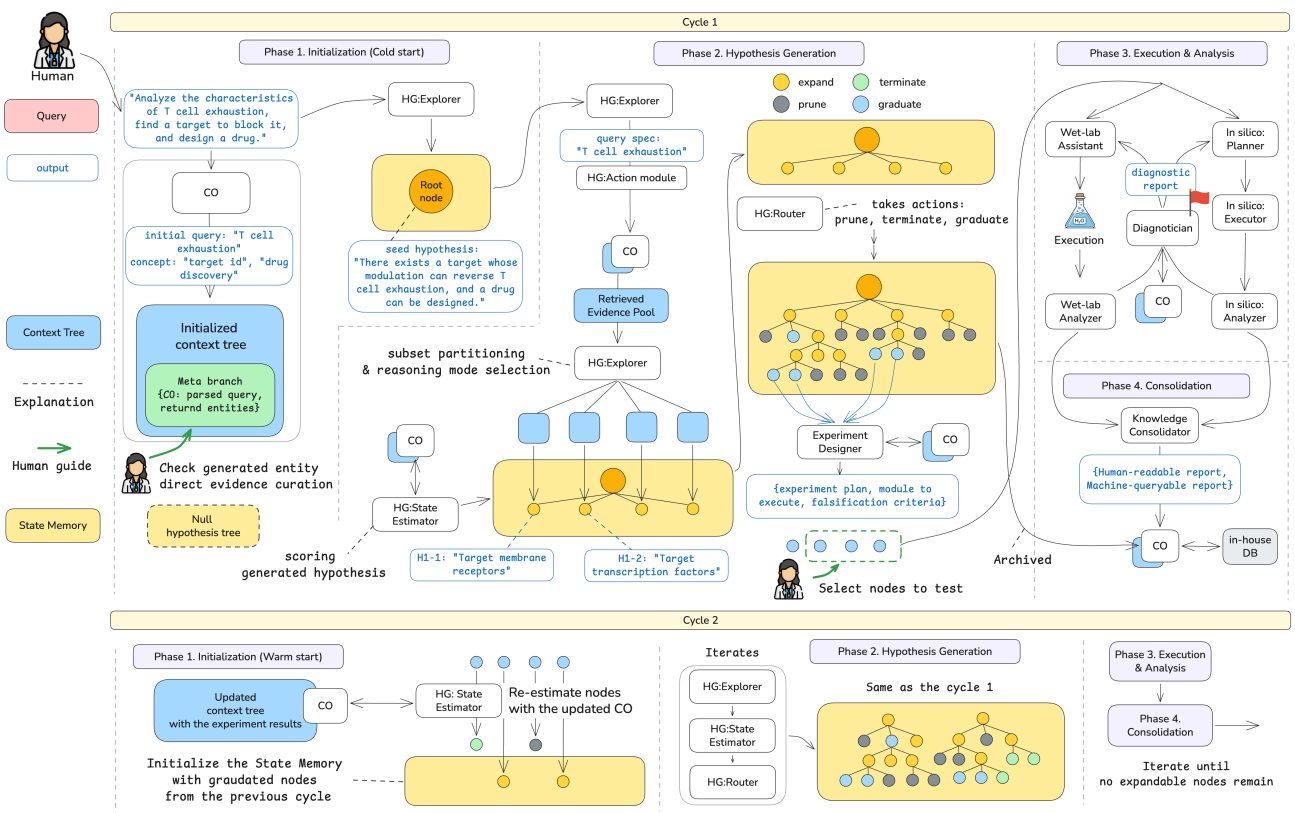


Figure 8. Workflow of EGAS. The framework operates in iterative cycles, each comprising initialization, hypothesis generation, execution and analysis, and consolidation. The Hypothesis Generator performs a Tree-of-Thoughts search to expand seed hypotheses into verifiable claims, while the Context Optimizer autonomously tracks and updates the evidence base. Graduated hypotheses are routed for computational or wet-lab execution, and the resulting experimental records are consolidated into an Internal branch to serve as evidence for subsequent warm-start cycles.

to generate multiple child hypotheses. For instance, two child nodes might emerge:

- **H1-1 (Mechanism proposal mode):** "Target membrane receptors dictate the progression of exhaustion"
- **H1-2 (Causal mode):** "Target transcription factors drive the epigenetic locking of T cell exhaustion regardless of temporal duration."

Once child nodes are generated, the State Estimator performs node-level and path-level evaluations. The Router then assigns an action to each node. For example, an unsupported node may receive a *prune* action, a partially valid node may receive an *expand* action, and a highly concrete node like H1-1 may receive a *graduate* action. For all graduated nodes, the Experiment Designer specifies a verification plan. For H1-1, it proposes generating and analyzing time-series transcriptomic data, designates the Wet-Lab Assistant as the executing module, and pre-specifies verdict criteria (e.g., "Refutes if exhaustion markers remain fully reversible across all time points"). To optimize resources, a human guide reviews the State Estimator's scores alongside the proposed experiments and decides which graduated hypotheses to physically or computationally execute.

Phase 3. Execution & Analysis Since H1-1 was assigned to the Wet-Lab Assistant, the Wet-Lab Assistant selects the appropriate protocol (e.g., a time-course RNA-seq protocol with library prep and sequencing parameters), and the Wet-Lab Analyzer processes the resulting data using its registered analysis pipeline. During analysis, the Wet-Lab Analyzer checks whether the protocol's acceptance criteria are satisfied. If acceptance fails (e.g., unexpected variance in standard

housekeeping genes), the Diagnostician is triggered in reactive mode. It conducts a root-cause analysis and forwards its report to the Wet-Lab Assistant. Based on this report, the protocol is revised and the experiment is re-executed without generating biological interpretation from untrusted data.

Phase 4. Consolidation Upon successful execution, the Knowledge Consolidator collects the data (including all troubleshooting logs) and produces a human-readable report as well as a machine-queryable summary for Context Optimizer. For H1-1, the summary might conclude that the exhaustion transcriptional signature stabilizes after a specific time point, with surface-receptor expression patterns implicating particular candidate receptors. Context Optimizer updates the context tree with this new finding and deposits the report into the in-house database. Separately, Hypothesis Generator persists its complete State Memory snapshot to Context Optimizer's StateMemoryArchive branch, closing out the cycle.

Cycle 2

Phase 1. Initialization (Warm Start) Cycle 2 begins via a warm start. The graduated nodes from Cycle 1, alongside the newly established prior experimental results (the established exhaustion timeline), serve as the new root nodes. The State Estimator re-evaluates these retained nodes using the updated evidence in the context tree. The Router assigns the *expand* action to promising nodes to initiate the next phase of hypothesis exploration.

Phases 2–4. Iterative Advancement The pipeline repeats the generation, execution, and consolidation processes. In this cycle, the Explorer generates new child hypotheses aimed at identifying the specific target protein responsible for the timeline established in Cycle 1, and eventually identifying an approved drug that inhibits this target.

In principle, EGAS repeats these cycles until no expandable nodes remain. However, a human operator can intervene between cycles to steer the progression—for example, ensuring that Cycle 1 strictly finalizes signature analysis before Cycle 2 transitions into target identification and Cycle 3 handles drug repurposing. Throughout the entire workflow, the human guide controls the evidence flow via the Meta branch and experimental gating, while the rigorous reasoning and execution are fully delegated to the autonomous agents.

E. Evaluations

EGAS targets a capability that sits upstream of existing benchmarks for scientific agents: given a high-level scientific objective, the system decomposes the goal into sub-hypotheses, generates and evaluates candidates, plans experiments, and feeds outcomes back into subsequent cycles. No single benchmark captures this end-to-end loop; current evaluations address narrower capabilities such as factual QA (Laurent et al., 2024; Phan et al., 2026), agentic reasoning on pre-specified tasks (Gao et al., 2025; Arora et al., 2025), data-analysis pipeline construction (Mitchener et al., 2025; Phylo Bio, 2026; Miller et al., 2025), hypothesis ranking against ground-truth discoveries (Guo et al., 2025; Yang et al., 2025), and domain-specific target retrieval (Leung et al., 2026). We adopt relevant benchmarks from this landscape for per-module evaluation (E.2) but the end-to-end loop from goal decomposition through experimental feedback needs to be tested separately.

For EGAS's end-to-end claim to hold, several subordinate capabilities must work: the system must retrieve and reason over evidence soundly, plan and interpret experiments that a domain expert would accept, and compound what it learns across cycles so that earlier findings inform later ones. We therefore adopt a multi-faceted evaluation that probes these capabilities at different granularities. The knowledge cutoff evaluation (E.1) is the primary axis: it tests whether EGAS's structural separation of evidence from reasoning actually produces hypotheses that go beyond the LLM's prior. Beyond end-to-end hypothesis quality, per-module probes (E.2) assess whether Context Optimizer's retrieval, Hypothesis Generator's reasoning steps, and the Diagnostician's anomaly detection each work reliably in isolation, using existing benchmarks and custom datasets. Wet-lab validation (E.3) extends evaluation to physical experiments, progressing from narrow candidate verification toward end-to-end therapeutic discovery. Long-horizon evaluation (E.4) measures whether compounding past evidence actually enhances subsequent hypothesis generation, the capability that distinguishes a sustained research program from single-query reasoning.

E.1. Knowledge Cutoff Evaluation

LLM priors favor likely continuations of existing knowledge, working against the novel connections that drive scientific ideation. EGAS addresses this through structural separation of evidence curation from reasoning. The knowledge cutoff evaluation tests whether this separation actually produces hypotheses that go beyond the prior.

We fix a knowledge cutoff and restrict the Context Optimizer's external retrieval to evidence published before the cutoff. The LLM retains its parametric knowledge, but Context Optimizer cannot retrieve post-cutoff literature, and the ground-truth discovery is not available through either channel. If EGAS generates hypotheses that recover post-cutoff discoveries from pre-cutoff evidence, this demonstrates prospective reasoning rather than reproduction of known answers. The cutoff should be recent enough that the LLM's training data provides reasonable domain coverage; a cutoff set far in the past (e.g., 2020) would test the system under artificially degraded parametric knowledge rather than testing its reasoning. We select holdout targets whose pre-cutoff evidence trail is rich enough to support prospective reasoning, to avoid penalizing the system for cases where the necessary evidence simply did not exist before the cutoff (Narganes-Carlón et al., 2023; Yang et al., 2024).

E.1.1. Literature Cutoff

We pose queries whose ground-truth answers were established in the literature after the cutoff: a therapeutic target whose mechanism was published post-cutoff, or a binder whose structure was deposited post-cutoff. We report recall of known discoveries and the ranking of the correct hypothesis among generated candidates.

Because the system's output includes not only hypotheses but the experiments it proposes to test them, this setting also exposes whether the system's experimental planning converges on methodology established in the target domain, providing a secondary signal about the experimental pipeline modules. For the specific case of target identification, TargetBench 1.0 (Leung et al., 2026) provides a published reference setup evaluating clinical target retrieval against curated ground truth, which can be adopted or adapted as a sub-benchmark for the target-identification stage of the pipeline. Analogous temporal-holdout protocols have been used for chemistry hypothesis generation (Yang et al., 2024) and for prospective target discovery (Narganes-Carlón et al., 2023).

E.1.2. Experimental Record Cutoff

A stronger variant, feasible where a single experimental group has kept detailed records, extends the cutoff from literature to experimental history: the pipeline is rolled back to a past cutoff and its proposed hypotheses, experiment plans, and protocol choices are compared against what the group actually did and found afterwards. This tests whether the system's decisions, not only its hypotheses, converge on what a working lab would have chosen, and it reuses existing experimental records rather than requiring fresh wet-lab work. We treat this as a conditional extension whose feasibility depends on access to a dated, structured archive from a collaborating lab.

E.2. Per-Module Evaluation

E.2.1. Diagnostician Evaluation

We evaluate the Diagnostician along both of its operating modes.

Reactive mode. We measure root-cause attribution accuracy on experimental failure cases. Each test case consists of a failed control output, the attempt history, and protocol metadata, with the ground-truth root cause determined from the recorded troubleshooting episode. Because such records are typically confined to internal lab notebooks rather than public repositories, constructing this benchmark requires collaboration with experimental groups willing to share structured troubleshooting histories. We report whether the Diagnostician's top-ranked candidate matches the documented root cause.

Proactive mode. We measure whether the Diagnostician correctly flags confounded data while avoiding false alarms on clean data. Test cases are drawn from public datasets where technical artifacts have been independently documented and characterized, such as probe barcode batch effects in single-cell RNA sequencing (Weir et al., 2026). We supplement these with synthetic data, where artificial batch effects of controlled magnitude are introduced into clean datasets, to test detection sensitivity across a range of effect sizes. To evaluate the contribution of the template-based check, we compare performance with and without access to prior diagnostic case studies for the same platform, directly testing whether accumulated failure reports improve detection of platform-specific artifacts.

E.2.2. Context Optimization Evaluation

We evaluate the Context Optimizer along two axes. First, we compare Context Optimizer against standard RAG pipelines to test whether our design principles—interpretation-free curation, entity-centric indexing, and append-only provenance—produce measurable advantages in downstream reasoning. Second, we ablate individual components to verify

that observed gains stem from these architectural choices. The ability to cleanly swap Context Optimizer in and out for these experiments itself demonstrates the modular separation between evidence curation and reasoning outlined in Section A.2.

A Note on Evaluation Methodology Standard retrieval metrics like Recall@k or nDCG@k evaluate whether a system returns the gold passage for a natural-language query, the paradigm underlying benchmarks such as BEIR (Thakur et al., 2021). Context Optimizer rejects this paradigm: it accepts structured query specifications instead of natural-language questions, returns atomic claims rather than passages, and sorts by neutral criteria rather than learned relevance. We therefore evaluate Context Optimizer primarily through its downstream effect on the Hypothesis Generator, reserving standard IR metrics for internal sanity checks on individual components.

Context Optimizer vs RAG Comparison Context Optimizer differs structurally from conventional RAG in three ways: it returns atomic, source-grounded claims rather than passage chunks—a granularity motivated by prior findings on fine-grained factuality assessment (Min et al., 2023); it sorts by neutral criteria (extraction time, cycle stage) instead of learned relevance; and it exposes evidence through stateless, query-spec-driven APIs rather than a single similarity-based interface.

We compare Context Optimizer against three baselines of increasing sophistication: dense retrieval over chunked documents using E5 (Wang et al., 2022) or BGE-M3 (Chen et al., 2024a); hybrid retrieval combining BM25 with ColBERTv2 late-interaction reranking (Santhanam et al., 2022); and an LLM-augmented RAG pipeline such as HyDE (Gao et al., 2023) that injects LLM interpretation directly into retrieval. The third baseline is critical because it represents the entangled retrieval-and-interpretation paradigm Context Optimizer is designed to avoid (Abe et al., 2025).

For the primary comparison, we hold Hypothesis Generator fixed and swap the evidence layer between Context Optimizer and the three baselines, measuring hypothesis quality with HypoBench (Liu et al., 2025) and IdeaBench (Guo et al., 2025) alongside search-process metrics like diversity and grounding. We expect Context Optimizer to match or exceed RAG baselines on quality while showing higher diversity, since LLM-augmented retrieval imports interpretive framing that narrows the downstream inference space.

As supporting checks, we evaluate the faithfulness of atomic claim extraction using AttributionBench (Li et al., 2024) and the accuracy of entity linking against BELB (Garda et al., 2023). These confirm that internal mechanics operate correctly, ensuring observed downstream effects reflect design choices rather than implementation flaws. The knowledge cutoff setting is the most direct test of our central claim that interpretation-free curation lets the system escape LLM biases and recover newly published discoveries. We conduct evidence-supplier swaps under this setting as part of the end-to-end evaluation in Section E.1

Ablation Studies To isolate each design choice, we report both intrinsic metrics and extrinsic effects on downstream outputs. This separation matters because features like neutral sorting may underperform on standard IR metrics while improving downstream goals like hypothesis diversity.

- **Atomic Claim Extraction.** Context Optimizer uses an LLM to extract factual and causal claims tied to single raw sources (Min et al., 2023). We replace this with passage-level retrieval to test whether atomic decomposition genuinely improves grounding fidelity, measuring the fraction of hypotheses unambiguously attributable to specific evidence units via AttributionBench (Li et al., 2024).
- **Entity-Centric Indexing.** The system normally maps entity mentions to canonical IDs using tools like BioBERT (Lee et al., 2020) or SciSpacy (Neumann et al., 2019). We ablate in two stages: removing ontology linking so entities match by surface form only, then removing entity extraction entirely. We measure linking accuracy against BELB (Garda et al., 2023) to test whether canonical resolution adds value beyond string matching, especially for the rare entity pairings that drive the system's "rare tilt" search strategy.
- **Neutral Sorting.** This is our most contested choice, since standard IR practice favors relevance ranking. We replace neutral sorting with a learned reranker (e.g., ColBERTv2 (Santhanam et al., 2022) or LLM-based scoring), hypothesizing that learned rerankers promote mainstream connections and suppress rare ones—importing the very bias the system aims to mitigate (Abe et al., 2025; Sciavolino et al., 2021). We expect the reranker to win on standard IR metrics but lose on downstream diversity and post-cutoff discovery, validating the design rationale through this trade-off.
- **Query Spec Translation.** Cognitive modules normally send structured query specifications that the Action module translates into API calls. We collapse this into a single natural-language query passed to a similarity-based retriever

(e.g., E5 (Wang et al., 2022) or BGE-M3 (Chen et al., 2024a)) to test whether explicitly separating strategic intent from mechanical retrieval improves search effectiveness. We measure the effect through Hypothesis Generator's exploration-exploitation balance and search behavior.

E.2.3. Hypothesis Generation Evaluation

We evaluate the Hypothesis Generator with the goal of verifying whether its design principles translate into measurable improvements in both output quality and search behavior. Hypothesis Generator is designed to (1) induce structural diversity through evidence subset partitioning and reasoning modes, (2) maintain grounding through node- and path-level evaluation, and (3) perform hypothesis generation as a Tree-of-Thoughts (ToT) search process. To assess these properties, we evaluate Hypothesis Generator along two complementary axes: hypothesis quality and search process integrity.

Hypothesis Quality Evaluation. We first evaluate the quality of the final hypotheses produced by Hypothesis Generator, i.e., the set of candidate hypotheses (graduated nodes) obtained at the end of the ToT search. This evaluation follows recent work on biomedical hypothesis generation, including Dyport (Tyagin & Safro, 2024) and LLM-based biomedical hypothesis generation studies (Qi et al., 2024), which treat hypothesis generation as a distinct scientific task rather than as generic biomedical QA or text generation. We measure whether the system produces high-quality, competitive hypotheses in terms of explanatory power, novelty, feasibility, and experimental readiness.

We employ existing benchmarks tailored to different aspects of hypothesis generation. HypoBench (Liu et al., 2025) is used to evaluate explanatory hypothesis generation, where the objective is to produce hypotheses that account for observed phenomena. Its combination of real-world and synthetic datasets enables controlled evaluation, including robustness under noisy or incomplete evidence. IdeaBench (Guo et al., 2025) complements this by evaluating open-ended scientific ideation. In this setting, the model generates research ideas based on reference papers, without access to the target paper, and its outputs are compared against human-authored ideas using an LLM-as-a-judge protocol. The resulting Insight Score measures the relative competitiveness of generated ideas in terms of novelty, feasibility, and clarity. This allows us to assess whether Hypothesis Generator's structurally induced diversity leads to meaningful exploration of new research directions.

We further evaluate the domain validity of individual reasoning chains using biomedical reasoning and agent-task benchmarks, including Biomni-style evaluations (Huang et al., 2025) and CURE-Bench (Gao et al., 2025). These benchmarks are not treated as direct measures of open-ended hypothesis search; rather, they serve as auxiliary probes for whether individual Hypothesis Generator's hypotheses are supported by coherent biomedical reasoning, valid data handling, and causal or mechanistic consistency. Unlike search process evaluation, which considers interactions across multiple hypotheses, these benchmarks evaluate the validity of individual reasoning chains.

Search Process Integrity Evaluation. While hypothesis quality evaluates final outputs, it does not capture whether these outputs are produced through meaningful search. Prior work has shown that evaluating only final answers is insufficient for reasoning systems, motivating approaches that consider intermediate reasoning steps and trajectories beyond final answers (Cobbe et al., 2021; Lightman et al., 2024; Yao et al., 2023). Structured reasoning methods (Sprueill et al., 2023; 2024) further motivate treating reasoning as an explicit search process rather than as single-pass generation. This perspective is also consistent with autonomous scientific agent systems such as The AI Scientist (Lu et al., 2026) and AI Co-Scientist (Gottweis et al., 2025), where scientific discovery is modeled as an iterative loop involving proposal generation, evaluation, tool use, and refinement.

However, such process evaluations are typically defined for tasks with well-specified ground truth. In open-ended hypothesis generation, where gold trajectories are unavailable, we instead evaluate search through observable behavioral signatures. We define diversity, grounding, and search behavior as the three complementary criteria to evaluate different aspects of the search process: coverage of the hypothesis space, local validity of reasoning steps, and global progression of the search trajectory.

Diversity. We evaluate whether Hypothesis Generator explores distinct regions of the hypothesis space. Prior work on creative or set-valued hypothesis generation (Chen et al., 2025; Gero et al., 2022) primarily evaluates diversity at the output level, for example by measuring semantic differences among generated hypotheses or inspirations. In contrast, we assess diversity as a property of the search process. We measure diversity both at the leaf level and along search trajectories, capturing not only differences in final hypotheses but also the extent to which the system explores distinct reasoning paths. Concretely, we quantify this through hypothesis embedding similarity, evidence subset overlap across nodes, and the

distribution of reasoning modes used during search. High diversity indicates that the system explores multiple reasoning trajectories rather than producing redundant variations.

Grounding. We evaluate whether reasoning remains supported by evidence throughout the search trajectory. This includes detecting unsupported inference, over-claims, and semantic units that lack evidential backing. We operationalize grounding using signals produced by the State Estimator, including support/refute ratios, semantic grounding status at the unit level, and distortion risk, which captures whether reasoning transitions introduced unsupported inference or over-specification.

In addition to these internal signals, we use literature-grounded biomedical and chemistry QA/RAG benchmarks as external grounding probes. BioASQ-QA (Krithara et al., 2023) and PubMedQA (Jin et al., 2019) test biomedical literature-grounded reasoning, while ScholarChemQA (Chen et al., 2024b) and ChemRAG-Bench (Zhong et al., 2025) provide analogous evaluation settings for chemistry literature and retrieval-augmented reasoning. Chemistry reasoning benchmarks such as ChemLLMBench (Guo et al., 2023) and ChemBench (Mirza et al., 2025) further test whether chemistry-specific claims are domain-valid rather than merely fluent. Unlike final-answer evaluation, this criterion assesses whether grounding is maintained across intermediate reasoning steps.

Search Behavior. We evaluate whether the sequence of refinement steps forms an effective search trajectory. While grounding assesses the validity of individual steps, search behavior evaluates how effectively the search process navigates the hypothesis space. We measure (1) discovery efficiency, defined as the number of explored nodes required to identify high-quality hypotheses under a fixed compute budget, (2) trajectory-level improvement, defined as the likelihood of discovering better hypotheses as search progresses, and (3) redundancy versus exploration, defined as the extent to which the search avoids repeated generation of similar hypotheses. We further compare ToT against best-of-N sampling under matched computational budgets, isolating the contribution of structured search over naive parallel generation. If Hypothesis Generator achieves comparable or better hypothesis quality under the same computational budget with lower redundancy, this supports the claim that its tree structure contributes to more effective search rather than merely increasing sample count.

Together, these criteria allow us to validate Hypothesis Generator's core claim: that structured search over evidence-grounded reasoning paths produces better hypotheses than unstructured generation.

E.3. Wet-Lab Validation

Wet-lab validation is treated as a layered extension rather than a required axis. At minimum, validation includes synthesis and assays for a small panel of candidates for a narrow, pre-scoped goal, such as small-molecule drug design against a specified target. This gives a closed-loop demonstration that is modest in scale but physically grounded, and it is the scope of validation most existing AI-for-drug-discovery systems reach. More ambitiously, an end-to-end test, in which the pipeline is handed an open-ended therapeutic goal (a disease, a patient subgroup) and its proposed targets and modalities are followed through to wet-lab confirmation, would exercise the goal-decomposition and evidence-accumulation claims of the framework in a way that narrow-scope validation cannot. Feasibility of the stronger test depends on lab capacity and development timelines.

E.4. Long-Horizon Evaluation

The knowledge cutoff evaluation (E.1) runs the full pipeline to termination, potentially across many cycles, testing whether the system recovers post-cutoff discoveries from pre-cutoff evidence. A successful run shows that the system reached the right endpoint, but it does not reveal whether the compounding mechanism specifically contributed. A system that generates the correct hypothesis in an early cycle and carries it forward without meaningfully building on deposited evidence would pass the knowledge cutoff evaluation while failing to compound.

We instrument knowledge cutoff runs with additional measurements that target the compounding mechanism directly. These measurements require no experimental overhead beyond logging, since they operate on artifacts the pipeline already produces.

Retrieval utilization. At each cycle boundary, we check whether Context Optimizer queries return records deposited by earlier cycles and whether the Hypothesis Generator incorporates those records into its reasoning paths. Concretely, we measure the fraction of Hypothesis Generator query specifications that match internal evidence from prior cycles, and the fraction of State Estimator grounding maps that cite such evidence. A system that deposits records but never retrieves them

would pass single-cycle metrics while failing to compound.

Hypothesis convergence dynamics. We track how the distribution of active hypotheses evolves across cycles. Because the knowledge cutoff provides a known post-cutoff discovery sequence, we have a ground-truth trajectory against which to evaluate convergence. In early cycles, before the system has accumulated experimental evidence, the hypothesis frontier should be broad (many sibling branches with distinct reasoning modes and evidence subsets). As cycles progress and deposited evidence increasingly supports or refutes specific branches, the frontier should concentrate on the directions that align with the known post-cutoff discovery. We measure effective diversity at each cycle boundary through embedding similarity among active leaf nodes and the distribution of reasoning modes in use, and we track whether branches receiving experimental support attract proportionally more expansion while refuted branches are pruned. This complements the recall metric in E.1 by tracking *how* the system arrived at its final ranking, not just whether the ranking was correct.

Self-reinforcement probe. A risk specific to multi-cycle systems is that Hypothesis Generator may treat its own deposited evidence as more authoritative than external evidence, creating a feedback loop driven by the same prior bias that favors likely continuations of existing knowledge (Liu et al., 2024). Within the knowledge cutoff runs, we test for this by examining cases where deposited internal evidence and external pre-cutoff evidence point in different directions for the same sub-hypothesis. If the system systematically follows internal evidence over equally or more authoritative external evidence, this indicates self-reinforcement. The knowledge cutoff makes this measurement interpretable: because the ground-truth discovery is known, we can identify cases where the system should have revised a branch based on external evidence but did not.

Controlled injection. As a standalone complement to the instrumented knowledge cutoff runs, we evaluate compounding behavior under controlled conditions by injecting synthetic experimental outcomes at specific cycle boundaries. We provide results that support one branch and contradict another, then measure whether Hypothesis Generator appropriately expands the supported branch, prunes or revises the contradicted branch, and reflects both outcomes in subsequent State Estimator grounding maps. This isolates the compounding mechanism from the confounds of real retrieval quality and evaluation target difficulty, and can be run without a full knowledge cutoff setup.

Where feasible, a **reset comparison** provides a stronger signal for either the instrumented or controlled-injection setting: the same query is run with the internal evidence cleared between cycles, isolating the contribution of accumulated evidence from within-cycle effects. If the persistent-evidence condition produces measurably better later-cycle hypotheses than the reset condition, the improvement can be attributed to cross-cycle compounding. We treat this as a recommended extension, since controlled runs of this kind are resource-intensive.

Together, these measurements test whether the compounding mechanism claimed by EGAS is operational, beneficial, and robust to self-reinforcement, capabilities that distinguish a sustained discovery program from single-query reasoning.

F. Comparison with Existing AI Scientist Systems

F.1. Comparison Axes

1. Evidence–Reasoning (Coupled / Partial / Channel / Decoupled)
How does evidence (literature, data, experimental results) enter the system, and is evidence curation architecturally separated from reasoning/interpretation?

- **Coupled**: the same agent/module retrieves and reasons simultaneously, no separation
- **Partial**: retrieval exists as a tool but the retrieving agent also interprets and reasons over what it finds
- **Channel**: evidence flows through a dedicated retrieval channel but some interpretation leaks in
- **Decoupled**: a dedicated module curates and indexes evidence without interpretation; reasoning modules access evidence only through a stateless retrieval interface

2. Structurally Induced Diversity (Yes / No)
Is hypothesis diversity induced by architectural design rather than by stochastic sampling or agent-level variation alone? `Yes` requires that distinct reasoning paths arise from a structural mechanism such as partitioning retrieved evidence into disjoint

subsets and exploring each under different reasoning modes, so sibling hypotheses differ along interpretable dimensions rather than by random variation. Temperature sampling, multiple seeds, persona-based agents, debates, and tournament evolution are not sufficient on their own.

3. Negative-Result Reasoning (Yes / No)
Are failed or negative experimental outcomes explicitly analyzed and re-entered into the system as inputs to subsequent cycles, rather than discarded? `Yes` requires that the system reasons over failure cases (e.g., failed syntheses, search stalls, refuted hypotheses) and uses the analysis to redirect later hypothesis generation or experimental design. Simply logging failures without using them does not qualify.

4. Wet-Lab Integration (Yes / No)
Does the system close the experimental loop through system-generated protocols, robotic or human execution, and downstream analysis of the resulting data? `Yes` requires at minimum dedicated module(s) that produce executable experimental protocols and consume the resulting data, beyond suggesting experiments as part of hypothesis text. In-silico-only validation, or hypothesis output without protocol generation, does not qualify.

5. Human-in-the-Loop Collaboration (Yes / No)
Can the scientist provide mid-run, hypothesis-level guidance beyond the initial goal specification? `Yes` requires that the system accepts ongoing human input during execution (e.g., reviews of intermediate outputs, additional hypotheses, refocusing of the search, or steering of the evidence base) rather than only consuming an initial prompt and returning a final answer.

6. Multi-Agent / Multi-Role Architecture (Yes / No)
Does the system decompose its workflow across multiple specialized agents or modules with distinct roles, rather than relying on a single general-purpose agent that calls tools? `Yes` requires that responsibilities such as literature retrieval, hypothesis generation, critique, ranking, experiment design, or analysis are assigned to architecturally distinct components. A single LLM agent invoking external tools or commands does not qualify.

7. Domain-Knowledge Integration (Yes / No)
Is the system equipped with domain-specific tools, databases, or knowledge bases beyond the parametric knowledge of a general-purpose LLM? `Yes` requires explicit integration of domain resources such as curated chemistry tools, biomedical databases, omics knowledge bases, or materials property databases. Systems that operate purely on a general-purpose LLM with web search and code execution do not qualify.

8. Multi-Step Workflow (Yes / No)
Does the system proceed through multiple distinct stages rather than producing an answer in a single forward pass? Stages here include goal interpretation, evidence gathering, hypothesis generation, evaluation, and experimental planning. `Yes` requires that the workflow exposes identifiable stages with handoffs between them, whether implemented as fixed phases, agent loops, or iterative refinement.

F.2. Comparison Table

Table 6. Comparison of EGAS with existing AI scientist systems. Axis definitions are provided in Section F.1. ● = yes; ○ = no. Evidence–Reasoning levels: Coupled → Partial → Channel → Decoupled. Score counts satisfied binary axes (out of 7).

System	Ev–Reas	Struct. Div.	Neg. Result	Wet-lab	Human	Multi-agent	Domain	Multi-step	Score
AI Scientist (Lu et al., 2026)	Coupled	○	○	○	○	●	○	●	2/7
Biomni (Huang et al., 2025)	Coupled	○	○	○	○	○	●	●	2/7
ChemCrow (Bran et al., 2024)	Coupled	○	○	●	○	○	●	●	3/7
MEDEA (Sui et al., 2026)	Channel	○	○	○	○	●	●	●	3/7
Virtual Lab (Swanson et al., 2025)	Coupled	○	○	○	●	●	●	●	4/7
AI Co-Scientist (Gottweis et al., 2025)	Partial	○	○	○	●	●	●	●	4/7
SAGA (Du et al., 2025)	Coupled	○	●	○	●	●	●	●	5/7
EGAS (Ours)	**Decoupled**	●	●	●	●	●	●	●	**7/7**

Table 6 evaluates each system on the eight axes defined in Section F.1. Systems are ordered by Evidence–Reasoning level, then by score.

F.3. Positioning of EGAS

Several systems in Table 6 share individual capabilities with EGAS. The AI Co-Scientist (Gottweis et al., 2025) supports open-ended goals with rich human collaboration; SAGA (Du et al., 2025) reasons over negative outcomes to redirect its optimization loop; ChemCrow (Bran et al., 2024) integrates wet-lab workflows through protocol generation and robotic or human execution; and most recent systems integrate domain-specific tools and multi-step workflows. EGAS's positioning rests not on any single axis but on a specific combination that no existing system provides.

Evidence–Reasoning decoupling. All compared systems allow some degree of interpretation to occur during evidence retrieval. In systems classified as Coupled, the same agent that retrieves literature or data also reasons over it within the same call. The AI Co-Scientist (Partial) uses web search as a tool within its Generation and Reflection agents, which simultaneously interpret what they find. MEDEA (Channel) comes closest to separation (Sui et al., 2026). Its LiteratureReasoning module is architecturally distinct from the Analysis module, but it synthesizes and interprets retrieved papers before passing results to the consensus stage. In EGAS, the Context Optimizer curates and indexes evidence without interpretation, and the Hypothesis Generator accesses it exclusively through stateless retrieval APIs. This separation ensures that the evidence base is not filtered by the LLM's prior before reasoning begins, preserving room for unexpected connections.

Structurally induced diversity. Existing systems induce hypothesis diversity through agent-level mechanisms. The AI Co-Scientist uses persona-based debate and tournament evolution; Virtual Lab (Swanson et al., 2025) uses specialist agents with parallel high-temperature meetings; SAGA uses objective evolution across outer-loop iterations. These approaches produce variation, but sibling hypotheses differ by stochastic sampling or agent personality rather than along interpretable, evidence-grounded dimensions. EGAS's Explorer partitions retrieved evidence into disjoint subsets and assigns each a distinct reasoning mode (causal, analogical, mechanism proposal, gap-filling), so sibling hypotheses differ because they reason over different evidence through different cognitive strategies.

Negative-result reasoning and cross-cycle compounding. SAGA is the only compared system that explicitly reasons over negative outcomes. Its Analyzer identifies failure modes (conflicting objectives, over-represented labile moieties) and feeds structured reports back to the Planner. However, SAGA's feedback loop operates within computational optimization cycles and does not persist outcomes into a knowledge base for future research campaigns. In EGAS, the Knowledge Consolidator deposits all experimental outcomes, including negative results and diagnostic records, into the Internal branch as structured evidence. The Context Optimizer retrieves these records in subsequent cycles on the same terms as external evidence, transforming the lab's accumulated experience into a compounding, retrievable asset that prevents redundant exploration of previously refuted mechanisms.

Combined positioning. These three capabilities are complementary. Decoupled evidence curation from reasoning ensures that the growing evidence base is not narrowed by the LLM's prior. Structurally induced diversity ensures that hypothesis exploration reaches beyond familiar associations. Persistent deposition of all experimental outcomes, including negative results, ensures that the evidence base grows with each experimental episode rather than resetting. Together, they enable the sustained, evidence-grounded ideation across a research program that is the core claim of EGAS.

F.4. Worked Comparison with the AI Co-Scientist

To illustrate how our framework differs from existing AI scientist systems while sharing several high-level goals, we trace both architectures through the same illustrative task. Gottweis et al. (2025) uses a recurring ALS example in which the scientist provides a research goal:

> *Develop a novel hypothesis related to phosphorylation of the Nuclear Pore Complex (NPC) as a causative mechanism for Amyotrophic Lateral Sclerosis (ALS).*

The AI co-scientist processes this goal through a multi-agent generate–review–rank–evolve loop. A Generation agent proposes candidate mechanisms via literature exploration and simulated scientific debate; a Reflection agent critiques assumptions and novelty; a Ranking agent compares hypotheses through Elo-based tournament evaluation; and Evolution and Meta-review agents refine the hypothesis space over subsequent iterations. The system's appendix example produces a hypothesis centered on stress-induced post-translational modification (PTM) of FG-nucleoporins such as Nup98 and Nup62, altered interaction with TDP-43, increased TDP-43 retention at the NPC, disruption of nucleocytoplasmic transport, and

contribution to ALS pathology. We use this hypothesis to illustrate and compare system behavior, not to make independent scientific claims. The biological details and hypotheses are taken from the Gottweis et al. (2025) appendix example.

Evidence curation before reasoning. Our framework would begin from the same scientific goal. The Context Optimizer would decompose the goal into entities and concepts—ALS, NPC, Nup98, Nup62, TDP-43, phosphorylation, O-GlcNAcylation, nucleocytoplasmic transport, cellular stress, and motor neuron vulnerability—and retrieve external literature, public database entries, and any relevant in-house experimental results. Each source would be converted into an atomic evidence record with provenance. Importantly, Context Optimizer would not interpret these records, rank their relevance, or decide whether NPC phosphorylation is likely to cause ALS. It would only curate and expose grounded evidence through its retrieval interface.

Hypothesis tree construction. The Explorer initializes the search by rewriting the research goal into a seed hypothesis (H0): *NPC phosphorylation may contribute to ALS-relevant nucleocytoplasmic transport defects.* The Explorer would then expand H0 into multiple child branches using diverse reasoning modes, constructing a hypothesis tree rather than producing a single ranked proposal. One early branch (H1) might recover a hypothesis similar to the AI co-scientist example: *stress-induced phosphorylation or O-GlcNAcylation of Nup98/Nup62 increases TDP-43 residence at the NPC, causing nucleocytoplasmic transport defects in ALS motor neurons.* Other branches could explore alternative mechanisms, such as indirect effects on importin/exportin selectivity (H2), motor-neuron-specific vulnerability of long-lived nucleoporins (H3), the possibility that reported TDP-43–NPC associations reflect a stress-granule or fractionation artifact (H4), or phosphorylation of a non-Nup NPC-associated transport regulator as the proximal ALS mechanism (H5). Each branch is annotated with its reasoning mode (causal, mechanism proposal, gap-filling, or analogical), and all branches are maintained as auditable tree nodes rather than collapsed into a single output.

Grounding assessment and tree refinement. The State Estimator would decompose H1 into semantic units and checks each against retrieved evidence (Sub-2). Node-level evaluation then estimates validity, expected gain from further expansion, and experiment readiness. Path-level attribution checks whether the evidence selection, reasoning mode, and query scope were appropriate for the claim produced. For the NPC-phosphorylation hypothesis, the likely well-supported components are the association of ALS with TDP-43 pathology and nucleocytoplasmic transport defects. The likely weak point is the direct causal link from Nup98/Nup62 PTM to altered TDP-43 retention at the NPC. This is a gap also identified by the AI co-scientist's Reflection agent, but here represented as a structured localization of evidential weakness rather than only a natural-language critique.

Given this assessment, the Router would not immediately graduate H1 to experimentation. Instead, recognizing that the mechanism linking PTM to TDP-43 retention is underspecified, it would call `expand`. The Explorer would query the Context Optimizer for more specific evidence about FG-repeat interaction dynamics and TDP-43 binding, producing a refined child node:

> **H1.1:** *In ALS-relevant cellular stress states, phosphorylation/O-GlcNAcylation of Nup98 and/or Nup62 alters FG-Nup interaction dynamics with TDP-43, increasing TDP-43 retention at the NPC and thereby impairing nucleocytoplasmic transport in motor neurons.*

H1.1 adds mechanistic specificity about FG-Nup interaction dynamics that was absent in H1. This version, now sufficiently concrete and testable, would be graduated by the Router to the Experiment Designer.

Experimental graduation and evidence feedback. The Experiment Designer would translate H1.1 into verdict criteria. An in silico episode could prioritize candidate phosphorylation sites on Nup98/Nup62 and assess whether they overlap with disordered FG-repeat regions or predicted interaction surfaces. A wet-lab episode could perturb candidate PTM sites, measure TDP-43 residence or colocalization at the NPC, and assay nucleocytoplasmic transport using NLS/NES reporters in motor-neuron models. A supporting result would require the PTM perturbation to alter TDP-43–NPC association and transport function in the predicted direction; a contradictory result would show no such effect under validated assay conditions.

If experimental results showed that a Nup62 phosphomimetic mutant increased TDP-43 NPC retention under stress while a Nup98 perturbation showed no effect, the Knowledge Consolidator would deposit these outcomes as structured internal evidence. In the next cycle, H1.1 would be retained as a warm-start candidate node. The Context Optimizer would retrieve

Table 7. Conceptual comparison between the AI co-scientist and our framework on the ALS/NPC worked example.

Dimension	AI co-scientist	Our framework
Organizing principle	Multi-agent generation, review, ranking, and evolution within a tournament loop	Decoupled evidence curation (Context Optimizer) and hypothesis reasoning (Hypothesis Generator) connected through explicit retrieval APIs
Evidence handling	Agents search and reason over literature during generation and review	Context Optimizer curates atomic evidence records without interpretation; Hypothesis Generator retrieves selectively via query specifications
Hypothesis improvement	Tournament ranking, scientific debate, evolution, and meta-review prompt feedback	Tree search with node-level grounding analysis, semantic-unit failure localization, and experimental graduation
Evaluation granularity	Assumption decomposition with independent sub-assumption verification; natural-language reasoning about validity; Elo-based relative ranking across hypotheses	Structured node-level and path-level scores; each semantic unit mapped to support, gap, or over-claim status
Long-horizon memory	Context memory and prompt-level meta-review feedback across iterations	In-house database of structured experimental outcomes, including negative and diagnostic records, retrievable as internal evidence in subsequent cycles

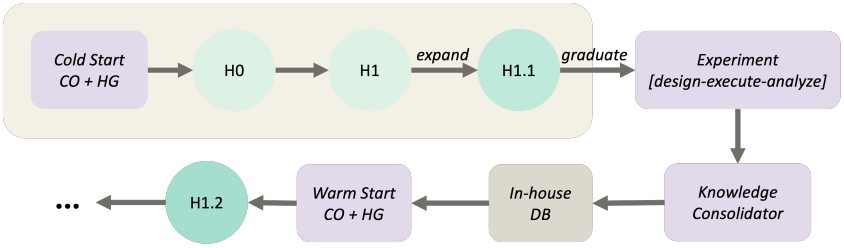
the deposited experimental outcomes alongside external evidence, and the State Estimator would re-evaluate H1.1 against this expanded evidence pool. Hypothesis Generator would then refine the branch toward a more specific Nup62-centered mechanism (H1.2):

> **H1.2:** *Stress-induced phosphorylation of Nup62, rather than Nup98, increases TDP-43 residence time at the NPC and partially drives nucleocytoplasmic transport impairment in ALS-relevant motor neurons.*

Negative and inconclusive outcomes including diagnostic records such as antibody specificity failures or high-variance imaging batches would likewise remain retrievable as internal evidence. This prevents future cycles from repeatedly exploring the same unsupported mechanism.

Key architectural differences. Both systems accept open-ended natural-language scientific goals, use literature-grounded reasoning, produce testable hypotheses, and keep the scientist in the loop. However, they are organized around different principles. In the AI co-scientist, hypotheses improve through review, debate, Elo-based ranking, and evolution within a multi-agent tournament. In our framework, hypotheses improve through evidence-grounded tree search, explicit failure localization, experimental graduation, and deposition of results back into the Internal branch. Table 7 summarizes these distinctions across five dimensions. The result is a pipeline designed not only to generate plausible hypotheses, but to accumulate grounded experimental knowledge over long-horizon discovery.

Simplified pipeline of EGAS



How core modules of EGAS would work towards hypothesis generation for ALS causality

CO **Context Optimizer** constructs a query-local evidence context by decomposing the goal into entities & concepts

Entity / concept
ALS / Amyotrophic Lateral Sclerosis
Nuclear Pore Complex, NPC
Nup98, Nup62, possibly other FG-Nups
phosphorylation, O-GlcNAcylation
TDP-43 / TARDBP
nucleocytoplasmic transport, nuclear import/export
cellular stress, motor neuron vulnerability
TDP-43 retention, mislocalization, transport defect, degeneration

Evidence ID	Atomic claim	Source branch
E1	ALS models show nucleocytoplasmic transport defects.	External
E2	TDP-43 mislocalization is associated with ALS pathology.	External
E3	Nup98/Nup62 are FG-nucleoporins involved in NPC transport selec tivity.	External
E4	Cellular stress can induce PTMs on nuclear transport-related protei ns.	External
E5	Prior in-house stress assay shows altered nuclear/cytoplasmic TDP -43 ratio under condition X.	Inhouse-DB
M1	Human guide asks system to prioritize phosphorylation rather than therapeutic intervention.	Meta

HG **Explorer** generate hypotheses, writes retrieval specifications, partitions returned evidence into diverse subsets, assigns distinct reasoning modes to each subset, and produces the resulting hypotheses as nodes in the tree.

Root H0: NPC phosphorylation may contribute to ALS-relevant nucleocytoplasmic transport defects.

Node	Search Intent	Query Spec	Hypothesis claim	Reasoning Mode	Router Action
H1	broad exploration	ALS + NPC + phosphorylation, wide scope, rare tilt	Stress-induced phosphorylation or O-GlcNAcylation of Nup98/Nup62 increases TDP-43 residence at the NPC, causing nucleocytoplasmic transport defects in ALS motor neurons.	causal mechanism	expand
H2	mainstream grounding	ALS + TDP-43 + nucleocytoplasmic transport, narrow/mainstream	NPC phosphorylation disrupts importin/exportin selectivity, indirectly promoting TDP-43 mislocalization rather than direct TDP-43 trapping.	mechanism proposal	expand
H3	mechanism refinement	Nup98/Nup62 + TDP-43 + PTM, narrow	Motor neuron vulnerability arises because long-lived FG -Nups accumulate PTMs under chronic stress, making NPC transport defects cell-type selective.	gap-filling	expand
H4	novelty expansion	rare co-occurring entities around NP C PTM + motor neuron stress	Reported Nup–TDP-43 association may be a stress-gran ule or fractionation artifact rather than a causal NPC mechanism.	gap-filling + analogical	expand / graduate
H5	broad exploration	non-Nup NPC transport regulators + phosphorylation + ALS, wide scope	Phosphorylation of a non-Nup NPC-associated transport regulator, rather than Nup98/Nup62 themselves, is the true proximal ALS mechanism.	analogical	expand

State Estimator evaluates each child hypothesis node through a grounded two-level assessment.

Sub-1 Generation of a verification-oriented query

: Retrieve evidence containing all or most of: ALS, TDP-43, Nup98/Nup62, NPC, phosphorylation/O-GlcNAcylation, nucleocytoplasmic transport, motor neurons.

Sub-2 Hypothesis is decomposed into semantic units, and each unit is aligned against the retrieved evidence

Semantic unit	Grounding map
U1: cellular stress induces PTMs on Nup98/Nup62	weakly supported (general stress-PTM link exists, Nup-specific evidence sparse)
U2: Nup98/Nup62 interact directly or functionally with TDP-43	weakly supported (no direct interaction data)
U3: altered PTM changes TDP-43 retention at NPC	reasoning gap (key mechanistic link undemonstrated)
U4: TDP-43 retention disrupts nucleocytoplasmic transport	weakly supported (plausible from related findings, no direct validation)
U5: NCT defect contributes to ALS pathology	well-grounded
U6: motor neurons are selectively vulnerable to this mechanism	unsupported (reasoning gap)

Figure 9. Worked example applying our framework to the ALS research goal from the AI co-scientist paper. Top: simplified pipeline flow from seed hypothesis generation to the second cycle. Bottom panels detail core modules: the Context Optimizer and two cognitive modules of the Hypothesis Generator.