

HALO: A Physics-Aware LLM Agent Framework for Nanophotonic Design

Yubo Zhang^{1*}, Jinlin Xiang^{1*}, Zijun Zhao¹, Yang Zhao¹, Eli Shlizerman¹, Arka Majumdar¹

¹ Department of Electrical & Computer Engineering, University of Washington, Seattle, WA, USA

Abstract

Language models have recently been applied to nanophotonic design, including device generation, optimization, and tool-assisted simulation. However, it remains less clear whether such systems can translate optical objectives into complete simulation-ready designs, execute electromagnetic analysis, and revise decisions from numerical feedback, or how these capabilities should be evaluated reliably across diverse tasks. To address these issues, we introduce HALO, a physics-aware framework that couples language-model planners with a typed, machine-readable design specification, electromagnetic simulation, diagnostic evaluation, and optional reuse of prior failure trajectories in an iterative design loop. We further introduce HALO-Bench, a 52-task benchmark spanning lab-derived, paper-derived, and open-ended nanophotonic design tasks, to evaluate executable design performance under a shared protocol. To study how workflow structure and execution freedom affect scientific agent performance, we compare three planner configurations: a Fixed Structured Workflow, an Autonomous Structured Agent that retains the same simulation interface, and an Autonomous Coding Agent that directly writes and executes simulation code. We find that the Fixed Structured Workflow is consistently the most token-efficient and incurs no observed code- or path-level failures, whereas autonomous coding can achieve higher task success with stronger models at the cost of additional operational failures. Because iterative design may repeatedly encounter similar failure modes, we also study whether prior failed trajectories can be reused to accelerate correction. On targeted cases requiring multiple feedback rounds, retrieved failure feedback reduces the number of iterations to first success and total token use. These findings clarify when scientific agents benefit from explicit interfaces, autonomous execution, and reusable design experience.

Introduction

Language-model systems increasingly plan, use tools, execute code, and revise from feedback (Yao et al. 2022; Schick et al. 2023; Wu et al. 2024). In scientific design, however, these general capabilities are not sufficient: the system needs to translate an objective into an executable representation, run the relevant tools, interpret numerical evidence, and improve within limited evaluation budgets (Chen et al. 2025; Starace et al. 2025).

Nanophotonic design makes this challenge concrete. A short optical goal or device description may omit the ge-

ometry, material assignments, illumination settings, solver configuration, design variables, and evaluation metrics required for electromagnetic simulation. Classical inverse-design methods are powerful after this representation and objective have been specified (Molesky et al. 2018), but the upstream conversion from natural-language intent to a simulation-ready problem remains a substantial bottleneck.

We introduce **HALO**, a physics-aware executable framework that standardizes the conversion from natural-language design intent into a simulation-ready formulation and supports electromagnetic execution, diagnostic evaluation, iterative revision, and optional reuse of prior failed trajectories. In this work, physics awareness means that planner decisions are evaluated and revised through electromagnetic simulation and task-specific physical metrics. For auditability, HALO stores prompts, structured designs or generated code, simulation outputs, scores, evaluator notes, and failure records as design trajectories.

The associated **HALO-Bench** evaluates this process across *Lab-Derived Simulation Tasks*, *Paper-Derived Tasks*, and *Open-Ended Tasks*. These regimes respectively test whether planners can recover known feasible designs, reconstruct simulation-ready devices from paper descriptions, and improve open-ended designs through iterative feedback.

Our central question is how much workflow and interface structure a scientific planner needs. We compare three planner configurations along two dimensions: workflow orchestration, ranging from fixed execution to autonomous agent control, and simulation access, ranging from a typed machine-readable interface to self-generated simulation code. We find that explicit scientific interfaces improve reliability and token efficiency while providing directly inspectable design records, whereas autonomous coding can achieve higher success with sufficiently capable models. Because some cases require multiple feedback rounds, we also examine whether retrieved failure feedback can accelerate convergence.

Our contributions are:

- **HALO**, a physics-aware framework for executable and iterative nanophotonic-agent design and evaluation.
- **HALO-Bench**, a 52-task benchmark spanning Lab-Derived Simulation, Paper-Derived, and Open-Ended nanophotonic design tasks.

*These authors contributed equally.

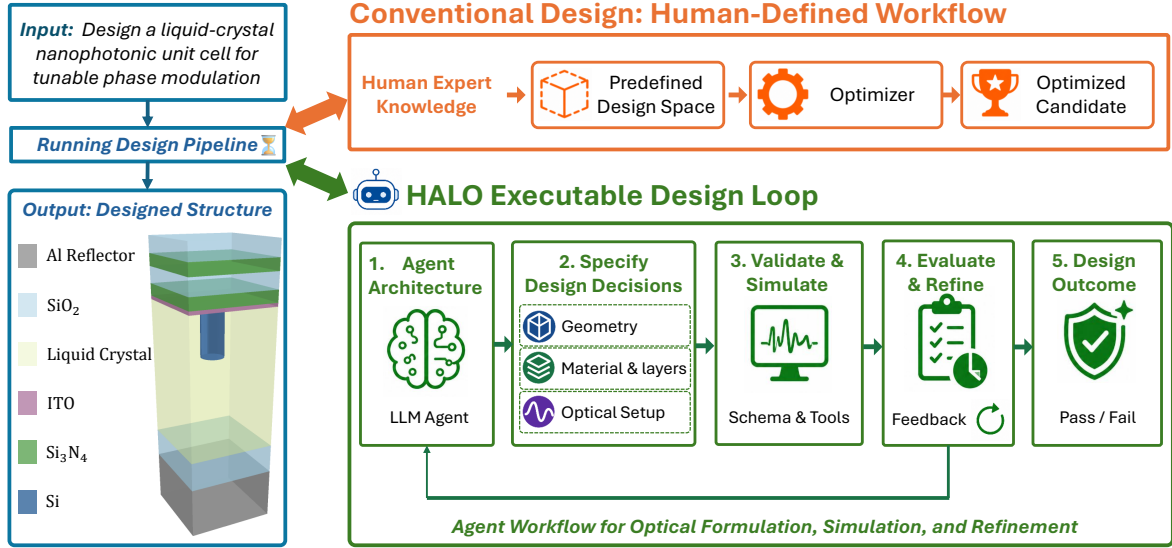


Figure 1: Overview of the HALO problem formulation. HALO converts an underspecified natural-language optical objective into explicit geometry, material assignments, simulation settings, and evaluation criteria, and iteratively revises the design using simulator feedback.

- A system-level comparison of planner structure and autonomy, together with a targeted study showing that schema-assisted retrieval of failed trajectories accelerates convergence on delayed-success cases.

Related Work

Nanophotonic Inverse Design

Nanophotonic inverse design includes gradient-based optimization, topology optimization, evolutionary search, parameter sweeps, and fabrication-aware numerical methods (Molesky et al. 2018; Jensen and Sigmund 2011; Christiansen and Sigmund 2021; Piggott et al. 2015, 2017; Hughes et al. 2018; Elsayy et al. 2020). Machine-learning approaches learn forward models, inverse mappings, generative representations, and surrogate-assisted search strategies (Liu et al. 2018; Deng et al. 2021; Jiang and Fan 2019; Wiecha et al. 2021; Ma et al. 2021, 2022; Naseri and Hum 2021; Hen et al. 2025). These methods are effective once the device representation, design variables, simulator, and objective have been specified. HALO addresses a complementary problem: how a language-model planner formulates this simulation-ready design problem from text and revises it using electromagnetic feedback.

LLM Agents for Scientific and Optical Design

LLM agents support planning, tool use, code execution, and iterative problem solving (Yao et al. 2022; Schick et al. 2023; Wu et al. 2024; Shinn et al. 2023; Hong et al. 2024; Yang et al. 2024). Scientific-agent systems increasingly connect language models with experiments, data analysis, simulation, and domain-specific software (Chen et al. 2025; Bran et al. 2023; Boiko et al. 2023; Lu et al. 2024; Cheng, Zhang, and He 2025). In optics and photonics, language models

have been applied to multilayer design, nanophotonic structure generation, metasurface optimization, and tool-assisted simulation (Ma, Wang, and Guo 2024; Kim, Park, and Shin 2025; Lupoiu et al. 2025; Kharel et al. 2026; Huang et al. 2025; Lu, Malof, and Padilla 2025; Geng et al. 2026; Zhang et al. 2025).

Table 1 compares representative systems by the role of the language model, execution control, explicit geometry and material representation, trajectory memory, and benchmark-based evaluation. HALO evaluates fixed and autonomous planner configurations within a shared executable design framework, while retaining complete trajectories for failure analysis. It also studies whether retrieving prior failed trajectories can accelerate convergence in later design attempts.

Structured Spatial and Physical Reasoning

Executable optical design requires jointly consistent decisions: components must be arranged spatially, materials assigned to the intended regions, solver conditions specified, and design variables connected to measurable objectives. Spatial-reasoning benchmarks show that language models can struggle to preserve geometric consistency in structured representations (Rodionov et al. 2025; Luo et al. 2025). Related materials-science work studies domain language models, scientific agents, and knowledge representations that connect text, material information, and computational tools (Gupta et al. 2022; Song, Miret, and Liu 2023; Venugopal and Olivetti 2024; Statt et al. 2023; Ye et al. 2024; Zhou et al. 2025; Xia et al. 2025).

These capabilities are often evaluated separately. HALO-Bench instead tests how layout, material assignment, solver configuration, and design variables interact within one executable workflow. Its structured representation makes these decisions directly inspectable and supports field-level eval-

System	System Architecture				Geometry Grounding			Material Grounding			
	LM Role	Control	Mem.		Dimensionality	Layout	Valid.	Rep.	Bind.	Valid.	Bench.
OptoGPT (Ma, Wang, and Guo 2024)	inverse model	single-pass	×		1D stack	✓	×	✓	✓	×	×
Chat-to-Chip (Zhang et al. 2025)	design generator	single-pass	×		2D planar	✓	✓	×	×	×	×
OPTIAGENT (Geng et al. 2026)	learned policy	optimizer-in-loop	×		2D lens	✓	✓	✓	✓	✓	×
MetaChat (Lupoiu et al. 2025)	multi-agent system	autonomous tool loop	×		2D freeform	✓	×	✓	✓	×	×
Metamaterial Agent (Lu, Malof, and Padilla 2025)	multi-agent system	autonomous coding loop	✓		3D parameterized unit	×	×	×	×	×	×
HALO (ours)	planner configs	fixed + autonomous	✓		3D typed geometry	✓	✓	✓	✓	✓	✓

Table 1: Representative language-model optical-design systems. Mem. indicates support for or explicit study of reusable memory. Rep. denotes an explicit machine-readable representation; Layout and Bind. indicate explicit spatial-layout and material-binding support, respectively; Valid. indicates validation of the corresponding representation or binding. Bench. denotes executable multi-task evaluation under a shared protocol.

uation before simulation feedback. This scope concerns simulation-ready physical consistency rather than general materials discovery.

Method

HALO Workflow

HALO is an executable framework that standardizes iterative nanophotonic design as a propose–execute–evaluate–revise loop. HALO represents each candidate device using a typed, machine-readable simulation specification that explicitly defines geometry and spatial relations, material assignments, illumination and solver settings, design variables, and evaluation objectives. We refer to this specification as the design schema. Structured proposals are validated before RCWA execution; code-based proposals are executed in a workspace and their outputs are collected for the same evaluation protocol. Figure 2 summarizes the workflow.

HALO distinguishes two forms of history. Within-run trajectory history contains previous iterations of the same case, including planner outputs, schemas or generated code, simulation results, evaluator feedback, and logs; it is available during the default closed-loop experiments. Retrieval memory instead exposes prior design trajectories from a memory pool and is enabled only in the memory ablation.

HALO-Bench

HALO-Bench is the associated benchmark instantiated within HALO. It contains three regimes, shown in Figure 3.

Lab-Derived Simulation Tasks. This setting contains 20 design problems adapted from prior projects in our laboratory. Each task has a known workable reference configuration previously verified using the simulation workflow employed in this study or a compatible workflow. These cases provide

feasible simulation targets, but they should not be interpreted as experimentally fabricated or measured devices.

Paper-Derived Tasks. This setting contains 21 short natural-language design goals generated from published nanophotonic device descriptions through LLM-assisted drafting followed by manual refinement. The prompts describe the relevant structure and optical objective without giving the evaluated planner the complete source paper. Here, recovering a Paper-Derived design means producing a working simulation-compatible design that captures the reported structure and target behavior, not reproducing every experimental curve, fabrication detail, or reported numerical value.

Open-Ended Tasks. This setting contains 11 tasks derived from a selected subset of Paper-Derived Tasks by relaxing selected numerical or structural constraints while preserving the broad physical objective. They require exploration and iterative improvement rather than exact reconstruction. Detailed generation prompts and source-case mappings are provided in the supplementary material.

Planner Configurations

The three planner configurations form a progression of increasing autonomy while sharing the same task inputs, outer-iteration budget, wall-clock budget, and evaluator protocol. Fixed Structured Workflow uses a predefined workflow and HALO’s structured simulation interface. Autonomous Structured Agent adds agentic orchestration and workspace interaction while retaining the same structured simulation interface. Autonomous Coding Agent retains autonomous orchestration but additionally allows the agent to write and execute simulation code directly, removing the predefined interface constraint. Accordingly, Fixed Structured Workflow versus Autonomous Structured Agent provides a practical comparison of fixed versus agentic orchestration under the same

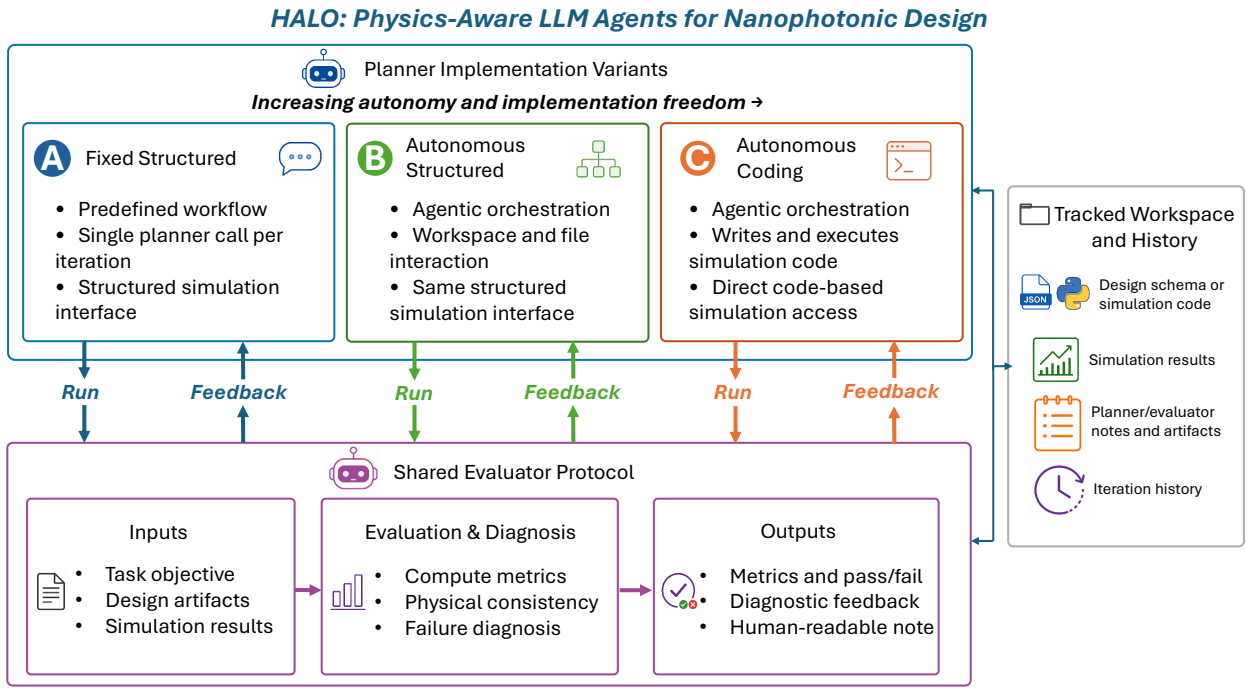


Figure 2: HALO workflow. A planner converts a design goal into either a design schema or executable simulation code, receives simulator-grounded diagnostic feedback, and revises under a fixed outer-iteration budget. HALO logs prompts, artifacts, outputs, scores, notes, and failures.

structured simulation interface, whereas Autonomous Structured Agent versus Autonomous Coding Agent compares structured versus code-based simulation access under the same autonomous workspace.

Fixed Structured Workflow. This configuration is implemented as a fixed LangGraph workflow. The execution path is predefined, with no dynamic tool-selection branch; the only conditional branch occurs at the end of an outer design iteration, where the run stops if the evaluator confirms success and otherwise continues. Each outer design iteration contains one planner LLM call. The planner returns a structured design output requested through prompting and checked afterward with Pydantic validation. The workflow then invokes HALO’s predefined validation and RCWA simulation tools. The planner cannot freely read or write files, change the workflow, write simulator code, or select arbitrary tools.

Autonomous Structured Agent. This configuration is implemented with a LangChain Deep Agent using a local-shell/filesystem workspace. It can perform multistep analysis, read and write files, maintain notes, inspect artifacts, and decide intermediate actions. Its final design representation remains structured, and all formal RCWA simulations must use HALO’s predefined structured simulation interface. It is therefore more autonomous in orchestration than the fixed workflow while retaining the same scientific execution boundary.

Autonomous Coding Agent. This configuration uses the same autonomous workspace but directly writes, executes, inspects, and revises Python simulation code using the available RCWA package. It is not constrained to HALO’s predefined design schema for simulation access. This provides greater implementation freedom, but the physical design decisions are less structurally auditable even though files, code, commands, outputs, and logs are retained. It also introduces code, API, path, artifact, and output-format failure modes.

Common Evaluation Protocol

All planner configurations follow a shared evaluator-generation protocol with the same task objective, metric requirements, execution environment, and pass/fail output contract. Because schema-based and code-based planners produce heterogeneous artifacts, the evaluator inspects the available design artifact and simulation output and generates a case-specific executable evaluation script. The generated script converts saved outputs into numerical metrics and a score/pass decision, and is then fixed and reused across subsequent iterations of that run. The evaluator then generates a human-readable diagnostic note that summarizes the numerical result and explains the reason for success or failure; these notes were independently sanity-checked by additional optics experts for physical and logical consistency with the corresponding design objectives and numerical results. Because evaluator generation uses the same underlying model as the corresponding planner, the reported results should be interpreted as system-level comparisons of complete model–

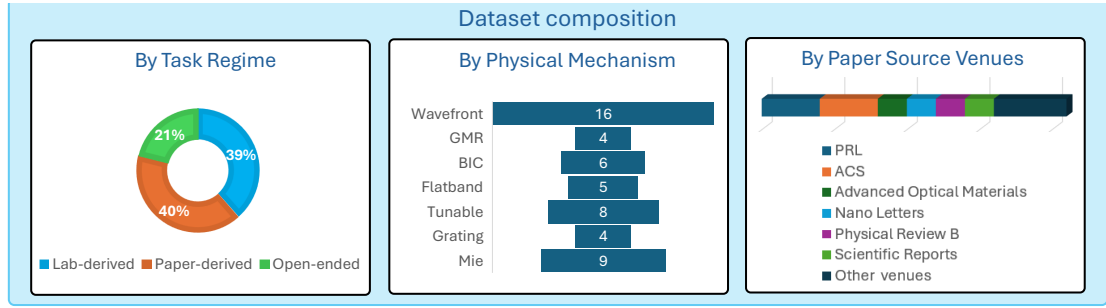


Figure 3: Composition of HALO-Bench. The benchmark contains 20 Lab-Derived, 21 Paper-Derived, and 11 Open-Ended tasks spanning seven nanophotonic mechanisms and multiple publication venues.

configuration pipelines under a shared evaluator-generation protocol, rather than as isolated measurements of planner quality.

Optional Retrieval Memory

Retrieval memory stores prior failed trajectories that may contain useful geometry choices, parameter ranges, evaluation signals, and recovery strategies. The memory ablation compares no retrieved memory, schema-assisted retrieval, and a same-task failure-memory control. Unlike within-run history, retrieval memory can introduce information from prior cases and is analyzed separately from the default closed-loop success comparison.

Experiments

Our experiments ask: (i) whether models can produce an initially plausible structured design before feedback; (ii) how planner structure affects closed-loop success and token efficiency; (iii) whether retrieved failed trajectories accelerate delayed-success cases; and (iv) where each configuration fails.

Experimental Setup

HALO-Bench contains 52 tasks: 20 Lab-Derived Simulation Tasks, 21 Paper-Derived Tasks, and 11 Open-Ended Tasks. Closed-loop runs compare the Fixed Structured Workflow, Autonomous Structured Agent, and Autonomous Coding Agent. Each model-configuration-case is run once with the same task prompt, task-specific objective, common evaluator protocol, at most five outer design iterations, and a one-hour wall-clock limit. Runs stop early after confirmed success. The one-hour limit prevents individual cases from blocking the full benchmark and defines a practical bounded evaluation setting.

Experiments were run on one workstation with an NVIDIA RTX 6000 GPU. For each model-configuration pair, the planner and evaluator use the same underlying language model. We record planner and evaluator token usage separately.

Because the three planner configurations perform different amounts of tool use, code execution, and debugging, their internal workloads are not directly comparable. We therefore report successful cases per million recorded language-model

Model	Layout	Numeric	Material	Sweep	Overall
Gemini 2.5 Flash	95.9	90.6	92.9	93.3	93.2
Gemini 2.5 Pro	96.9	97.1	97.9	96.8	97.2
Gemini 3.1 Pro Preview	97.6	93.2	92.7	97.7	95.3
Claude Haiku 4.5	94.2	94.7	97.1	97.3	95.8
Claude Sonnet 4.6	97.4	91.6	94.4	97.7	95.3
Claude Opus 4.8	99.6	94.6	98.3	98.3	97.7

Table 2: First-step structured design accuracy (%) on perturbed Lab-Derived Simulation Tasks. Each model generates a schema without simulation feedback or iterative revision. Predictions are compared against human-annotated reference schemas using a fixed evaluation procedure. Results are averaged over 80 prompt-schema pairs per model. Metric definitions and scoring details are provided in the supplementary material. Overall is the unweighted mean of the four reported dimensions. Bold values indicate the best result within each model family.

tokens as a measure of token efficiency. This metric is most reliable for comparisons within the same model family, since token accounting differs across providers. Each case is run once because of the scale and computational cost of the benchmark.

Structured Design Understanding

Closed-loop performance combines prompt understanding, artifact construction, simulation, feedback interpretation, and tool execution. We isolate first-step design formulation using the 20 Lab-Derived Simulation Tasks, which provide paired natural-language descriptions and human-annotated, simulation-ready reference schemas. Evaluating only the original prompts would be insufficiently diagnostic because they already describe known feasible configurations. We therefore construct controlled prompt-schema perturbations by modifying selected requirements in the natural-language description together with their corresponding fields in the reference schema. Numerical values and supported material or categorical substitutions are updated synchronously, preserving a consistent input-reference pair while testing whether the model follows the modified requirement rather than reproducing the original design.

The model receives only the input prompt for each con-

Model	Configuration	Lab (%)	Paper (%)	Open (%)	Macro avg. (%)	Plan tok.	Eval tok.	Total tok.	Succ./M tok. $\uparrow$
Gemini 2.5 Flash	Fixed Structured Workflow	90.0	23.8	36.4	50.1	49k	573k	622k	0.835
	Autonomous Structured Agent	30.0	4.8	0.0	11.6	1.49M	308k	1.80M	0.075
	Autonomous Coding Agent	55.0	23.8	18.2	32.3	1.28M	227k	1.50M	0.231
Gemini 2.5 Pro	Fixed Structured Workflow	75.0	42.9	63.6	60.5	52k	879k	932k	0.640
	Autonomous Structured Agent	50.0	42.9	18.2	37.0	1.76M	587k	2.35M	0.172
	Autonomous Coding Agent	55.0	19.0	27.3	33.8	3.53M	235k	3.76M	0.092
Gemini 3.1 Pro Preview	Fixed Structured Workflow	85.0	47.6	63.6	65.4	59k	773k	833k	0.785
	Autonomous Structured Agent	90.0	52.4	54.5	65.6	2.01M	406k	2.42M	0.278
	Autonomous Coding Agent	95.0	47.6	63.6	68.8	1.44M	263k	1.71M	0.405
Claude Sonnet 4.6	Fixed Structured Workflow	95.0	66.7	45.5	69.1	22k	292k	314k	2.327
	Autonomous Structured Agent	95.0	57.1	54.5	68.9	1.17M	328k	1.50M	0.474
	Autonomous Coding Agent	90.0	85.7	63.6	79.8	1.18M	181k	1.37M	0.604
Claude Haiku 4.5	Fixed Structured Workflow	75.0	47.6	45.5	56.0	32k	981k	1.01M	0.571
	Autonomous Structured Agent	75.0	38.1	36.4	49.8	749k	1.16M	1.91M	0.272
	Autonomous Coding Agent	80.0	47.6	36.4	54.7	1.85M	471k	2.32M	0.249

Table 3: Closed-loop success and token use. Lab, Paper, and Open denote Lab-Derived Simulation, Paper-Derived, and Open-Ended Tasks. Macro avg. is the simple average of the Lab, Paper, and Open success rates. Plan, Eval, and Total tok. report mean per-run token counts over runs. Succ./M tok. is the number of successful runs per million aggregate recorded tokens. Bold marks the best value within each model block.

Memory condition	Succ.@1	Mean iter. $\downarrow$	Median tokens (M) $[Q_1, Q_3] \downarrow$	Aggregate tokens (M) $\downarrow$	Succ./M tokens $\uparrow$
No Retrieved Memory	0/16	2.44	1.67 [1.09, 2.88]	41.38	0.387
Schema-Assisted Top-3 Retrieval	10/16	1.44	0.34 [0.20, 1.17]	14.79 (−64.3%)	1.082
Same-Task Top-1 Control	8/16	1.75	0.47 [0.26, 1.81]	20.71 (−50.0%)	0.773

Table 4: Targeted trajectory-memory ablation on 16 baseline-delayed success cases. The subset is selected such that the No Retrieved Memory condition succeeds after, but not at, the first iteration; therefore, its Success@1 is zero by construction. All conditions eventually succeed; metrics report convergence speed and language-model token efficiency. The same-task Top-1 control provides one prior failed trajectory from the corresponding task.

dition, while the paired schema is used solely as the evaluation reference. Each task is evaluated under the original prompt and three controlled perturbation settings, yielding 80 prompt–schema pairs per model. Exact perturbation rules, conditions, and sample counts are provided in the supplementary material.

Before receiving validation, simulation results, or iterative feedback, the model generates a design schema, which is then evaluated against the paired human-annotated reference using the same fixed procedure. We evaluate layout representation, numerical binding, material assignment, and sweep-variable specification. Detailed metric definitions and scoring rules are provided in the supplementary material. We restrict this field-level comparison to the Lab-Derived tasks because the Paper-Derived and Open-Ended tasks may admit multiple simulation-compatible formulations rather than a unique reference schema.

Table 2 shows more than 93% overall accuracy for all evaluated models on the perturbed Lab-Derived tasks. Thus, closed-loop failures cannot be explained solely by an inability to produce an initially plausible structured design. Reliability also depends on preserving, executing, and revising these decisions from simulation feedback.

Closed-Loop Success and Efficiency

Table 3 reports complete propose–execute–evaluate–revise trajectories over all three HALO-Bench regimes.

Observed success varies with both the underlying model and planner implementation. The highest observed macro-average success is 79.8%, achieved by the Autonomous Coding Agent with Claude Sonnet 4.6. Autonomous Coding Agent also improves substantially across Gemini models, from 32.3% with Gemini 2.5 Flash to 68.8% with Gemini 3.1 Pro Preview, consistent with stronger models being better able to exploit code-generation freedom.

Autonomy is not uniformly beneficial. With Gemini 2.5 Flash and Gemini 2.5 Pro, the Fixed Structured Workflow achieves higher macro-average success than either autonomous configuration. The Autonomous Structured Agent also does not consistently outperform the fixed workflow. Across every model block, the Fixed Structured Workflow achieves the highest number of successful cases per million recorded tokens; with Claude Sonnet 4.6, it achieves 2.327 successes per million tokens, compared with 0.474 for Autonomous Structured Agent and 0.604 for Autonomous Coding Agent. These comparisons apply under five outer design iterations, a one-hour wall-clock limit, and non-equalized

internal tool actions.

Memory Retrieval Ablation

Cases that succeed only after multiple feedback rounds motivate a targeted study of whether retrieved failure history can accelerate convergence. We first examine how to retrieve relevant failed trajectories from memory, comparing text-only retrieval with a schema-assisted alternative. Because text-only matching may miss physically related trajectories when descriptions differ in wording or order, our schema-assisted method augments text similarity with schema-level similarity over geometry, material assignments, optical setup, and sweep variables. The text- and schema-based similarity scores are combined into a final ranking score, and the three highest-ranked failed trajectories are retrieved. Detailed retrieval design, scoring, and the comparison with text-only retrieval are provided in the supplementary material.

We conduct the ablation on 16 cases in which the No Retrieved Memory baseline eventually succeeded but required more than one iteration. Using the same Gemini 3.1 Pro Preview Autonomous Structured Agent, we compare No Retrieved Memory, Schema-Assisted Top-3 Retrieval, and Same-Task Failure Memory (Top-1 Control). The retrieval pool is constructed from failed trajectories produced by prior Gemini 3.1 Pro Preview runs; successful trajectories and success-revealing artifacts are excluded. We allow task-level overlap because the experiment evaluates convergence acceleration rather than cross-task memory generalization. Accordingly, this ablation evaluates retrieval-assisted reuse within an available failure-history pool, rather than transfer to previously unseen tasks. Schema-Assisted Top-3 ranks all eligible failed trajectories using the retrieval score. The Same-Task Top-1 control bypasses retrieval and directly provides one failed trajectory from the corresponding task.

Among the 16 baseline-delayed-success cases, Schema-Assisted Top-3 Retrieval converts 10 cases from multi-iteration to first-iteration success. It reduces the mean first-success iteration from 2.44 to 1.44, lowers the median per-case token usage from 1.67M to 0.34M, and reduces aggregate token use from 41.38M to 14.79M, a 64.3% reduction. The Same-Task Top-1 Control also accelerates convergence, reducing the mean first-success iteration to 1.75 and aggregate token use to 20.71M, a 50.0% reduction relative to No Retrieved Memory. This condition serves as a diagnostic control because it directly provides a failed trajectory from the corresponding task; it should not be interpreted as either a realistic first-time deployment setting or a strict upper bound.

Failure Analysis

Each failed case is assigned one dominant category using a consistent annotation rule based on the complete trajectory and final artifacts; category definitions and representative examples are provided in the supplementary material. Successful cases are excluded; if a later valid iteration succeeds, earlier failed iterations do not count as a case failure.

Table 5 aggregates 100 Fixed Structured Workflow failures, 133 Autonomous Structured Agent failures, and 115 Autonomous Coding Agent failures over all five evaluated models. Fixed Structured Workflow failures occur mainly at

Failure category	Fixed Structured Workflow	Autonomous Structured Agent	Autonomous Coding Agent
Timeout / incomplete evaluation	29 (29.0%)	34 (25.6%)	33 (28.7%)
Generated artifact / path error	0 (0.0%)	1 (0.8%)	21 (18.3%)
Code / tool / API error	0 (0.0%)	0 (0.0%)	29 (25.2%)
Simulator / runtime error	34 (34.0%)	16 (12.0%)	13 (11.3%)
Missing / invalid result	4 (4.0%)	38 (28.6%)	13 (11.3%)
Wrong setup / sweep	9 (9.0%)	25 (18.8%)	0 (0.0%)
Target not met	24 (24.0%)	19 (14.3%)	6 (5.2%)
Total failures	100	133	115

Table 5: Failure composition aggregated over all five evaluated models. Each entry reports the number of failed cases, with percentages computed within the failed cases of each planner configuration. Successful cases are excluded, and each failed case is assigned one dominant category. Labels describe the dominant terminal failure of each case rather than every error observed during its trajectory.

simulator/evaluation or scientific-target stages and show no code/path errors. Autonomous Structured Agent has more missing or invalid results and setup/sweep-management errors. Autonomous Coding Agent adds code/tool/API and artifact/path failures. This operational burden does not make Autonomous Coding Agent uniformly worse: it achieves the highest observed success with a strong model, but pays a larger software and artifact-management cost.

Discussion and Conclusion

HALO frames nanophotonic-agent design as an executable loop from natural-language intent to simulation-ready formulation, simulator execution, diagnostic evaluation, revision, and optional reuse of prior failures. Across HALO-Bench, the Fixed Structured Workflow is the most token-efficient, avoids the code- and path-level failures observed in Autonomous Coding, and produces directly inspectable design records. Autonomous Coding can achieve the highest success with a strong model, but incurs additional software, artifact, and output-format failures. The Autonomous Structured Agent further suggests that added orchestration is not uniformly beneficial when the structured simulation boundary is retained.

Memory provides a second lever. On cases solved by the baseline only after multiple iterations, retrieved failure feedback reduced both convergence time and total token use, helping already-solvable cases converge more efficiently.

Limitations. HALO-Bench primarily covers periodic nanophotonic structures, and each model-configuration-case is evaluated once. The memory study targets baseline-delayed-success cases and measures convergence rather than benchmark-wide success gains. Final designs may also require application-specific, higher-fidelity validation.

References

Boiko, D. A.; MacKnight, R.; Kline, B.; and Gomes, G. 2023. Autonomous chemical research with large language models.

Nature, 624(7992): 570–578.

Bran, A. M.; Cox, S.; Schilter, O.; Baldassari, C.; White, A. D.; and Schwaller, P. 2023. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*.

Chen, Z.; Chen, S.; Ning, Y.; Zhang, Q.; Wang, B.; Yu, B.; Li, Y.; Liao, Z.; Wei, C.; Lu, Z.; et al. 2025. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, volume 2025, 96934–96990.

Cheng, A.; Zhang, L.; and He, G. 2025. Re4: Scientific Computing Agent with Rewriting, Resolution, Review and Revision. *arXiv preprint arXiv:2508.20729*.

Christiansen, R. E.; and Sigmund, O. 2021. Inverse design in photonics by topology optimization: tutorial. *Journal of the Optical Society of America B*, 38(2): 496–509.

Deng, Y.; Ren, S.; Fan, K.; Malof, J. M.; and Padilla, W. J. 2021. Neural-adjoint method for the inverse design of all-dielectric metasurfaces. *Optics Express*, 29(5): 7526–7534.

Elsawy, M. M.; Lanteri, S.; Duvigneau, R.; Fan, J. A.; and Genevet, P. 2020. Numerical optimization methods for metasurfaces. *Laser & Photonics Reviews*, 14(10): 1900445.

Geng, Y.; Sun, L.; Gao, Y.; Hu, X.; Yi, Z.; Qian, X.; Hu, W.; Bai, J.; and Wang, K. 2026. OPTIAGENT: A Physics-Driven Agentic Framework for Automated Optical Design. *arXiv preprint arXiv:2602.23761*.

Gupta, T.; Zaki, M.; Krishnan, N. A.; and Mausam. 2022. MatSciBERT: A materials domain language model for text mining and information extraction. *npj Computational Materials*, 8(1): 102.

Hen, L.; Yosef, E.; Raviv, D.; Giryes, R.; and Scheuer, J. 2025. Inverse design of diffractive metasurfaces using diffusion models. *ACS Photonics*, 13(1): 38–46.

Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Yau, S.; Lin, Z.; Zhou, L.; et al. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, 23247–23275.

Huang, Y.; Zheng, B.; Dong, Y.; Tang, H.; Zhao, H.; Shawon, R. H.; An, S.; and Zhang, H. 2025. MCP-enabled LLM for meta-optics inverse design: leveraging differentiable solver without LLM expertise. *Nanophotonics*, 14(27): 5589–5602.

Hughes, T. W.; Minkov, M.; Williamson, I. A.; and Fan, S. 2018. Adjoint method and inverse design for nonlinear nanophotonic devices. *ACS Photonics*, 5(12): 4781–4787.

Jensen, J. S.; and Sigmund, O. 2011. Topology optimization for nano-photonics. *Laser & Photonics Reviews*, 5(2): 308–321.

Jiang, J.; and Fan, J. A. 2019. Global optimization of dielectric metasurfaces using a physics-driven neural network. *Nano letters*, 19(8): 5366–5372.

Kharel, P.; Khavasi, A.; Chen, X.; and Hughes, T. W. 2026. Autonomous agentic design for photonics. *arXiv preprint arXiv:2606.00915*.

Kim, M.; Park, H.; and Shin, J. 2025. Nanophotonic device design based on large language models: multilayer and metasurface examples. *Nanophotonics*, 14(8): 1273–1282.

Liu, D.; Tan, Y.; Khoram, E.; and Yu, Z. 2018. Training deep neural networks for the inverse design of nanophotonic structures. *ACS Photonics*, 5(4): 1365–1369.

Lu, C.; Lu, C.; Lange, R. T.; Foerster, J.; Clune, J.; and Ha, D. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Lu, D.; Malof, J. M.; and Padilla, W. J. 2025. An agentic framework for autonomous metamaterial modeling and inverse design. *ACS Photonics*, 12(11): 6071–6080.

Luo, S.; Zhu, Z.; Yuan, Y.; Yang, Y.; Shan, L.; and Wu, Y. 2025. Geogrambench: Benchmarking the geometric program reasoning in modern llms. *arXiv preprint arXiv:2505.17653*.

Lupoiu, R.; Shao, Y.; Dai, T.; Mao, C.; Edée, K.; and Fan, J. A. 2025. A multi-agentic framework for real-time, autonomous freeform metasurface design. *Science Advances*, 11(44): eadx8006.

Ma, T.; Tobah, M.; Wang, H.; Guo, L. J.; et al. 2022. Benchmarking deep learning-based models on nanophotonic inverse design problems. *Opto-Electron. Sci*, 1(1): 210012.

Ma, T.; Wang, H.; and Guo, L. J. 2024. OptoGPT: a foundation model for inverse design in optical multilayer thin film structures. *Opto-Electronic Advances*, 7(7).

Ma, W.; Liu, Z.; Kudyshev, Z. A.; Boltasseva, A.; Cai, W.; and Liu, Y. 2021. Deep learning for the design of photonic structures. *Nature photonics*, 15(2).

Molesky, S.; Lin, Z.; Piggott, A. Y.; Jin, W.; Vucković, J.; and Rodriguez, A. W. 2018. Inverse design in nanophotonics. *Nature photonics*, 12(11): 659–670.

Naseri, P.; and Hum, S. V. 2021. A generative machine learning-based approach for inverse design of multilayer metasurfaces. *IEEE Transactions on Antennas and Propagation*, 69(9): 5725–5739.

Piggott, A. Y.; Lu, J.; Lagoudakis, K. G.; Petykiewicz, J.; Babinec, T. M.; and Vučković, J. 2015. Inverse design and demonstration of a compact and broadband on-chip wavelength demultiplexer. *Nature photonics*, 9(6): 374–377.

Piggott, A. Y.; Petykiewicz, J.; Su, L.; and Vučković, J. 2017. Fabrication-constrained nanophotonic inverse design. *Scientific reports*, 7(1): 1786.

Rodionov, F.; Eldesokey, A.; Birsak, M.; Femiani, J.; Ghanem, B.; and Wonka, P. 2025. Floorplanqa: A benchmark for spatial reasoning in llms using structured representations. *arXiv preprint arXiv:2507.07644*.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36: 68539–68551.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36: 8634–8652.

Song, Y.; Miret, S.; and Liu, B. 2023. MatSci-NLP: Evaluating scientific language models on materials science language tasks using text-to-schema modeling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3621–3639.

Starace, G.; Jaffe, O.; Sherburn, D.; Aung, J.; Chan, J. S.; Maksin, L.; Dias, R.; Mays, E.; Kinsella, B.; Thompson, W.; et al. 2025. PaperBench: Evaluating AI’s Ability to Replicate AI Research. *arXiv preprint arXiv:2504.01848*.

Statt, M. J.; Rohr, B. A.; Guevarra, D.; Breeden, J.; Suram, S. K.; and Gregoire, J. M. 2023. The materials experiment knowledge graph. *Digital Discovery*, 2(4): 909–914.

Venugopal, V.; and Olivetti, E. 2024. MatKG: An autonomously generated knowledge graph in Material Science. *Scientific Data*, 11(1): 217.

Wiecha, P. R.; Arbouet, A.; Girard, C.; and Muskens, O. L. 2021. Deep learning in nano-photonics: inverse design and beyond. *Photonics research*, 9(5): B182–B200.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First conference on language modeling*.

Xia, Z.; Ma, J.; Zheng, C.; Zhang, S.; Li, Y.; Su, H.; Hu, P.; Zhang, C.; Gong, X.; Ouyang, W.; et al. 2025. An Agentic Framework for Autonomous Materials Computation. *arXiv preprint arXiv:2512.19458*.

Yang, J.; Jimenez, C. E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K. R.; and Press, O. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yao, S.; Zhao, J.; Yu, D.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2022. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Ye, Y.; Ren, J.; Wang, S.; Wan, Y.; Razzak, I.; Hoex, B.; Wang, H.; Xie, T.; and Zhang, W. 2024. Construction and application of materials knowledge graph in multidisciplinary materials science via large language model. *Advances in Neural Information Processing Systems*, 37: 56878–56897.

Zhang, H.; Kang, L.; Campbell, S. D.; and Werner, D. H. 2025. Chat to chip: large language model based design of arbitrarily shaped metasurfaces. *Nanophotonics*, 14(22): 3625–3633.

Zhou, L.; Ling, H.; Yan, K.; Zhao, K.; Qian, X.; Arróyave, R.; Qian, X.; and Ji, S. 2025. Toward greater autonomy in materials discovery agents: Unifying planning, physics, and scientists. *arXiv preprint arXiv:2506.05616*.