INTENTSCI-GRAPH: A SCIENTIFIC REASONING LAYER FOR AUTONOMOUS RESEARCH AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous research agents are increasingly capable of executing scientific workflows, but reliable scientific discovery cannot be reduced to execution capability alone. Reliable scientific discovery also depends on whether an agent can reason over an underspecified research intent, organize a scientifically coherent plan, and determine what evidence is needed to support a conclusion. Existing research agents can organize this reasoning through planning and execution workflows, but the resulting organization is often incomplete or insufficiently verified. A method may not be clearly connected to the evidence required for the intended claim, or the final outputs may not be aligned with the criteria used to validate the result. To address this gap, we introduce IntentSci-Graph, a scientific reasoning layer for autonomous research. IntentSci-Graph casts autonomous research as a pre-execution scientific program synthesis problem, transforming research intent into an explicit and verifiable scientific program. Rather than asking a downstream executor to infer scientific structure while acting, IntentSci-Graph produces a structured interface that links research objectives, methods, required evidence, and validation criteria. The program is refined through structured synthesis and verification, making the intended study inspectable before execution and reusable across different downstream agents. On ASI-Bench, IntentSci-Graph effectively improves autonomous research performance, particularly under underspecified research prompts, demonstrating that explicit and verifiable scientific programs can provide a practical foundation for reliable autonomous research.

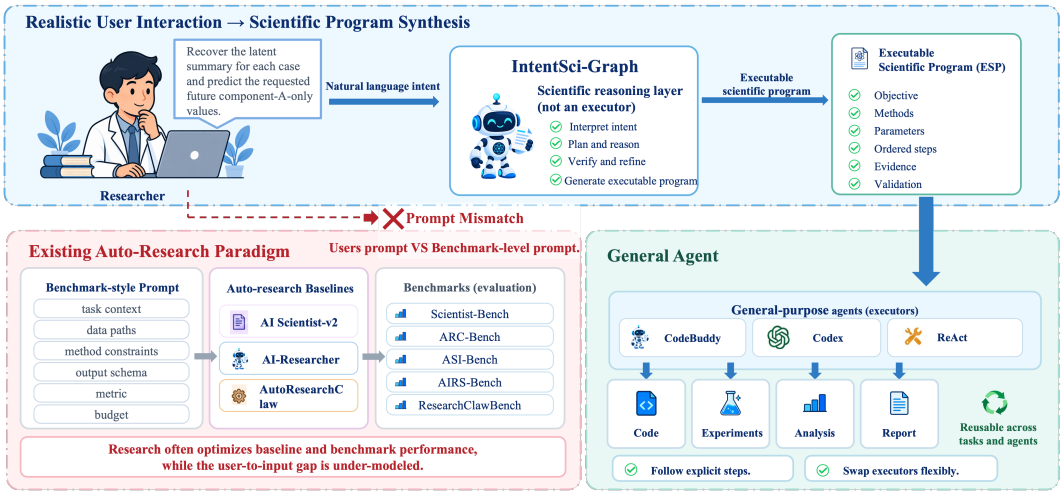


Figure 1: **Scientific program synthesis is the missing layer between realistic user intent and reliable execution.** Realistic scientific requests are often underspecified and do not match benchmark-style prompts. IntentSci-Graph interprets the intent, plans and verifies an executable scientific program, and passes this explicit interface to general-purpose executors.

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

1 INTRODUCTION

Autonomous research agents aim to automate scientific discovery by combining large language models with literature retrieval, code execution, external tools, and long-horizon workflow orchestration (**?**Yang et al., 2024; Hu et al., 2025; Tang et al., 2025; Liu et al., 2026). Given a research intent I, a workspace W, and task constraints C, an agent is expected to organize a workflow involving information gathering, method selection, implementation, experimentation, analysis, and reporting. Existing systems already provide different forms of planning, workflow orchestration, and iterative reflection to structure these activities. However, reliable scientific discovery depends not only on whether a workflow is organized, but also on whether its organization preserves the scientific relationships that connect the research objective, the chosen methodology, the required evidence, and the validation of the final result. These relationships are often distributed across plans, intermediate notes, tool interactions, and execution outputs rather than organized as one coherent scientific structure.

This limitation becomes important when research intents are underspecified and the information needed to interpret them is distributed across the workspace and task constraints. An agent may produce an operationally coherent workflow while failing to connect a selected method to the evidence required for the intended claim, or while generating outputs that are not aligned with the criteria used to validate the result. Existing approaches improve autonomous research through agent orchestration, scientific workflow integration, reflection, and self-healing execution (**?**Hu et al., 2025; Tang et al., 2025; Liu et al., 2026). These approaches improve how agents coordinate actions and respond to execution feedback, but they do not necessarily ensure that the objective, methodology, evidence requirements, and validation criteria are jointly specified and checked before execution. This limitation is reflected in project-level evaluations: ASI-Bench reports a sharp performance decline as methodological guidance is progressively removed (Zhou et al., 2026), while recent end-to-end studies identify plan–execution mismatch, missing scientific cores, unsupported artifacts, and insufficient checking as recurring failure modes (Xu et al., 2026; Zhang et al., 2026; Fei et al., 2026; Brueckner et al., 2026).

The challenge is therefore not to introduce organization into autonomous research, but to improve the quality of that organization. A reliable system should make the relations among the research objective, methodology, evidence, validation criteria, and execution requirements explicit and jointly verifiable before the executor begins acting. Without such a representation, it is difficult to determine whether an unsuccessful outcome results from an inappropriate scientific plan, an incomplete evidence requirement, or an implementation failure. This motivates the following question: *Can a structured and verifiable scientific program improve the organization of research intent before execution and provide a reliable interface for downstream research agents?* Figure 1 summarizes this perspective: scientific program synthesis provides an explicit interface between realistic research intent and reliable execution.

To address this challenge, we introduce INTENTSCI-GRAPH, a scientific reasoning framework for pre-execution scientific program synthesis. Given I, W, and C, INTENTSCI-GRAPH organizes an underspecified research intent into an explicit scientific program that connects the research objective with the intended methodology, required evidence, validation criteria, and execution contract. The program is neither a runnable code fragment nor a reformulated prompt. It is an intermediate scientific representation designed to make the structure of the intended study inspectable and internally consistent before execution. INTENTSCI-GRAPH improves this organization through a modular process consisting of task grounding, program construction, critical review, independent validation, and execution-interface condensation. The resulting program can be passed to different downstream research agents, allowing scientific organization to be improved independently of the executor architecture.

Our contributions are summarized as follows:

- We propose INTENTSCI-GRAPH, a scientific reasoning framework that improves the organization of research intent through pre-execution scientific program synthesis and establishes an explicit interface between research intent and downstream execution.
- We develop a modular scientific program synthesis architecture consisting of task grounding, program construction, critical review, independent validation, and execution-interface

condensation. These modules jointly organize and verify the relationships among scientific objectives, methods, evidence, validation criteria, and execution requirements.

- We conduct controlled experiments on ASI-Bench across progressively underspecified research prompts, multiple executor architectures, and different model settings. The results demonstrate that INTENTSCI-GRAPH effectively improves autonomous research performance, particularly when methodological guidance is limited.

2 RELATED WORK

Autonomous research agents and execution orchestration. Recent work has improved autonomous research through agent orchestration, tool-use interfaces, and end-to-end workflow automation. GPTSwarm and SWE-agent improve how agents coordinate tools and interact with execution environments (Zhuge et al., 2024; Yang et al., 2024), while AI-Researcher and AutoResearchClaw extend these capabilities to literature analysis, experimentation, result verification, and scientific reporting (Tang et al., 2025; Liu et al., 2026). These systems make long-horizon execution more capable, but their main object remains the execution workflow. The relations among research objectives, methods, evidence, and validation are not treated as an independent scientific reasoning interface.

Scientific-agent benchmarks and failure diagnosis. Recent benchmarks evaluate agents on increasingly complex scientific tasks and provide evidence for this limitation. ASI-Bench shows that performance drops as methodological guidance is progressively removed (Zhou et al., 2026). ResearchClawBench and AutoResearchEval further identify protocol mismatch, evidence mismatch, missing scientific cores, and insufficient checking as recurring failure modes (Xu et al., 2026; Fei et al., 2026). These findings suggest that autonomous research failures are not caused only by execution errors; they also reflect incomplete organization and verification of the scientific task before execution. This motivates an explicit representation of the intended scientific structure.

Research intent, workflow, and specification generation. Prior work has explored transforming natural-language intent into structured specifications and workflows. Turning Intent into Specifications studies the generation of executable software specifications (Wang et al., 2026), while From Research Question to Scientific Workflow focuses on converting research questions into structured workflow graphs (Balis et al., 2026). GTA also shows that realistic user requests contain implicit requirements for tool selection and multi-step planning (Wang et al., 2024). These studies demonstrate the value of making intent explicit, but they do not jointly represent scientific objectives, methods, evidence requirements, validation criteria, and execution contracts as one verifiable scientific program. IntentSci-Graph addresses this gap by providing such a scientific reasoning layer before execution.

3 METHOD

3.1 PROBLEM FORMULATION

Let $I = (p, W, C)$ be a research request, where p is a natural-language description ranging from a detailed experimental specification to a colloquial goal, W contains data, code, and metadata, and C specifies output, resource, time, and safety constraints. IntentSci-Graph synthesizes a scientific program and projects it into an execution-facing interface:

$$S = G_{\theta,\xi}(I) = (O, H, M, E, V, X),$$
$$S^{\text{exec}} = \Pi_{\text{exec}}(S) = (T, P, \mathcal{A}, O_{\text{out}}, N).$$

Here θ denotes role prompts, tools, and validation rules, and ξ denotes model and randomness settings. The complete program comprises objectives O, testable hypotheses H, methods, equations, and parameters M, evidence requirements E, validation criteria V, and execution contracts X, including inputs, dependencies, expected outputs, and formats.

The execution view retains a task summary T, global parameters and their values or sources P, ordered steps $\mathcal{A} = \{a_i\}_{i=1}^{K}$, final output contracts O_{out}, and execution-critical notes N. Each step $a_i = (g_i, d_i, m_i, f_i, p_i, y_i, r_i)$ specifies its objective, inputs and dependencies, operation, equations or computational rules, local parameters, generated data or intermediate result, and output

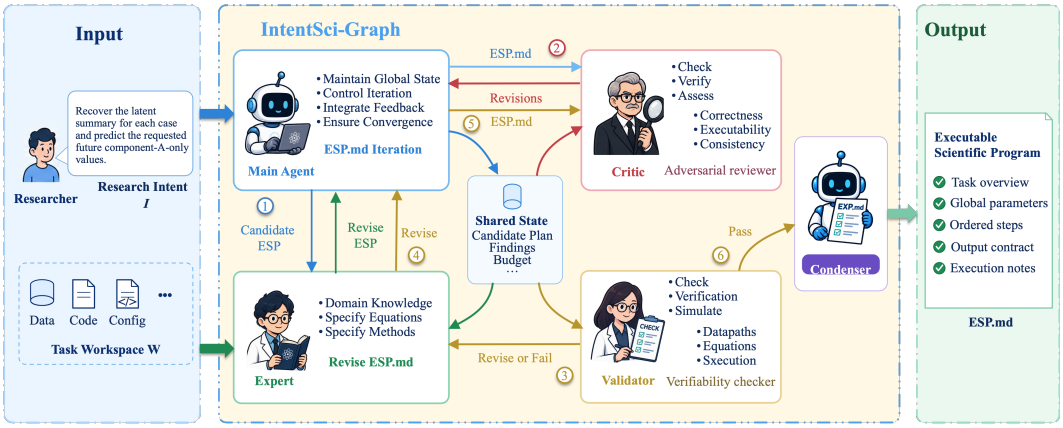


Figure 2: **Overview of IntentSci-Graph's role-playing self-play system.** A natural-language intent and task workspace enter a shared state. The Main Agent, Expert, Critic, Validator, and Condenser exchange plans and structured feedback until an executable scientific program is produced. The image labels the output as ESP.md; in our implementation, the executable program is published as EXP.md.

path/format, respectively. Evidence, review findings, and validation traces support synthesis but are excluded from S^{exec}.

A downstream executor $\mathcal{E}_{a,m}$, with architecture a and model m, reads this interface and the same workspace to produce $Y = \mathcal{E}_{a,m}(S^{\text{exec}}; W)$. The score $R = \text{Score}(Y)$ reflects scientific criteria, evaluation gates, and artifact completeness. This separates program specification from execution and evaluates scientific outcomes rather than textual similarity to a reference prompt.

3.2 Overview: Role-Playing Self-Play for Scientific Program Synthesis

As shown in Figure 2, five roles operate on a shared candidate program: the Main Agent controls the process and integrates feedback, the Expert constructs a domain-conditioned plan, the Critic identifies defects, the Validator independently checks executability, and the Condenser produces the final execution interface. We call this mechanism *self-play-inspired role-based deliberation*: asymmetric roles propose, challenge, and revise one another's decisions through context isolation, structured findings, and bounded iteration, without a symmetric game, competitive rewards, or online parameter updates.

At round t, the shared state is $Z_t = (I, W, F, T, S_t, C_t, V_t, b_t)$, where F contains workspace facts, T is the structured task specification, S_t is the candidate, C_t and V_t are review findings, and b_t is the remaining iteration budget. The Main Agent routes required repairs to the Expert and invokes the Condenser only after validation succeeds. The following subsections detail these roles.

3.3 Main Agent: Shared State and Process Control

The Main Agent is graph-control logic rather than an additional LLM call. Its `bootstrap` stage scans the workspace without assuming a fixed directory layout, recording file paths, types, sizes, hash prefixes, array shapes, dtypes, and lightweight profiles. Task paths are labeled `declared`, `alias`, `missing`, or `ambiguous`. The `interpret` stage produces a `TaskSpec` containing the objective, task type, domain, declared inputs, required/optional outputs, constraints, unknowns, research topics, and questions. Unsupported information remains an unknown rather than a fact.

Starting from an empty S_0, the controller supplies the task specification, workspace facts, and findings to the Expert and coordinates the review loop below. Successful condensation is atomically published as `EXP.md`.

```
state <- bootstrap(I, W); task <- interpret(state)

repeat while expert budget remains:
```

```
    plan <- Expert(task, facts, findings, previous_plan)
    findings <- Critic(task, facts, plan)
    if findings contain blockers or warnings: continue

repeat while revision budget remains:
    if ValidateStructure(plan) fails:
        plan <- Expert(task, facts, findings, plan); continue
    result <- IndependentValidator(task, facts, plan)
    if result requires revision:
        plan <- Expert(task, facts, result.findings, plan); continue

    publish(Condenser(plan), EXP.md); return success

return validation_failed
```

3.4 Expert: Domain Activation and Scientific Plan Construction

Conditioned on the domain, task specification, and current candidate, the Expert produces a complete `PlanDoc` with an objective, workspace, input files, and ordered steps. Each step specifies IDs and dependencies, methods and equations with symbols, units, and boundary conditions, numerical parameters, code responsibilities, dataflow, output contracts, validation criteria, and fallbacks.

After review, the Expert returns a complete revised plan. For example, it repairs a missing symbol in `formulas` or `method`, and a missing intermediate file in the upstream `outputs`, downstream `inputs`, and dependency relation. Revisions thus propagate across affected fields.

3.5 Critic: Adversarial Plan Review

The Critic reads the `TaskSpec`, workspace facts, and current `PlanDoc` without editing it. It checks scientific correctness, required-output coverage, step executability, consistency of IDs, dependencies, inputs, outputs and shapes, and use of missing or protected data. Each localized finding contains a severity, aspect, message, and repair suggestion. Severity is `blocker`, `warning`, or `suggestion`; blockers identify direct execution failures, incorrect results, unavailable required inputs, missing required outputs, or infeasible dependencies. The Main Agent uses these findings to decide whether another Expert round is needed.

3.6 Validator: Deterministic and Independent Verification

The Validator determines whether a candidate can be handed to an executor that has not observed the synthesis process. First, `validate_plan` checks nonempty objectives, methods, and validation fields, unique step IDs, known dependencies, input closure over workspace files and upstream artifacts, required-output closure, placeholder and unsafe-path rules, and acyclicity using Kahn's algorithm. A structural failure is returned to the Expert without invoking the independent LLM Validator.

After structural checks pass, the Validator runs in an independent session that receives only the `TaskSpec`, workspace facts, and current `PlanDoc`. It checks whether files, parameters, and symbols suffice to start each step; whether equations, units, boundary conditions, and numerical precision are coherent; whether outputs flow correctly between steps; whether final filenames, shapes, dtypes, columns, axes, and ordering are explicit; and whether the dependency order permits execution. It returns `pass`, `revise`, or `fail`; revisions are bounded by `max_revisions`, after which synthesis ends with `validation_failed` and no publication.

3.7 Condenser: From the Full Plan to the Executable Scientific Program

After critique and validation, the Condenser applies Π_{exec} to retain the task summary, global parameters and their sources, DAG-ordered steps with equations and paths, final file contracts, and essential units, ordering, naming, and environment notes. The resulting `CondensedPlan` contains a title and problem statement, parameters, steps with methods and outputs, an output table specifying paths, shapes, dtypes and content, and a small set of notes.

Search queries, evidence URLs, role dialogue, review findings, acceptance discussions, risk lists, and process artifacts such as `execution_status.json` are removed. The full `PlanDoc`, candidate versions, and validation trace remain archived for audit; downstream CodeBuddy, Codex, or minimal ReAct consumes only the published `EXP.md`.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

4.1.1 EVALUATION DATA AND PROMPT CONSTRUCTION

We evaluate IntentSci-Graph on ASI-BENCH using its official task manifest, execution interface, and scorer (Zhou et al., 2026). Its 60 project-level tasks span 11 domains: Mathematics, Physics, Chemistry, Biology, Astronomy, Materials Science, Earth Science, Medicine and Biostatistics, Computer Science, Robotics, and Electrical Engineering. Each task provides four matched information levels: B1 supplies the complete method, equations, and procedure; B2 retains method guidance without the full procedure; B3 retains only the research objective, data, and deliverables; and B4 adds plausible but task-irrelevant distractors to B3. The objective, workspace, scorer, and required deliverables are fixed across levels.

For task t and level $l \in \{1, 2, 3, 4\}$, we apply the synthesis and projection defined above to $I_{t,l} = (B_{t,l}, W_t, C_t)$, denoting $S^{\text{exec}}_{t,l}$ by $\text{EXPB}_{t,l}$. Thus Raw receives B1–B4, whereas EXP receives only the corresponding EXPB1–EXPB4. Role prompts, validation rules, iteration budgets, and planner configuration (θ, ξ) remain fixed across paired conditions.

4.1.2 EXECUTION CONFIGURATION AND METRICS

We use DeepSeek-V4.1-Flash and GLM-5.3-Flash as downstream execution models, each paired with CodeBuddy, Codex, and a minimal ReAct architecture. CodeBuddy and Codex use pinned CLI adapters. ReAct follows an observation, reasoning/action, and tool-use loop without an additional planner or reflection module; near the context limit, it summarizes the earlier half of the trajectory while retaining recent interactions, current task state, and generated files. The planning model is recorded separately from the execution model.

All runs use Ubuntu 22.04 on the same host with eight NVIDIA A800-SXM4 80GB GPUs and 1 TiB memory. Few tasks require GPU-intensive computation; most rely on CPU processing, file operations, and model calls. Each run starts in a clean, independent workspace containing task data, starter code/configuration, its condition-specific description, and permitted dependencies. Reference answers, private scoring resources, and artifacts from other runs are unavailable.

Paired runs share the task snapshot, random seed, scorer, software versions, permissions, network policy, temperature, context/output-token limits, retry policy, and execution budget; execution order is randomized. Model API endpoints are fixed, and web search and browsing are disabled except for services explicitly permitted by the task. CLI versions, dependencies, tool permissions, budgets, and collection rules are recorded in run metadata.

The only outcome is the official task score $R^{c}_{t,l,a,m,s} = \text{Score}(Y^{c}_{t,l,a,m,s})$, where $c \in \{\text{Raw}, \text{EXP}\}$, a denotes architecture, m the execution model, and s the run index. We report the paired difference $\Delta R_{t,l,a,m,s} = R^{\text{EXP}}_{t,l,a,m,s} - R^{\text{Raw}}_{t,l,a,m,s}$, first averaging runs within a task and then giving each task equal weight.

4.2 MAIN EXPERIMENTAL RESULTS AND FINDINGS

Table 1 compares Raw and EXP inputs across the four reported architecture–model configurations.

Table 1 shows a clear information-dependent pattern. At the macro level, EXP changes the score by -7.92 points on B1, -0.45 on B2, $+2.20$ on B3, and $+7.00$ on B4. Thus, the raw-prompt drop from B1 to B4 is 25.80 points, whereas the EXP drop is 10.88 points: the explicit program reduces the degradation caused by missing instructions by 14.92 points. The average over all 16 configuration–level cells changes only from 23.12 to 23.33 ($+0.21$), because the gains on sparse prompts do not fully compensate for the losses on B1.

Table 1: Main ASI-BENCH scores for B1–B4. Each cell is the macro-average over tasks. R_l^c denotes the task-level average for level l and condition c, $\Delta R_l = R_l^{\mathrm{EXP}} - R_l^{\mathrm{Raw}}$, and Avg. ΔR is the mean across the four levels. Bold entries are the numerically largest values within each reported column.

Agent	Model	B1	EXPB1	ΔR_1	B2	EXPB2	ΔR_2	B3	EXPB3	ΔR_3	B4	EXPB4	ΔR_4	Avg. ΔR
CodeBuddy	DeepSeek-V4.1-Flash	**43.88**	**35.71**	-8.17	**30.39**	**32.09**	**+1.70**	**20.94**	**27.74**	**+6.80**	**18.67**	**29.56**	**+10.89**	**+2.91**
CodeBuddy	GLM-5.3-Flash	34.90	27.81	-7.09	25.33	22.29	-3.04	18.06	14.17	-3.89	11.51	15.73	+4.22	-2.45
Codex	DeepSeek-V4.1-Flash	43.10	31.85	-11.25	22.05	20.27	-1.78	19.59	21.10	+1.51	14.20	19.59	+5.39	-1.53
ReAct	DeepSeek-V4.1-Flash	31.26	25.71	**-5.55**	15.60	16.91	+1.31	9.83	14.20	+4.37	8.11	15.60	+7.49	+1.91
Macro average		38.92	31.00	-7.92	23.34	22.89	-0.45	17.11	19.30	+2.20	13.12	20.12	+7.00	+0.21

Table 2: Descriptive domain-level ASI-BENCH scores for CodeBuddy with DeepSeek-V4.1-Flash (Raw / EXP). Bold entries are the numerically largest Raw or EXP value within each prompt-level column; domain rows are not combined into the main macro-average.

Scientific domain	Tasks	B1	B2	B3	B4
Mathematics	12	27.56 / 19.61	35.23 / 36.19	15.18 / 16.75	13.83 / 31.72
Physics	8	49.32 / 31.34	36.16 / 28.22	26.92 / 20.24	9.19 / 27.12
Chemistry	6	65.58 / 34.59	35.98 / 26.63	18.54 / 46.32	29.88 / 34.42
Biology	6	29.73 / 22.68	18.77 / 32.05	17.46 / 20.94	15.91 / 24.03
Astronomy	3	60.63 / 70.83	**74.21** / 37.62	31.33 / **63.44**	11.84 / 41.33
Materials Science	2	**87.09** / **89.31**	20.43 / 22.20	25.25 / 22.53	25.14 / 19.63
Earth Science	4	60.71 / 64.69	0.00 / 30.62	**31.80** / 39.75	**37.25** / 43.31
Medicine and Biostatistics	1	0.00 / 2.98	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
Computer Science	9	16.52 / 27.06	17.29 / 20.02	10.64 / 25.58	12.03 / 10.55
Robotics	7	83.69 / 57.88	46.59 / **52.52**	29.89 / 26.17	29.02 / **48.52**
Electrical Engineering	2	1.00 / 5.65	19.28 / 1.00	3.52 / 1.00	3.71 / 1.00
Total / task-weighted mean	60	43.88 / 35.71	30.41 / 30.69	19.95 / 26.27	17.80 / 28.69

The configuration-level results clarify when this effect is reliable. EXPB4 improves over B4 in all four reported configurations ($+10.89$, $+4.22$, $+5.39$, and $+7.49$), while B3 improves in three of four configurations and B2 has mixed directions. Averaged over B3 and B4, the gain is $+8.85$ for CodeBuddy–DeepSeek-V4.1-Flash, $+5.93$ for ReAct–DeepSeek-V4.1-Flash, $+3.45$ for Codex–DeepSeek-V4.1-Flash, and only $+0.16$ for CodeBuddy–GLM-5.3-Flash. CodeBuddy with DeepSeek-V4.1-Flash has the highest absolute scores on every prompt level and the largest four-level average gain ($+2.91$), whereas ReAct with DeepSeek has lower absolute scores but benefits consistently on B2–B4. These comparisons describe the reported configurations; the table is not a complete factorial estimate of independent architecture and model main effects.

All four B1 differences are negative (-7.77, -7.09, -11.25, and -5.55). B1 already contains an expert-authored procedure, priors, model-selection rules, failure checks, and an output contract. Rewriting such a protocol can compress or alter precisely the tacit choices that make it effective. IntentSci-Graph should therefore be viewed as a structural repair layer for under-specified intent, while complete expert protocols should be preserved and subjected to verification rather than rewritten without evidence of fidelity.

4.3 Results and Findings Across Scientific Domains

Table 2 reports within-domain mean scores for B1–B4 using CodeBuddy with DeepSeek-V4.1-Flash, with each cell showing Raw / EXP. We use this breakdown to study domain heterogeneity; domains with one to three tasks are interpreted descriptively.

Benefits are domain dependent and concentrate in under-specified tasks. At B4, EXP exceeds Raw in seven of the eleven domains, with the largest gains in Astronomy ($+29.49$), Robotics ($+19.50$), Physics ($+17.93$), and Mathematics ($+17.89$). At B3, the largest gains occur in Astronomy ($+32.11$), Chemistry ($+27.78$), and Computer Science ($+14.94$), consistent with explicit steps, parameters, and output contracts repairing missing procedural structure.

Rewriting a method-rich prompt can remove useful domain choices. The B1 score decreases in Chemistry (-30.99), Robotics (-25.81), and Physics (-17.98), as well as in several smaller negative cases. These regressions are consistent with the loss or alteration of expert-provided period priors, model-selection rules, stability checks, and output conventions.

Execution artifacts do not imply scientific correctness. A score of 0 or 1 can correspond to relevant files being produced, while the artifacts still fail ASI-BENCH's hard minimum gates. The result exposes a completion–quality gap: autonomous research must evaluate whether the scientific artifact satisfies the task's hard criteria, not only whether an execution trace reaches the requested files.

4.4 Case Study

As shown in Figure 3, we examine `Astronomy_Planet_Activity_Sep`, which requires separating a case-specific component A from a shared background component B in three irregularly sampled observation channels and predicting future component-A-only values. The left panel compares the information supplied at each prompt level: red, yellow, green, and blue markers denote B1, B2, B3, and B4, respectively. All four levels retain the goal, input files, and output requirements. B1 and B2 provide procedural guidance, while B1 further details the scientific model, recommended methods, numerical conventions, and failure modes. These categories summarize the methodological guidance; coefficient definitions retained in the output contract remain available at the lower levels. In addition to the three marked categories, B4 contains task-irrelevant scientific context that is not expanded in the panel.

From a sparse request to an explicit scientific program. The transformation shown is specifically B4 to its executable scientific program, rather than four separate synthesis trajectories. Starting from $I_{t,4} = (B_{t,4}, W_t, C_t)$, IntentSci-Graph uses the B4 request and task workspace to specify the missing methodological decisions; the more detailed B1–B3 prompts on the left serve only as comparisons. The right panel summarizes the resulting task objective, input inspection, parameter specification, ordered execution steps, output summary, and execution notes. The steps connect input parsing and period search to harmonic regression, parameter estimation, and file generation. The output contract requires `signal_summary.csv` with four rows and eleven columns and `target_predictions.csv` with rows aligned to `input_2.csv`. The notes make the common `reference_epoch_day`, exact column order, and no-A cases explicit. In the figure, *IntentSci* abbreviates IntentSci-Graph, and `ESP_B4` denotes $S_{t,4}^{\mathrm{exec}}$, called EXPB4 in the experiments.

The missing information concerns scientific decisions, not only deliverables. B4 already names the required output files; synthesis adds a proposed route from the observed data to those outputs. In the inspected B4 plan, the cosine/sine basis order is swapped relative to the convention stated in the B4 prompt. This is a plan-level schema inconsistency rather than evidence of a scored execution failure, and it illustrates why the Critic and Validator must check equations against output columns before condensation. The figure summarizes the program representation, not an execution trace or a task-specific score. A single input–program example does not establish that the recovered method matches an expert protocol or quantify variability across repeated synthesis runs.

5 Discussion

The results expose a useful boundary for scientific program synthesis. When methodological information is missing, an explicit program supplies a stable interface: it makes dependencies, equations, parameters, and output contracts available to an executor before tool use, and the gains are largest on B4. When the input already contains a complete expert protocol, synthesis can reduce performance, as shown by the uniformly negative B1 deltas. The overall mean gain is only $+0.21$ because these two effects interact; it should not be read as a uniform improvement across prompt levels or executors. Current role-playing experts therefore recover procedural structure more reliably than tacit scientific expertise. They can produce a coherent plan, yet still select the wrong model, prior, or validation criterion.

The domain table reinforces this distinction: a produced artifact can receive a score of 0 or 1 when ASI-BENCH hard gates are unmet. Execution completion is consequently insufficient evidence of scientific correctness. Reliable agents will need expert-protocol preservation, uncertainty-aware

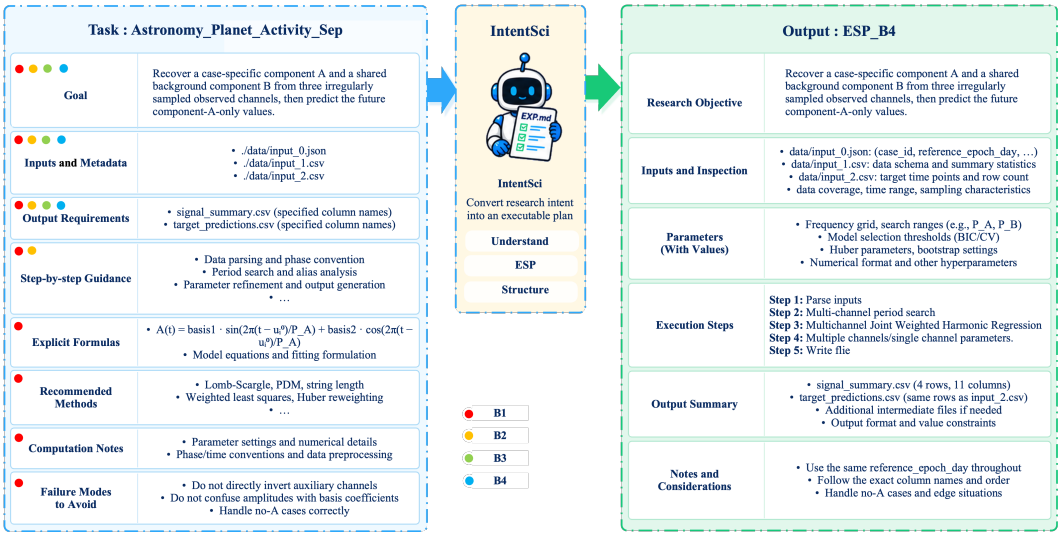


Figure 3: **B4-to-ESP synthesis for Astronomy_Planet_Activity_Sep.** Left: colored markers summarize information coverage in B1 (red), B2 (yellow), B3 (green), and B4 (blue); B4 also includes irrelevant context not expanded here. Center: IntentSci-Graph constructs an explicit scientific program from B4 and the task workspace. Right: the execution-facing program, labeled `ESP_B4` (EXPB4 in the experiments), specifies the objective, input inspection, parameters, execution steps, output contract, and notes. The panel summarizes program content rather than downstream execution results.

synthesis, and validation tied to scientific semantics rather than formatting alone. The potential basis-order mismatch in the B4 plan illustrates a concrete failure mode: a coherent-looking program can still violate the output schema. The stochastic variation observed in the case study therefore motivates repeated synthesis, schema-aware checks, and explicit preservation of expert-authored equations and conventions.

6 Limitation

Our results expose a persistent knowledge gap in autonomous research. Agents may achieve high task-completion rates while still producing results that fail rigorous domain-expert evaluation, because IntentSci-Graph cannot reliably recover the tacit knowledge, priors, and decision criteria used by experts. On some tasks, executor performance remains poor with or without the synthesized program, showing that improving the execution interface alone is insufficient. Reliable autonomous research therefore requires joint progress in executable scientific-program synthesis and downstream execution, together with mechanisms for acquiring, preserving, and validating domain expertise. IntentSci-Graph explores the program-synthesis side of this challenge, but it does not by itself close the gap between producing an artifact and conducting scientifically correct research.

7 Conclusion

We introduced IntentSci-Graph, a scientific reasoning layer that converts research intent into an explicit, verifiable scientific program before execution. On ASI-BENCH, EXP programs improve average scores at B3 and B4, where methodological guidance is sparse, while B2 remains near parity and fully specified expert prompts remain stronger than their synthesized counterparts. The result establishes both the utility and the limit of the layer: explicit programs reduce planning ambiguity, but current LLMs do not yet reproduce the tacit knowledge encoded by domain experts. Separating program synthesis from execution provides a concrete interface for future systems to acquire, verify, and preserve such expertise.

AI USE STATEMENT

In this work, generative AI applications were employed to facilitate and polish the writing process, encompassing sentence restructuring, grammatical corrections, and the enhancement of textual readability. Additionally, AI tools were utilized for literature retrieval and discovery (e.g., locating related research). No other AI-assisted tasks were incorporated into this work. The authors have conducted a comprehensive review of all AI-assisted material. All AI-generated outputs were manually inspected and modified by the authors to maintain accuracy, coherence, and scientific rigor. The authors bear full responsibility for the final content of this paper, including any text, claims, or artifacts developed with generative AI assistance.

ETHICS STATEMENT

This work uses the ASI-Bench benchmark and does not involve human subjects, personal data, or human-subject annotation. The study evaluates autonomous research agents in a controlled benchmark setting and does not deploy them in high-stakes scientific or operational environments. We follow the applicable benchmark, software, and model-use requirements.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the scientific-program representation, role-based synthesis procedure, validation rules, prompt construction, executor configurations, and evaluation metrics. Experiments use the official ASI-Bench task interface and scorer with fixed task snapshots and recorded model and execution settings. Anonymous implementations, prompts, configuration files, and supplementary execution artifacts will be provided to support reproduction.

REFERENCES

Bartosz Balis, Michal Orzechowski, Piotr Kica, Michal Dygas, and Michal Kuszewski. From research question to scientific workflow: Leveraging agentic ai for science automation. *arXiv preprint arXiv:2604.21910*, 2026.

Aubrey Brueckner, Darshil Patel, Yuhuan He, and Timothy Kassis. K-bench: Measuring model performance on real scientific agent requests. *arXiv preprint arXiv:2608.21601*, 2026.

Yanlin Fei et al. How do agents fail on autoresearch: End-to-end diagnostic evaluation on 100 real-world frontier research tasks. *arXiv preprint arXiv:2608.14905*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Congyu Zhang, et al. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration. *arXiv preprint arXiv:2605.20025*, 2026.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-researcher: Autonomous scientific innovation. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Hao Wang, Ligong Han, Kai Xu, and Akash Srivastava. Turning intent into specifications: A benchmark and an interactive user-assistant agent. *arXiv preprint arXiv:2606.20585*, 2026.

Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. GTA: A benchmark for general tool agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 75749–75790, 2024.

Wanghan Xu et al. Researchclawbench: A benchmark for end-to-end autonomous scientific research. *arXiv preprint arXiv:2606.07591*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Zhengxin Zhang, Ning Wang, Sainyam Galhotra, and Claire Cardie. How far are we from true auto-research? *arXiv preprint arXiv:2605.19156*, 2026.

Junwei Zhou et al. ASI-bench: At the dawn of artificial superintelligence. *arXiv preprint arXiv:2608.17271*, 2026.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and J"urgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62743–62767, 2024.