AGENT BEHAVIOR AS CODE: EFFICIENT AND ROBUST LLM AGENTS WITH PROGRAMMATIC SPECIFICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents based on foundation models (FMs) have demonstrated strong capabilities to perform complex open-ended tasks. However, they face some common challenges in practice: (a) agent behavior can deviate drastically even for semantically similar tasks, leading to catastrophically propagated errors; (b) high cost and latency due to FM calls, repeated in full whenever a task recurs with different inputs; (c) FMs' limited context and instruction following capability confine how well agents manage the ever-growing execution context and follow complex plans. We introduce **A**gent **B**ehavior as **C**ode **Agent** (ABCAGENT), which uses a symbolic program (*e.g.*, Python code with potential neural functions) to fully specify the agent's behavior at runtime, with a powerful FM agent editing that program for flexibility. Behavior is thus specified without premature variable binding, and its execution is deterministic. We evaluate ABCAGENT on six agent benchmarks, two of which we construct to test how well a derived program generalizes to variants of the task it was written for. ABCAGENT matches a model-matched neural agent on GAIA and augmented GAIA, and surpasses it where robustness and long control flows matter: 98.3% against 97.3% on GSM-Symbolic ($p = 0.001$), 71.9% against 47.4% Pass^4 on the telecom domain of τ^2-bench ($p = 0.0001$), and more records written correctly at every loop length on our control-flow-augmented WorkArena benchmark. For more parametric task families, ABCAGENT is also significantly superior in efficiency. Without authoring a new program, ABCAGENT solves 92.6% of GSM-Symbolic instances and 20.1% of augmented GAIA variants, which yields $5.2\times$ lower latency and $7.0\times$ lower cost on GSM-Symbolic, 19% lower cost on augmented GAIA, and $9.5\times$ lower agent latency on τ^2-telecom.

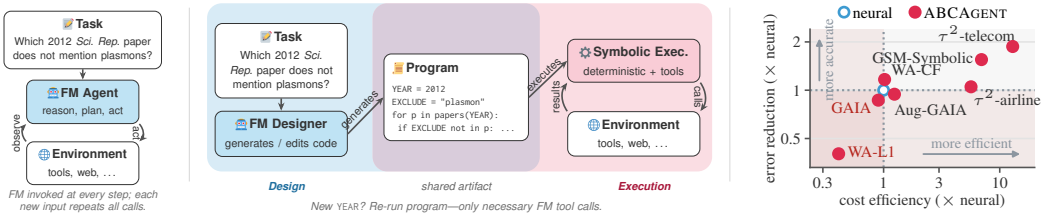


Figure 1: How ABCAGENT improves over traditional FM agents. **Left:** Traditional FM agents (e.g., ReAct (Yao et al., 2023)) re-invoke the FM at every action step, making reruns costly when task inputs change. **Center:** ABCAGENT separates *design* (task → program) from *execution* (program → result). The program can contain neural functions (e.g., `papers()` in this example) for flexibility. On any new input, only the program is re-run, eliminating redundant FM calls, especially orchestration calls. **Right:** Error reduction against cost efficiency on six benchmarks relative to the neural agent.

1 INTRODUCTION

Foundation models (FMs) have provided a unified and generalizable interface to numerous artificial intelligence (AI) tasks spanning different modalities, giving rise to numerous agentic systems including GUI agents (Zheng et al., 2024), deep research (Huang et al., 2025; Zheng et al., 2025c), coding

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.

agents (Zhang et al., 2024b; Yang et al., 2024a; Wang et al., 2025a), and interactive agents that help users complete tasks with external tools (Schick et al., 2023), to name a few.

Although FM agents have demonstrated wide applicability in sophisticated tasks that are *one-off*, they struggle with functioning efficiently and reliably for tasks that are *recurring yet varied*. For instance, Google's AI feature had struggled to handle "was 1995 30 years ago"(Marcin, 2025), and continued to struggle on similar queries like "was 1996 30 years ago" after fixes. This is challenging for purely FM-based solutions since they are shown to struggle with compositional generalization (Xu et al., 2024b; Lake & Baroni, 2018). Such erratic behavior would be unacceptable in real-world applications. Ideally, the agent's behavior should be similar to a function $f(X, Y)$ for all queries like "was Y X years ago", regardless of how sophisticated the underlying task might be. With such abstractions, it is also possible to convert some of the agent's behavior specifications into programmatic code, improving both the cost/latency by reducing unnecessary FM calls and tokens, and the transparency and controllability of the agent's behavior.

Recent work has proposed to induce programmatic skills to address these issues (Wang et al., 2023; 2024b; 2025b; Zheng et al., 2025a; Yu et al., 2026). With skill induction, FM agents can write programmatic subroutines into a library that they use for subsequent tasks. While the agent is still orchestrated by the FM, these subroutines provide determinism and efficiency for small parts of agent operations when reused. Concurrent work (Qi et al., 2026) argues that a program rather than the FM should govern an agent's control flow; our focus is instead on reusing such programs across the tasks of a family, which requires them to be re-bindable and checked for generality.

To address these challenges, various neuro-symbolic frameworks have been proposed to combine the strengths of these systems. However, most of these are either calling relatively simple neural functions from a purely programmatic foundation, or simply leveraging symbolic programs as tools in a larger neural agent that is still orchestrated by a foundation model (we refer the reader to Sarker et al. (2021) for a taxonomical review of various neuro-symbolic approaches in artificial intelligence). Neither paradigm fundamentally address the challenge of balancing the flexibility of neural agents in solving diverse tasks and the robustness and efficiency symbolic engines afford, since nesting neural and symbolic executions in a layered system is largely impossible with these formalisms.

In this work, we present ABCAGENT, which tackles these challenges by organically combining FM capabilities with code execution to combine the advantages of both neural and symbolic systems. Instead of relying on purely neural FMs to keep track of and execute complex agentic workflows, we introduce the concept of *agent behavior as code (ABC)*, where the behavior of the agent is fully specified as programming code with potential calls to neural models. This allows ABCAGENT to enjoy the efficiency and robustness of programming code in orchestrating complex control flows when FM capabilities are not needed, while enjoying the flexibility of adapting to new use cases when execution failures occur, or when the neural agent determines that the requirement for the agent has changed (See Figure 1). As we will show in subsequent sections, this also allows the agent to seamlessly generalize to similar tasks with minimal to no changes to the code that defines agent logic, leveraging code to store behavioral knowledge of the AI agent to allow for efficient adaptation.

2 AGENT BEHAVIOR AS CODE

In this section, we introduce the framework of agent behavior as code, with the crucial concept of separating agent design from agent execution, and detail how we implement the ABCAGENT based on this framework.

2.1 SEPARATING AGENT DESIGN FROM EXECUTION

Before introducing the *agent behavior as code* framework, we make the distinction between two phases of activities when working with AI agents, *agent design* and *agent execution*. Specifically, by *agent design* we mean all activities aiming at altering the behavior of the agent to achieve the desired outcome, including but not limited to explicitly steering agent behavior through chat interactions, providing exemplars for agent tasks, providing feedback on intermediate agent outputs, etc. On the other hand, we use *agent execution* to refer to autonomous operations that the agent performs towards completing the task the user has in mind, including correct and incorrect attempts at solving the problem at design time, and subsequent application of the agent to new inputs where immediate user

feedback may or may not be available. By this definition, the design and execution of many AI agents today are tightly coupled, where they share the same context, where agent execution logic and the execution process is coupled with the specific user inputs and examples provided at design time (with the exception of explicitly induced skills). This coupling allows the user to intuitively design agent behavior with a responsive and flexible interface, but poses challenges when later the user needs to reuse the agent on different inputs, which we outline in subsequent paragraphs.

Challenge 1: Variable binding. As agent design typically involves real or hypothetical examples of agent task input, which the agent works through with user supervision to achieve the desired behavior. However, when agent behavior is defined and the agent needs to generalize to new inputs, we typically rely on FMs to correctly remove the influence of example inputs provided at design time, where hallucinations can contribute to example leakage. Classical AI systems faced the same problem: explanation-based generalization replaces the constants of a worked example with variables (Mitchell et al., 1986), macro-operators and chunking store generalized solutions for later reuse (Fikes et al., 1972; Korf, 1985; Laird et al., 1986), and case-based reasoning adapts a retrieved solution to a new problem (Kolodner, 1993; Veloso & Carbonell, 1993).

Challenge 2: Robust edit of agent behavior. Since all agent behavior is typically defined and edited in the form of agent interaction with users and the environment, this also presents a challenge when users want to edit the agent behavior at design time (*e.g.*, by introducing if-this-then-that rules to handle a class of edge cases). The FMs' capability to correctly follow prompt instructions in a complex context is the only lever available, and there is little verifiable guarantee that the agent design reaches the goals the user sets out to achieve or well-generalizes to unseen cases.

Challenge 3: Context management and execution efficiency. If all the design-time context is fully kept during execution time, this can introduce significant context management overhead and reduce agent execution efficiency at runtime. Even with context compaction between design time and execution time, the aforementioned challenges still stand.

To address these challenges, we propose agent behavior as code, a framework that cleanly separates agent design from execution by leveraging programming code as the source of truth to fully specify agent behavior at execution time. As a result, the goal of agent design is to arrive at an agentic program that captures the desired behavior with explicit generalization considerations and no premature variable binding, focusing on specifying agent behavior in general rather than what is the expected result in a specific case. This addresses all aforementioned challenges by 1) explicitly define and use unbound input variables in the agentic program; 2) use code logic to reflect agentic behavior to ensure edit lineage and avoid regression; 3) delegate expensive LLM-based orchestration to much more efficient code executors when possible, which is not only efficient and robust, but also enjoys unbounded execution context thanks to modularization, variable scoping, and nesting natively supported by programming languages (*e.g.*, a for loop that runs n times would not require $O(n)$ context / memory, an observation concurrent work also makes (Qi et al., 2026)).

2.2 Implementation of ABCAgent

To generalize to unseen inputs, an ABCAgent program declares the parameters it can be re-bound on up front, separating the values a particular task supplies from the logic that consumes them. A program is therefore applied to a later task by supplying values rather than by regenerating its logic.

The authoring loop itself is a standard code-generation cycle. The FM Designer analyzes the task with the current program, execution state and feedback from previous executions (if any) to propose a program that addresses the task requirements. It then hands the proposed program to the Symbolic Executor for end-to-end validation. When execution fails or when an LLM-based critic detects an issue with the program, the outcome is returned to the FM Designer for revision. During authoring, we ask the FM Designer to propose variants of the task at hand, and solve them alongside the original task with its program. These variants are then used to check the generality of the program through execution before it is accepted; if the program fails to change its behavior appropriately for the variants, it is rejected. This check is executable and cheap, and it rejects programs whose generality is merely declared but not implemented. This check is akin to metamorphic testing (Chen et al., 1998; Segura et al., 2016): each variant asserts how the output should change when a parameter does, so a

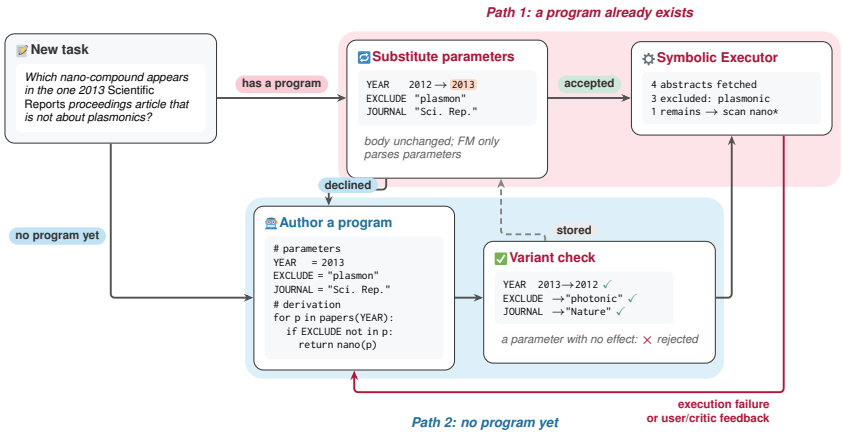


Figure 2: ABCAGENT processes new tasks by authoring programs from scratch, or reusing them from previously solved similar tasks. In the latter case, ABCAGENT attempts to substitute parameter values into the existing program (with a lightweight FM call for parsing only when necessary), before resorting to editing the program if substitution fails.

program that ignores the parameter fails it. It is related to validating generated code against generated tests (Chen et al., 2022a) and to property-based testing (Claessen & Hughes, 2000); see Zheng et al. (2026) for a survey of metamorphic testing with language models.

When ABCAGENT encounters a new task, it begins by authoring a fresh program from scratch. Once a program is drafted and a subsequent task in the same problem family is encountered, ABCAGENT will first attempt to apply the existing program to the new task by substituting the new task's parameter values. Should this attempt fail, the FM Designer is invoked to revise the program so that it can handle the new task without regressing on previously solved tasks. This allows the system to leverage existing knowledge efficiently, only invoking the FM Designer when necessary to accommodate new variations.

For illustration, consider the parameterized literature-search task of Figure 2. A purely FM-based agent would repeat the entire search process with FM-orchestrating calls due to the nested, iterative nature of this task; in contrast, ABCAGENT can reuse generally applicable programs to handle new tasks more efficiently.

2.3 IMPLICATIONS FOR STATE AND MEMORY MANAGEMENT

ABCAGENT's neural-symbolic architecture fundamentally changes agent memory management by structuring it around the principles of program execution.

- **Short-Term Knowledge (Variable Management):** Instead of injecting all knowledge back into a flat text context, agent state and short-term memory are managed as program variables. When the orchestrator executes `x = 5`, the variable `x` is stored with its value and type. This state is precise, typed, and requires no interpretation. It avoids the ambiguity and context-length limitations of trying to remind an FM that "we previously established a value of 5 for the concept x."
- **Execution Progress (Execution Frames and Scoping):** The agent's progress is not merely a transcript of past actions but a structured call stack. When the orchestrator enters a for loop or a function, it creates a new execution frame with its own local scope. This allows the agent to handle complex, nested tasks without losing track of its position in the overall plan. For instance, the orchestrator, not the FM, is responsible for tracking that it is on iteration 2 of 3 in the loop from our example. This robustly manages execution state, preventing the kind of drift or forgetfulness common in long-running, purely neural agents. Furthermore, variable scoping is handled naturally by the symbolic engine, preventing state pollution between different subtasks or loop iterations. This structure inherently supports modularity, nesting, and even recursion, enabling a level of complexity that is intractable with a flat memory model.

3 EXPERIMENTS

In this section, we demonstrate the efficacy and efficiency of ABCAGENT on a number of diverse agent benchmarks.

3.1 EXPERIMENTAL SETUP

Benchmarks & Metrics. We evaluate ABCAGENT on the following agent benchmarks: 1) **GAIA** (Mialon et al., 2023) is a collection of general AI agent tasks that require multistep autonomous reasoning and tool use for textual question answering. The dataset is divided into L1, L2, and L3 subsections for increasing difficulty, where more tools and reasoning steps are required for higher levels. We use the development set for evaluation. 2) **WorkArena** (Drouin et al., 2024; Boisvert et al., 2024) features realistic, enterprise-focused web tasks on the ServiceNow platform. WorkArena is similarly divided by difficulty levels; we evaluate on the L1 subset. 3) **GSM-Symbolic** (Mirzadeh et al., 2024) augments 100 grade-school math word problems with symbolic templates, and samples 50 instances from each template with varying names and numerical values while keeping the underlying reasoning fixed. 4) **τ^2-bench** (Barres et al., 2025) evaluates conversational agents in customer-service domains, where the agent must follow a domain policy while helping a simulated user. We use the telecom and airline domains, and report the benchmark's Pass^k metric, which requires all k trials to succeed ($k \in \{1, 4\}$). For all of these benchmarks, we adopt the official evaluation metric that calculates average task success rate on each benchmark or subset thereof.

To stress test ABCAGENT on generalization and efficacy of handling complex tasks, we further augment existing benchmarks.[1] For GAIA, we generate questions that slightly modify the original ones in the development set, and annotate corresponding answers to these modified questions with Co-Sight (Zhang et al., 2025), a high-accuracy system on the GAIA leaderboard with some human verification. We then test how well the agent behavior program that ABCAGENT derives from the original question generalizes to these new tasks. For WorkArena, we compose the original L1 tasks into more complex ones with control flows (e.g., if/else branching or for loops) that the agent must follow to successfully complete these tasks. We created 18 augmented tasks in total for this WorkArena-CF benchmark, 8 involving loops and 10 involving branching. For both augmented datasets, we use evaluation metrics consistent with the original dataset for task success rate calculation. We refer the reader to Appendices C and D for details about these augmented datasets.

To further illustrate the execution efficiency of ABCAGENT, we also compare the time it takes a neural agent and ABCAGENT to finish tasks on these benchmarks, the time it takes for execution without redesign, the cost of FM calls, the share of tasks solved without authoring a program, and other diagnostic metrics, if applicable. We refer the reader to Appendix B for further details about the evaluation protocols, including cost accounting and statistical tests.

Baselines. We compare ABCAGENT to state-of-the-art approaches on each benchmark: for public reference, we compare against the leading agents on the same data split that we evaluate on for each benchmark. We also implement our own version of a purely neural agent for comparison, which leverages the same Claude Sonnet 4.6 model as its backbone and Claude Code as its agent design – but unlike ABCAGENT, it does not store or reuse agent behavior programs. This neural baseline is more important especially for GAIA tasks, where targeted harness engineering can make a significant difference in performance.

3.2 MAIN RESULTS

ABCAGENT is as accurate as neural agents, and more robust in complex and symbolic-like tasks. As can be seen in Table 1, on GAIA, augmented GAIA, WorkArena, and τ^2 airline domain, we note that the accuracy of ABCAGENT is statistically indistinguishable from that of a model- and harness-matched neural agent. For GAIA and Augmented GAIA,this accuracy match holds across different difficulty levels and task variations (see Appendix Table 5). This shows that we do not lose accuracy by specifying agent behavior through explicit programs compared to relying on neurally-orchestrated agents. Furthermore, on GSM-Symbolic and τ^2 telecom, two tasks that are

[1] We will release the implementation of ABCAGENT and both augmented benchmarks upon acceptance.

System	GAIA		WorkArena		GSM-Symbolic	τ^2-bench (Pass^4)	
	base (165)	augmented (378)	L1 (31)	CF (42)	(5000)	telecom (114)	airline (50)
Reference	74.6	—	42.7	—	94.9	29.0	42.0
Neural agent	*76.4*	*59.5*	*84.9*	58.3	97.3	47.4	58.0
ABCAGENT	72.7	57.1	62.4	*64.3*	**98.3**	**71.9**	*60.0*
p-value	0.377	0.281	0.041	0.453	**0.0011**	**0.0001**	1.000
Cost efficiency	0.90×	1.24×	0.41×	1.02×	6.97×	12.93×	5.66×
Speedup	0.85×	1.33×	0.36×	0.92×	5.21×	9.50×	4.08×

Table 1: Task success rate (%) and efficiency of ABCAGENT relative to model- and harness-matched neural agent on six agent benchmarks. Bold marks a statistically significant difference ($p < 0.01$) and italics the stronger system otherwise; cost efficiency and speedup are the neural agent's cost and time divided by those of ABCAGENT on the tasks where both systems produced a result, highlighted when the deviation is $\geq 20\%$. The reference row is the strongest published result we are aware of, with — where none exists (see Appendix B for details).

more amenable to symbolic generalization, ABCAGENT demonstrates a clear advantage over neural agents. On τ^2-bench, ABCAGENT is also more stable across repeated trials: its Pass^1 to Pass^4 drops from 81.6% to 71.9% on telecom and from 68.5% to 60.0% on airline, more modest compared to the drops of 60.8% to 47.4% and 68.5% to 58.0% for the stock neural agent (Appendix Table 7). On WorkArena-CF, even when full task success rate is comparable, ABCAGENT consistently writes more database records correctly than the neural agent at every loop length (Appendix Figure 4a). This highlights ABCAGENT's robustness in scenarios where explicit programmatic reasoning is beneficial.

ABCAGENT is more cost- and time-efficient, especially with reuse or on tasks with complex control flows. Table 1 also shows that ABCAGENT is at least as cost- and time-efficient as the neural agent on most benchmarks, and substantially more so where its programs are reused. On GSM-Symbolic and τ^2-bench, ABCAGENT is noticeably more cost- and time-efficient than the neural agent at the same time. We note that these savings do not come from cheaper inference for FM calls necessary to solve the task, which costs about as much for ABCAGENT as for the neural agent; instead, they come from the 92.6% of GSM-Symbolic tasks and 20.1% of augmented GAIA tasks that ABCAGENT solves without authoring, by rebinding the parameters of an existing program (see Section 4 for detailed analysis).

4 ANALYSIS

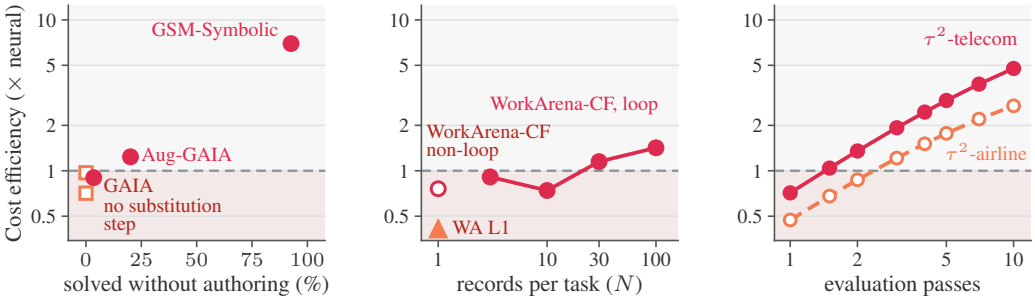
In this section, we perform detailed analysis of the mechanisms and behaviors of ABCAGENT.

Program reuse and variable rebinding drive the efficiency of ABCAGENT, and the variant check makes its programs generalizable. As can be seen in Table 2, while variant check introduces moderate overhead in cost, it helps improve the generalizability and correctness of the programs authored by ABCAGENT especially on WorkArena-CF. Meanwhile, sharing programs across similar tasks results in the biggest efficiency gains for ABCAGENT by reducing the need to author new programs for each task. We also note that these components are not additive: on GSM-Symbolic, family reuse without cheap rebinding is more expensive than no reuse at all (Appendix Table 9), since every instance still goes through the authoring loop, only now with the family's program in its context.

We find that these savings take three different forms (Figure 3): across the tasks of a family, cost efficiency grows with the share of tasks that ABCAGENT solves without authoring; within a single task, a program-driven loop amortizes the authoring cost over the records it processes; and on τ^2-bench, one authored policy is evaluated repeatedly and pays back its authoring cost in fewer than three passes. This also explains the below-parity cost efficiency and speedup on WorkArena L1 in Table 1. WorkArena L1 affords none of these, since each of its tasks is a one-off that is neither repeated nor looped over, so ABCAGENT pays the authoring cost in full without ever amortizing it. Its lower nominal accuracy there has a different cause: most of ABCAGENT's failures on WorkArena

Setting	Augmented GAIA L1 (129) SR	Free	Cost eff.	GSM-Symbolic (1000) SR	Free	Cost eff.	WorkArena-CF (18) SR	Records	Cost eff.
Neural agent	62.0	0.0%	1.00×	99.7	0.0%	1.00×	76.5	53.3	1.00×
ABCAGENT	58.9	21.7%	1.27×	99.1	96.0%	11.91×	*83.3*[§]	*63.9*	1.04×
− reuse & rebind	*62.8*	0.0%	1.02×	99.4	0.0%	0.96×	77.8	63.3	0.80×
− variant check	56.6	0.0%	1.30×	*99.8*	0.0%	0.97×	68.8	0.0[‡]	1.26×

Table 2: Ablation of ABCAGENT's components. We use the L1 subset of Augmented GAIA and 20% of GSM-Symbolic for this experiment. *Free* is the share of tasks solved without authoring a program, and *Records* is the percentage of records written correctly on the loop tasks ($N = 30$). [§]On WorkArena-CF, cheap rebinding never took effect, so this setting behaves as family reuse with the variant check alone. [‡]Without the variant check, the replayed programs commit records with wrong field values, including one program that ignores its per-record input entirely (details in Appendix H).



(a) across tasks in a family (b) within a task, over records (c) across evaluation passes

Figure 3: Cost efficiency of ABCAGENT relative to the neural agent under three forms of amortization: (a) reusing a program across the tasks of a family, (b) reusing a loop body, and (c) evaluating a policy repeatedly. The open squares in (a) are ABCAGENT on GSM-Symbolic without rebinding.

L1 end with the browser on a page other than the one the benchmark's validator reads, such as a catalog order that never reaches its confirmation page (Appendix G), which we suspect is due to the authored program not checking the page state it leaves behind.

Program reuse retains strong backward compatibility within a family while moderately improving forward generalization. We also find that program reuse substantially improves backward compatibility within a family of tasks on both GSM-Symbolic and Augmented GAIA, retaining the ability to solve earlier tasks even as new ones are added, allowing for more reliable cumulative learning via robustly evolving the agent's behavior (see Table 3). This is because a stored program is revised under the requirement that it still solves the tasks it has solved before. In contrast, neural agents and programs authored independently for each task carry no such guarantee. On the forward direction, we find that reuse also improves the program's ability to solve yet unseen tasks within the same family, substantially on GSM-Symbolic but only nominally on Augmented GAIA, where the effect is not as pronounced as in the backward direction, because the forward tasks may introduce new requirements that were not present in the previously solved tasks.

On GSM-Symbolic, where variations among the instances of a family are typically in names and numbers over one fixed derivation, a stored program transfers to nearly all of its family, whereas programs authored one task at a time fail noticeably more often on their siblings. This makes GSM-Symbolic more amenable to few-shot learning for both task logic and shared interface, as also evidenced by the high "free" program rate in Table 2.

Hence, what reuse primarily provides is a stronger guarantee against regression, with improved forward generalization when new variants conform better to the patterns of seen tasks, although little exposure is needed before a stored program covers most of its family. We refer the reader to Appendix F for further analysis.

Transfer setting	pairs	Reuse	Control	p
GSM-Symbolic				
Forward ($j > i$)	7 640	**0.958**	0.842	$\mathbf{< 10^{-4}}$
Backward ($j < i$)	7 360	**0.963**	0.826	$\mathbf{< 10^{-4}}$
Augmented GAIA				
Forward ($j > i$)	235	*0.332*	0.302	0.410
Backward ($j < i$)	239	**0.469**	0.305	**3.5e-5**
Backward, target known-passing	145	**0.703**	0.462	**5.1e-5**

Table 3: Fraction of task pairs within a family on which a program stored for one task (indexed by i) solves the other (indexed by j), for unseen (forward) and earlier (backward) tasks, with and without program reuse. On GSM-Symbolic, parameters are rebound with one FM call (Appendix B); bold marks $p < 0.01$ and italics the stronger result otherwise.

System	Behavior specification	Adaptation signal	Pass[1]
Neural agent	prompt only	none	0.603
Neural agent + traces	prompt + prior transcripts	task transcripts	0.671
ABCAGENT	symbolic program	task transcripts	0.789
+ reward at authoring	symbolic program	transcripts + reward	0.943
Control: neural agent, transcript injection disabled			0.638

Table 4: Prompt-based neural agents and ABCAGENT adapting in a continual learning setting on τ^2-telecom. Revealing the reward at authoring is an oracle ceiling.

The generalization of ABCAGENT does not come from observing past runs alone. A natural question is, does ABCAGENT generalize simply because it observes the transcripts of past runs? To answer this, we give a neural agent on τ^2-telecom the same transcripts as context (Table 4). While the neural agents benefit modestly from this information about prior tasks, ABCAGENT benefits considerably more, which suggests that it is specifying behavior as a program, rather than as prose context, that makes past runs useful. We additionally report an oracle ceiling that reveals the reward of the task being authored, to which our main configuration has no access, to indicate how much further ABCAGENT could benefit from richer adaptation signals.

Other analyses. We further find that programs can helpfully defer to ABCAGENT nested-calls only where it resists code, that errors on WorkArena-CF do not accumulate with loop length, and that when rebinding fails, the reason is legible in the authored program; see Appendix G for details.

5 Background and Related Work

Neuro-symbolic AI. The effective combination of neural and symbolic systems has a rich history (Garcez & Lamb, 2023; Sarker et al., 2021; Hitzler et al., 2022; Bhuyan et al., 2024; Belle et al., 2024). Kautz (2022) highlights prominent paradigms, among which *Symbolic[Neuro]* denotes a symbolic program that calls neural functions (*e.g.*, AlphaGo's MCTS over a neural value function), and *Neuro[Symbolic]* denotes a neural system with embedded symbolic reasoning (*e.g.*, an LLM invoking a theorem prover). ABCAGENT proposes a new paradigm $NS = \text{NEURAL}[\text{SYMBOLIC}[NS]]$, a recursive definition: the outermost $\text{NEURAL}[\cdot]$ is the FM Designer that generates and edits programs, while $\text{SYMBOLIC}[NS]$ is the executor—a deterministic engine that may invoke neural tools or recursively call the FM Designer itself to handle sub-problems, yielding the nested NS. This enjoys the plasticity of neural systems alongside the efficiency, robustness, and out-of-distribution generalization of symbolic execution. Representing agent behavior as code also improves transparency, interpretability, and controllability; we refer the reader to prior work with detailed discussions on this topic (Hamilton et al., 2022; Tocchetti et al., 2025).

Neuro-symbolic methods for specific tasks. Neuro-symbolic systems have been applied across many tasks, illustrating Kautz's taxonomy of collaboration schematics. In *Symbolic[Neuro]*, Silver

et al. (2016) and Silver et al. (2018) combine Monte Carlo Tree Search (MCTS) with deep neural value and policy functions; Manhaeve et al. (2018) embed neural networks as callable predicates within probabilistic logic programs; and Mitchener et al. (2022) learn an Answer Set Programming (ASP) meta-policy via Inductive Logic Programming (ILP) over deep reinforcement learning (RL) sub-policies. In *Neuro||Symbolic*, Mao et al. (2019) parse visual scenes into symbolic representations via neural perception, Chaudhury et al. (2021) learn symbolic action policies via differentiable rule learning from neural text encodings,and Liang et al. (2016) learn to generate executable programs against a knowledge base from weak supervision. In *Neuro_{Symbolic}*, Andreas et al. (2015) assemble a composition of neural modules from a parsed layout. In *Neuro[Symbolic]*, Zheng et al. (2025b) embed a symbolic planner inside a large language model (LLM)-driven embodied agent, Li et al. (2024) combine LLMs with symbolic solvers for math data generation, and—as discussed below—agents that execute generated code as actions also fall in this paradigm. Graph-based approaches further combine graph neural networks (GNNs) with symbolic relational structures for knowledge-grounded reasoning (Hitzler et al., 2022; Sarker et al., 2021). In contrast, the FM Designer in ABCAGENT can implement *any* symbolic structure that best fits the task, and its use of functionally versatile programmatic code paired with a general-purpose coding agent makes it applicable to arbitrary agentic tasks.

Agents that generate and execute code. Early neural program execution (Reed & de Freitas, 2016) predates the LLM era; LLMs trained on code (Chen et al., 2021) then enabled applications such as language-model programs for embodied control (Liang et al., 2023), ambiguous-to-executable program synthesis (Zelikman et al., 2022) and program-aided reasoning, where the model writes a program and an interpreter executes it (Gao et al., 2022; Chen et al., 2022b). Yang et al. (2024b) survey how code empowers LLMs as agents; Xu et al. (2024a) show that jointly training on natural language and code yields stronger language agent backbones. CodeAct (Wang et al., 2024a) and REPL-Plan (Liu et al., 2025) direct agents to emit executable code snippets as actions, while skill-induction approaches (Wang et al., 2023; 2024b; 2025b; Zheng et al., 2025a; Yu et al., 2026) extract reusable programmatic subroutines from execution traces. Among these, CodeAct and skill-induction agents are the most relevant to ABCAGENT; like other *Neuro[Symbolic]* systems, all place the FM as the outer neural controller that invokes symbolic code. However, both employ code only in the *periphery*: CodeAct generates transient snippets discarded after each task, and skill-induction agents still rely on the FM to orchestrate top-level control flow, with neither lifting variable scoping to the full agent level. ABCAGENT differs by making the symbolic program the *central artifact* that fully specifies agent behavior under the recursive $NS = \text{NEURAL}[\text{SYMBOLIC}[NS]]$ paradigm—elevating code from a tool called by the FM to a persistent specification managed by it. Another line of work has the FM write the agent itself: Meta Agent Search programs new agent designs into an archive (Hu et al., 2024), AFlow searches over workflows represented as code (Zhang et al., 2024a), Trace optimizes agent code from execution feedback (Cheng et al., 2024), and DSPy compiles declarative LM pipelines into self-improving programs (Khattab et al., 2023). These methods search offline for a general agent design scored on a benchmark, whereas ABCAGENT authors a program for one task family and reuses it by rebinding its parameters. Lifting the constants of a worked example into reusable parameters is also the goal of abstraction learning over program libraries (Ellis et al., 2020; Grand et al., 2023; Stengel-Eskin et al., 2024). In concurrent work, Qi et al. (2026) likewise let a program rather than the FM govern an agent's control flow, and Pedada et al. (2026) reuse generated programs across sessions by refreshing their data without a model call. Our focus differs in that every ABCAGENT program declares re-bindable parameters and passes an executable check of its generality before it is stored, and in that we measure how the resulting savings amortize over task families on six benchmarks.

6 CONCLUSION

In this paper, we presented ABCAGENT, which specifies the behavior of an FM agent as a symbolic program that the FM designs and edits, thereby separating the design of agent behavior from its execution. A program ABCAGENT authored on one task can be reused across other tasks of the same family, so that when tasks recur with varied inputs, ABCAGENT is as accurate as a neural agent or more, at a fraction of its cost and latency. We believe that treating agent behavior as code opens several directions for future work, such as agent behavior engineering and generalization limitations of code-driven agents.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data sets, cleaning and reformatting datasets, implementing methods, designing and providing feedback on experiments, interpreting results, and refining hypotheses and the conceptual framework, including the decomposition of amortization into three mechanisms shown in Figure 3. The main hypotheses and ideas of this work are the authors' own. The variant tasks of augmented GAIA were largely generated with generative AI tools, which also helped correct annotation errors in the dataset; these corrections were validated by the authors. The tasks of WorkArena-CF were largely proposed by the authors, and generative AI tools were used to develop the code that implements them. We have not used generative AI tools to formulate mathematical claims, to provide critical ingredients for proving mathematical claims, or to assist in the writing of proofs, and translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, creating and modifying scientific figures, drafting parts of the paper, identifying and summarizing relevant literature, searching for information, formatting references, and brainstorming. We have reviewed all AI-assisted work. AI-generated code was tested and verified by one or two authors, depending on the portion of the work. AI-suggested references were checked against their arXiv or Crossref records and read by an author before being cited. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, and none of our benchmarks contain personal data: GAIA, WorkArena, GSM-Symbolic and τ^2-bench are public benchmarks, the users of τ^2-bench are simulated, and augmented GAIA and WorkArena-CF are derived from GAIA and WorkArena as described in Appendices C and D. All web and enterprise tasks run on the benchmarks' own sandboxed environments. Agents that act autonomously on the web or in enterprise software can cause harm when their actions are wrong, and ABCAGENT replays a stored program without a model call, which can repeat a wrong action across many inputs. We mitigate this by checking a program on task variants before it is stored, and we recommend that deployments keep such checks, together with human review of stored programs for consequential actions.

REPRODUCIBILITY STATEMENT

Section 2.2 describes ABCAGENT, and Section 3 and Appendix B describe the baselines, the model used by both systems, the evaluation protocols, how cost and time are measured, and the statistical tests. The construction of augmented GAIA and WorkArena-CF is described in Appendices C and D. We will release the implementation of ABCAGENT and both augmented benchmarks upon acceptance.

REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks, 2015. URL https://arxiv.org/abs/1511.02799.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Vaishak Belle, Michael Fisher, Alessandra Russo, Ekaterina Komendantskaya, and Alistair Nottle. Neuro-symbolic AI + agent systems: A first reflection on trends, opportunities and challenges. In *Autonomous Agents and Multiagent Systems. Best and Visionary Papers*, pp. 180–200. Springer, 2024. doi: 10.1007/978-3-031-56255-6_10. URL https://link.springer.com/chapter/10.1007/978-3-031-56255-6_10.

Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, et al. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, 36:12809–12844, 2024. doi: 10.1007/s00521-024-09960-z. URL https://link.springer.com/article/10.1007/s00521-024-09960-z.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks, 2024. URL https://arxiv.org/abs/2407.05291.

Subhajit Chaudhury, Prithviraj Sen, Masaki Ono, Daiki Kimura, Michiaki Tatsubori, and Asim Munawar. Neuro-symbolic approaches for text-based policy learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3073–3078, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.245. URL https://aclanthology.org/2021.emnlp-main.245/.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests, 2022a. URL https://arxiv.org/abs/2207.10397.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Tsong Yueh Chen, Shing Chi Cheung, and Siu Ming Yiu. Metamorphic testing: A new approach for generating next test cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, 1998. URL https://arxiv.org/abs/2002.12543. Posted to arXiv by the authors in 2020 as arXiv:2002.12543.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2022b. URL https://arxiv.org/abs/2211.12588.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next AutoDiff: Generative optimization with rich feedback, execution traces, and LLMs, 2024. URL https://arxiv.org/abs/2406.16218.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pp. 268–279, 2000. doi: 10.1145/351240.351266.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning, 2020. URL https://arxiv.org/abs/2006.08381.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972. doi: 10.1016/0004-3702(72)90051-3.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models, 2022. URL https://arxiv.org/abs/2211.10435.

Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023. doi: 10.1007/s10462-023-10448-w. URL https://link.springer.com/article/10.1007/s10462-023-10448-w.

Gabriel Grand, Lionel Wong, Maddy Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. LILO: Learning interpretable libraries by compressing and documenting code, 2023. URL https://arxiv.org/abs/2310.19791.

Aryan Gulati, Brando Miranda, Eric Chen, Emily Xia, Kai Fronsdal, Bruno Dumont, Elyas Obbad, and Sanmi Koyejo. Putnam-AXIOM: A functional and static benchmark for measuring higher level mathematical reasoning in LLMs. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2508.08292.

Kyle Hamilton, Aparna Nayak, Bojan Božić, and Luca Longo. Is neuro-symbolic AI meeting its promises in natural language processing? A structured review. *Semantic Web*, 2022. doi: 10.3233/SW-223228. URL https://journals.sagepub.com/doi/10.3233/SW-223228.

Pascal Hitzler, Aaron Eberhart, Monireh Ebrahimi, Md Kamruzzaman Sarker, and Lu Zhou. Neuro-symbolic approaches in artificial intelligence. *National Science Review*, 9(6):nwac035, 2022. doi: 10.1093/nsr/nwac035. URL https://academic.oup.com/nsr/article/9/6/nwac035/6542460.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2024. URL https://arxiv.org/abs/2408.08435.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, Jianye Hao, Kun Shao, and Jun Wang. Deep research agents: A systematic examination and roadmap, 2025. URL https://arxiv.org/abs/2506.18096.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation, 2025. URL https://hal.cs.princeton.edu/gaia. GAIA track; evaluated on the public 165-question validation split.

Henry A. Kautz. The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, 2022. doi: 10.1002/aaai.12036. URL https://onlinelibrary.wiley.com/doi/full/10.1002/aaai.12036.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines, 2023. URL https://arxiv.org/abs/2310.03714.

Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

Richard E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985. doi: 10.1016/0004-3702(85)90012-8.

John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986. doi: 10.1007/BF00116249.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning (ICML)*, 2018. URL https://arxiv.org/abs/1711.00350.

Zenan Li, Zhi Zhou, Yuan Yao, Yu-Feng Li, Chun Cao, Fan Yang, Xian Zhang, and Xiaoxing Ma. Neuro-symbolic data generation for math reasoning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/29d319f7c1513c9ecd81d3a6e9632a6e-Abstract-Conference.html.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision, 2016. URL https://arxiv.org/abs/1611.00020.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://arxiv.org/abs/2209.07753.

Anthony Zhe Liu, Xinhe Wang, Jacob Sansom, Yao Fu, Jongwook Choi, Sungryull Sohn, Jaekyeom Kim, and Honglak Lee. Interactive and expressive code-augmented planning with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20330–20354, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.994. URL https://aclanthology.org/2025.acl-long.994/.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

Tim Marcin. "was 1995 30 years ago?" google's ai overviews is having issues with a simple question, 2025. URL https://mashable.com/article/google-ai-1995-30-years-ago.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986. doi: 10.1007/BF00116250.

Ludovico Mitchener, David Tuckey, Matthew Crosby, and Alessandra Russo. Detect, understand, act: A neuro-symbolic hierarchical reinforcement learning framework. *Machine Learning*, 2022. doi: 10.1007/s10994-022-06142-7. URL https://link.springer.com/article/10.1007/s10994-022-06142-7.

Sanjana Pedada, Aditya Dhavala, and Neelraj Patil. Shared selective persistent memory for agentic LLM systems, 2026. URL https://arxiv.org/abs/2607.09493.

Junjia Qi, Zichuan Fu, Jingtong Gao, Wenlin Zhang, Hanyu Yan, Xian Wu, and Xiangyu Zhao. LLM-as-code: Agentic programming for agent harness, 2026. URL https://arxiv.org/abs/2606.15874.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.06279.

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends. *AI Communications*, 34(3):197–209, 2021. doi: 10.3233/AIC-210084. URL https://journals.sagepub.com/doi/abs/10.3233/AIC-210084.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, 2016. doi: 10.1109/TSE.2016.2532875.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/10.1126/science.aar6404.

Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. ReGAL: Refactoring programs to discover generalizable abstractions, 2024. URL https://arxiv.org/abs/2401.16467.

Andrea Tocchetti, Lorenzo Corti, Agathe Balayn, Mireia Yurrita, Philip Lippmann, Marco Brambilla, and Jie Yang. A.I. robustness: a human-centered perspective on technological challenges and opportunities. *ACM Computing Surveys*, 57(6):1–38, 2025. doi: 10.1145/3665926. URL https://dl.acm.org/doi/full/10.1145/3665926.

Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278, 1993. doi: 10.1023/A:1022686910523.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *ICML*, 2024a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=OJd3ayDDoF.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=lsAY6fWsog.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, Zhoujun Cheng, Siheng Zhao, Lingpeng Kong, Bailin Wang, Caiming Xiong, and Tao Yu. Lemur: Harmonizing natural language and code for language agents. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=hNhwSmtXRh.

Zhuoyan Xu, Zhenmei Shi, and Yingyu Liang. Do large language models have compositional ability? an investigation into limitations and scalability. In *First Conference on Language Modeling*, 2024b. URL https://openreview.net/forum?id=iI1CzEhEMU.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://arxiv.org/abs/2405.15793.

Ke Yang, Jiateng Liu, John Wu, Chaoqi Yang, Yi R. Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Yiquan Wang, Heng Ji, and Chengxiang Zhai. If LLM is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents, 2024b. URL https://arxiv.org/abs/2401.00812.

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

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. One skill, many websites: Learning generalizable skills through polymorphic abstraction. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=KdEsujyiSV.

Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions, 2022. URL https://arxiv.org/abs/2212.10561.

Hongwei Zhang, Ji Lu, Shiqing Jiang, Chenxiang Zhu, Li Xie, Chen Zhong, Haoran Chen, Yurui Zhu, Yongsheng Du, Yanqin Gao, Lingjun Huang, Baoli Wang, Fang Tan, and Peng Zou. Co-sight: Enhancing llm-based agents via conflict-aware meta-verification and trustworthy reasoning with structured facts. *arXiv preprint arXiv:2510.21557*, 2025. URL https://arxiv.org/abs/2510.21557.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation, 2024a. URL https://arxiv.org/abs/2410.10762.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13643–13658, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.737. URL https://aclanthology.org/2024.acl-long.737/.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025a.

Kaizhi Zheng, Kaiwen Zhou, Jing Gu, Yue Fan, Jialu Wang, Zonglin Di, Xuehai He, and Xin Eric Wang. JARVIS: A neuro-symbolic commonsense reasoning framework for conversational embodied agents. In *Proceedings of the 19th International Conference on Neurosymbolic Learning and Reasoning*, volume 284 of *Proceedings of Machine Learning Research*, pp. 647–673. PMLR, 2025b. URL https://proceedings.mlr.press/v284/zheng25a.html.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments, 2025c. URL https://arxiv.org/abs/2504.03160.

Zheng Zheng, Zenghui Zhou, Yinwang Xu, Daixu Ren, and Tsong Yueh Chen. Bidirectional empowerment of metamorphic testing and large language models: A systematic survey, 2026. URL https://arxiv.org/abs/2605.13898.

A LIMITATIONS

We acknowledge several limitations of our study, which we discuss in turn below.

Implementation backend and run-to-run variance. ABCAGENT is implemented on a single agent harness, and its components only take effect on that harness, so the results reported here might not transfer directly to other agent frameworks. Agent evaluations are also noisy from run to run: on the L1 tasks of augmented GAIA, 19.4% of tasks change outcome when only the order of sibling tasks changes, and on WorkArena-CF the raw success rate of an identical configuration ranges from 3 to 5 of 8 composites across repeated draws. We therefore report replicated or paired comparisons wherever possible, and treat single-run differences of a few points as uninformative.

Ground truth of augmented GAIA. The answers to the augmented GAIA tasks are produced by an LLM agent through majority voting and validated by the authors where errors were found (Appendix C), so it cannot be guaranteed that every label is correct, especially for tasks that turned out to be infeasible. Because both systems are scored against the same labels, label errors affect absolute success rates more than the comparison between them, but they may still mask small differences.

Contamination. GAIA and the GSM8K problems underlying GSM-Symbolic were public before the training cutoff of the model we use, so the model may have seen some of these tasks or their answers. Our comparisons are less exposed to this than absolute scores, since the neural agent and ABCAGENT share the same model, and the variants in augmented GAIA and WorkArena-CF are our own and have not been released; nevertheless, we cannot rule out that contamination favors one system over the other.

Single model family. All agents in our experiments use Claude Sonnet 4.6, both as the neural agent and as the FM Designer of ABCAGENT, so we have not established how the results change with other foundation models, including smaller models for which authoring a correct program might be harder.

Dependence of the efficiency gains on reuse. The cost and latency savings of ABCAGENT depend on how much its programs are reused, whether across the tasks of a family, over the records processed by a loop, or over repeated evaluation passes (Figure 3). Below the corresponding break-even point ABCAGENT is more expensive than the neural agent: it costs about $2.4\times$ as much on the one-off tasks of WorkArena L1, is less cost-efficient on WorkArena-CF at 10 records per task, and needs about 1.4 and 2.4 evaluation passes to pay back its authoring cost on the two τ^2-bench domains. Our efficiency claims therefore apply to recurring tasks, and might not be generally applicable to workloads in which most tasks are seen only once.

Sample size. Several of our benchmarks are small: WorkArena L1 has 33 tasks, WorkArena-CF has 8 loop composites of which 6 write records, and the hardest level of GAIA has 26 tasks. At these sizes paired tests have little power, so a non-significant difference there should be read as underpowered rather than as evidence of no effect, and per-level results as descriptive.

Taken together, our results should not be treated as a universal statement about programmatic agents, but rather as evidence on recurring tasks under one model and one harness; a broader evaluation across models, harnesses and task distributions warrants further investigation, which we leave to future work.

B EXPERIMENTAL DETAILS

This section describes the evaluation protocols behind the results in the main text.

Models and reference results. All systems use Claude Sonnet 4.6 and are model-matched within each benchmark; on τ^2-bench, the neural baseline is the benchmark's own stock agent, which is also the agent that public τ^2-bench leaderboards vary the model under. The reference rows give the strongest published result we are aware of, which is not directly comparable to our rows: the

WorkArena L1 protocol samples 10 seeds per task under a 15-step cap where we run three replicates per task, the GSM-Symbolic figure is obtained by direct prompting rather than by a tool-using agent, the τ^2-bench figures use `o4-mini`, the strongest of the four models evaluated by Barres et al. (2025), under the stock agent, and the GAIA figure is the strongest generalist agent on the HAL leaderboard (Kapoor et al., 2025), which evaluates the same 165-question validation split, whereas the public GAIA leaderboard reports on the test split.

Scoring. On augmented GAIA, both systems are compared on the same 378 child tasks; ABCAGENT additionally solves the 165 parent tasks from which programs are reused, which the neural agent does not need. WorkArena-CF comprises 42 task configurations, namely 10 non-loop tasks and 8 loop composites at four loop lengths ($N \in \{3, 10, 30, 100\}$); in Table 1, all 42 configurations are scored by task success, with loop composites adjudicated against the live instance rows, the four repeated draws at $N = 3$ averaged, and re-run tasks superseding their original run, while the fraction of records written correctly is reported per loop length. Three WorkArena-CF runs flagged by an internal mechanism check, rather than by an environment fault, are kept with reduced denominators. On τ^2-bench, each task is run for four trials; Pass^1 is the success rate of a single trial, and Pass^4 counts a task as solved only if all four trials succeed. On WorkArena L1, both systems run with the same three seeds, and we compare the 31 tasks whose instances match in every replicate; the other two draw identifiers that the seed does not control.

Cost and latency. Costs are computed from the token usage of each model call, including the short calls that bind a new task onto a stored program and verify its result; a task solved by cheap rebinding makes two such calls and no others, so we call it *free* in the sense that it runs no authoring loop. On GSM-Symbolic these calls add \$18.01 over 5,000 tasks. Their cost was not recorded on augmented GAIA, base GAIA and WorkArena-CF; from the calls recorded on GSM-Symbolic, they would lower the cost efficiency of ABCAGENT on augmented GAIA by at most about 3%, only widen the neural agent's lead on base GAIA, and on WorkArena-CF could offset ABCAGENT's small cost advantage entirely, so we do not claim a cost difference on WorkArena-CF. Their time was not logged, so we measured it by rerunning the binding and verification of 20 randomly sampled free GSM-Symbolic tasks, which take 3.6 s per task (95% CI [3.2, 4.0] s), and added it to ABCAGENT's time; this lowers its speedup on GSM-Symbolic from 8.2× to 5.2× (5.0–5.4× over the CI). These calls cost little but take about as long as a short FM call each, which is why they reduce the speedup much more than the cost efficiency. Because the rerun was serial whereas the original run was concurrent, 5.2× is more likely an overestimate than an underestimate. On augmented GAIA, adding the binding and verification calls, measured at 5.4 s per accepted and 4.7 s per declined child, lowers the speedup only from 1.34× to 1.33×. Time counts the model calls that completed; calls interrupted by the evaluation infrastructure and retried are excluded on every benchmark, since they reflect the environment rather than the method, although their tokens remain in the cost. On GAIA and WorkArena-CF, the FM calls of the variant check were not timed either, which slightly favors ABCAGENT in their speedups. Time also excludes script execution and the variant check outside model calls; measured by wall clock from a task's first model call to its score, which includes both as well as retries, ABCAGENT is slower than the neural agent on augmented GAIA (0.64×) and GAIA (0.46×). Cost efficiency and speedup are the neural agent's cost and time divided by those of ABCAGENT, computed over the tasks on which both systems produced a result without an error, whether or not that result was correct; this excludes three of the 58 paired WorkArena-CF runs, which have no scored verdict, and nothing on the other benchmarks. On GSM-Symbolic, the neural agent's per-task results were not preserved, so both ratios are computed over all tasks. Figure 3 uses the same basis; Table 2 compares settings over all tasks except on WorkArena-CF (Appendix H). On augmented GAIA, costs are paired on the 378 child tasks, and the 165 parent tasks cost ABCAGENT a further \$257.90 and 70.0 hours, so no paired total over whole families exists. On τ^2-bench, cost and time count the agent alone and exclude the benchmark's simulated user, which runs on a different model provider and is driven identically by both systems; the one-off authoring cost is likewise agent-only, so that ABCAGENT pays back its authoring cost after 1.44 evaluation passes on telecom and 2.36 on airline. On WorkArena-CF, time is the agent's compute time over its model calls, plus the executor's replay time for ABCAGENT.

Statistical tests. We use the exact McNemar test for paired binary outcomes on GAIA, augmented GAIA and the per-task Pass^4 outcomes of τ^2-bench, the Wilcoxon signed-rank test over tasks for the paired Pass^1 difference on τ^2-bench, the sign test over tasks on WorkArena L1, and the sign

test over the 42 configurations of WorkArena-CF. Throughout the paper, we treat a difference as statistically significant at $p < 0.01$. On GSM-Symbolic, per-task outcomes of the neural agent were not preserved, so we use an unpaired two-proportion test, which is conservative. Small benchmarks and levels have little power, so the per-level results are descriptive.

Component ablations. The GSM-Symbolic ablation uses the 1000 tasks of the 20 templates with the lowest identifiers; its free share and cost are measured on this subset, where 96.0% of tasks are free against 92.6% on the full set, which is why its cost efficiency exceeds that in Table 1. The augmented GAIA setting without reuse ran at an earlier code revision than the other augmented GAIA settings. In the WorkArena-CF setting with family reuse, cheap rebinding was attempted on every task but never took effect: the loop programs take their per-record values from the task's data rather than from declared parameters, and the rebinding step neither maps onto that data nor supplies it, so this setting behaves as family reuse with the variant check alone. The WorkArena-CF settings with the executor loop disabled are valid only inside this decomposition, and the setting with the loop but without the variant check lost two tasks to infrastructure faults, so its denominator is 150 rather than 180 records.

Program transfer. On GSM-Symbolic, each family's program is authored in instance order, so a program saved at instance k has seen instances 0 to k; applying it to a later instance measures forward generalization and to an earlier one backward compatibility, over 38,503 substitute-and-execute trials without any model call. In 3.85% of pairs the stored program's parameter interface does not fit the target at all; we report these separately, so that coverage is 97.82% over the pairs where the program applies and 96.7% if every such pair is counted as a failure, and 310 of the 315 execution errors are interface mismatches. Of the 100 families, 91 reach complete forward coverage; of the remaining nine, four fail only on a single instance that no program of the family ever solved, four show diffuse variation in program quality, and one keeps changing its parameter interface. On augmented GAIA, the control is the same configuration with reuse disabled, and because the backward-compatibility requirement and the family regression cases are only added when family context exists, the control isolates this mechanism. For the GSM-Symbolic comparison in Table 3, the reuse side is the full configuration of Table 1 on all 100 families, taking at each instance k the version of the family's program in use at k, and the control is the configuration without reuse, whose programs are authored for a single instance at five randomly chosen k per family. Each control program and the corresponding stored program are applied to the same 30 randomly chosen other instances of the family: one FM call binds the values of the target instance to the program's own parameters, and the program is then executed without further model calls and scored against the target's answer, giving 15,000 pairs. We test significance with the Wilcoxon signed-rank test over the 100 families, since pairs within a family are not independent; the family-level difference is +12.5 points (95% bootstrap CI $[8.0, 17.4]$). The two sides were run at different code revisions, and the full configuration passes a program on to later instances only if it solved its own task; restricted to targets the stored program has not seen, the difference is still +11.6 points. Binding parameters by name instead, without an FM call, fails on 90% of the control's pairs, since independently authored programs name their parameters differently; where the names do fit, its forward coverage is 99.8% (542 of 543 pairs), and Table 8 reports both procedures.

Replay profile. We also profile the per-record replay time of ABCAGENT alone, since the neural agent has no executor to profile; the pooled median slope is +0.00029 per record, and at $N = 100$ the per-record time decreases from 1.12× the task mean in the first fifth of records to 0.93× in the last.

Adaptation on τ^2-telecom. In Table 4, the second row gives the neural agent the same task transcripts that ABCAGENT authors from, and the third row executes an authored policy for 96.2% of its actions. Table 4 comes from an earlier run of the same protocol, on the same 114 tasks with four trials each, than the runs behind Table 1 and Table 7, which is why its first and third rows differ slightly from the Pass^1 values there (0.603 against 0.608, and 0.789 against 0.816); its first two rows also ran on a different agent harness than its last two.

Case study. The programs in Figure 6 are quoted verbatim; in (a), instances `i01` and `i39` of the same GSM-Symbolic family differ only in the highlighted constants.

C AUGMENTED GAIA DATASET

C.1 CONSTRUCTION

The validation set of the GAIA dataset (Mialon et al., 2023) consists of 165 unique computer agent tasks. It is separated into 3 levels of difficulty, with 53, 86, and 26 tasks in level 1, 2, and 3. To assess the efficacy of our neuro-symbolic approach over this type of tasks, we augment this validation set with 129, 204, and 45 new data points for level 1, 2, and 3. We call this set of 378 data points the **Augmented GAIA dataset**. Constructing parametric variants of a fixed benchmark to separate reasoning from memorization follows GSM-Symbolic (Mirzadeh et al., 2024) and Putnam-AXIOM (Gulati et al., 2025).

The Augmented GAIA dataset is created through the followup pipeline.

- **Task proposer** For each data point in the original GAIA validation set, we ask an LLM to propose 3 alternative tasks that are very similar to the original task in format and potential solution, but nonetheless different in a small way. For example, for an original task of "What is the oldest Blu-Ray recorded in this spreadsheet?", the proposer is encouraged to propose tasks such as "What is the newest Blu-Ray recorded in this spreadsheet?", "What is the oldest DVD recorded in this spreadsheet?", or "What is the second oldest Blu-Ray recorded in this spreadsheet?" The proposer has access to the original question, answer, and a copy or textual representation of the file needed for the task, depending on the file types. We call the original data point the **parent** data point and all the proposed alternatives its **children** data points.
- **Ground Truth Generator** We use Co-Sight (Zhang et al., 2025), an LLM agent design that achieved SOTA performance over the test dataset of GAIA at the time of experiment[2], to attempt to answer the children tasks generated, with necessary files inherited from their parents. We run this answer generation process for each child task 3 times and conduct majority voting the collect the final answer that we label as ground truth. Due to the expected failures of the Co-Sight agent and infeasibility of some of the generated tasks, we do not consider this ground truth to be reliable. However, we consider them to be a good approximation of the true answer if the task is feasible, and they serve our purpose of evaluating the reusability of generated codes by the Neuro-Symbolic agent.
- **Filterer** As a final step, we use a simple, heuristic script to remove any data points that Co-Sight failed to find a meaningful answer to. The final result is the Augmented GAIA dataset.

C.2 ADDITIONAL RESULTS

Table 5 breaks down success rates on GAIA and augmented GAIA by difficulty level.

	GAIA				Augmented GAIA			
System	L1 (53)	L2 (86)	L3 (26)	all (165)	L1 (129)	L2 (204)	L3 (45)	all (378)
Reference (HAL)	82.1	72.7	*65.4*	74.6	—	—	—	—
Neural agent	*86.8*	*75.6*	57.7	*76.4*	*62.0*	*58.3*	*57.8*	*59.5*
ABCAGENT	*86.8*	72.1	46.2	72.7	58.9	57.4	51.1	57.1
McNemar p-value	1.000	0.648	0.453	0.377	0.481	0.851	0.508	0.281

Table 5: Task success rate (%) on GAIA and augmented GAIA by difficulty level, with the number of tasks in parentheses. The reference row is the strongest generalist agent on the HAL leaderboard (Kapoor et al., 2025), p is the exact McNemar test, and italics mark the strongest system in each column; no level differs significantly.

[2]This is no longer the case. For the GAIA test set leaderboard, see https://huggingface.co/spaces/gaia-benchmark/leaderboard

D AUGMENTED WORKARENA DATASET

D.1 CONSTRUCTION

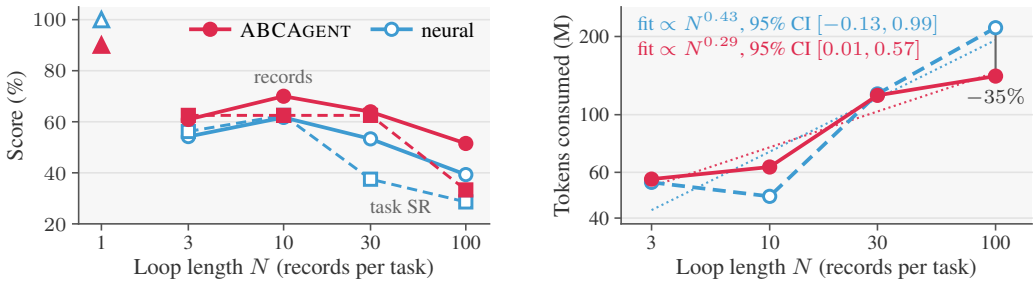
To better evaluate the Neuro-Symbolic agent, we create a new dataset focusing on complex enterprise tasks. In realistic enterprise tasks, human often need to perform multiple similar tasks, decide to take different task branches based on certain conditions and/or compare field values to find inconsistency. One example task is *Retrieve a list of products and for each of them, check the sale amount, and return the products if their sale amount are less than 100 in this month*. As no existing such benchmark exists, we extend the WorkArena benchmark with 18 new tasks with loop, conditional branch and/or comparison. Each new task is built using the original WorkArena L1 tasks with the new task structure. Details are shown in Table 6. Similar to the original WorkArena task, each complex task can be instantiated from a template with different parameters. For example, *CreateMultipleUserTask* can be instantiated to create 2-10 different users. The average length of the golden traces is 42 steps, significantly longer than the original L1 tasks. Loop tasks are the longest, with 81 steps on average, while conditional / comparison tasks are shorter, ranging between 14-25 steps.

Task Type	**Task Names**
Loop tasks	CreateMultipleChangeRequestTask CreateMultipleHardwareAssetTask CreateMultipleIncidentTask CreateMultipleProblemTask CreateMultipleUserTask DeleteMultipleUserTask OffBoardMultipleUserTask OrderMultipleDevicesTask
Conditional tasks	EditChangeRequestScheduleConditionalTask EditHardwareConditionalTask EditIncidentConditionalTask EditProblemConditionalTask OnBoardUserConditionalTask
Comparison tasks	CompareChangeRequestPriority CompareExpenseLineAmountTask CompareHardwareDate CompareIncidentBusinessDuration CompareProblemPriority

Table 6: Complex Task types and corresponding task names

D.2 ADDITIONAL RESULTS

Figure 4 shows accuracy and token consumption on WorkArena-CF at each loop length. Token consumption grows sublinearly with the number of records for both systems, roughly as $N^{0.29}$ for ABCAGENT (95% CI $[0.01, 0.57]$) and $N^{0.43}$ for the neural agent ($[-0.13, 0.99]$), so that ABCAGENT uses 35% fewer tokens at $N = 100$; with four loop lengths, the two exponents are not statistically distinguishable.



(a) Accuracy

(b) Tokens

Figure 4: Accuracy and token consumption on WorkArena-CF as a function of loop length N, with the legend in (a) applying to both panels. In (a), circles give the fraction of records written correctly and squares the task success rate, with the non-loop tasks at $N = 1$; in (b), the dotted lines are least-squares power-law fits, both sublinear in N, labeled with 95% confidence intervals on their exponents.

E ADDITIONAL RESULTS ON τ^2-BENCH

Table 7 reports both Pass^1 and Pass^4 on the two τ^2-bench domains, together with the paired difference between ABCAGENT and the stock agent.

	telecom ($n=114$)		airline ($n=50$)	
System	Pass^1	Pass^4	Pass^1	Pass^4
Reference (Barres et al., 2025)	0.59	0.29	0.59	0.42
Stock agent	0.6075	0.4737	*0.6850*	0.5800
ABCAGENT	**0.8158**	**0.7193**	*0.6850*	*0.6000*
paired Δ	+**0.208** (46/16/52)		+0.000 (8/8/34)	
Wilcoxon p	**0.0001**		0.958	

Table 7: Performance on the two τ^2-bench domains, with the benchmark's own stock agent as the model-matched neural baseline. The paired row gives the mean per-task difference, ABCAGENT minus stock, with win, loss and tie counts; the reference row is the strongest model evaluated by Barres et al. (2025) under the same stock agent.

F TRANSFER WITHIN A FAMILY

Table 8 extends Table 3 with further transfer settings, together with the transfer of a frozen augmented GAIA parent to its children. Figure 5 shows, on all 100 GSM-Symbolic families, how coverage of a family's other instances changes as the program sees more of them.

Transfer setting	pairs	Reuse	Control	p
Augmented GAIA				
All pairs	572	**0.388**	0.292	**2.2e-5**
Forward ($j > i$)	235	0.332	0.302	0.410
Backward ($j < i$)	239	**0.469**	0.305	**3.5e-5**
Backward, target known-passing	145	**0.703**	0.462	**5.1e-5**
Backward, rewritten & known-passing	103	**0.718**	0.388	**1.2e-6**
GSM-Symbolic, parameters bound by name				
Forward ($j > k$)	5 572	**0.966**	0.097	$< \mathbf{10^{-4}}$
Backward ($j < k$)	2 252	**0.972**	0.090	$< \mathbf{10^{-4}}$
Frozen-parent transfer, augmented GAIA, reuse only				
Unseen children, FM-assisted rebind	303	0.271	—	—
Unseen children, own parameters	303	0.241	—	—
Seen parents, own parameters	75	0.733	—	—
FM-assisted rebinding, each setting's own programs				
GSM-Symbolic, forward ($j > k$)	7 640	**0.958**	0.842	$< \mathbf{10^{-4}}$
GSM-Symbolic, backward ($j < k$)	7 360	**0.963**	0.826	$< \mathbf{10^{-4}}$
Augmented GAIA, siblings	234	0.303	0.235	0.020

Table 8: Transfer of a stored program to other tasks of the same family, to tasks it has not seen (forward) and to tasks it was authored on earlier (backward). Bold marks differences from the no-reuse control that are significant at $p < 0.01$, by McNemar's test on augmented GAIA and by a sign test over the 20 families on GSM-Symbolic, where a program whose parameters cannot be bound to the target counts as a failure. The frozen-parent rows bind a child task either with one FM call (FM-assisted) or with the parameter values from the child's own program, and the last block binds every pair with one FM call, on GSM-Symbolic over 100 families with the full configuration's programs, tested with the Wilcoxon test over families (Appendix B), and on augmented GAIA from each family's first child solved in both arms to its siblings.

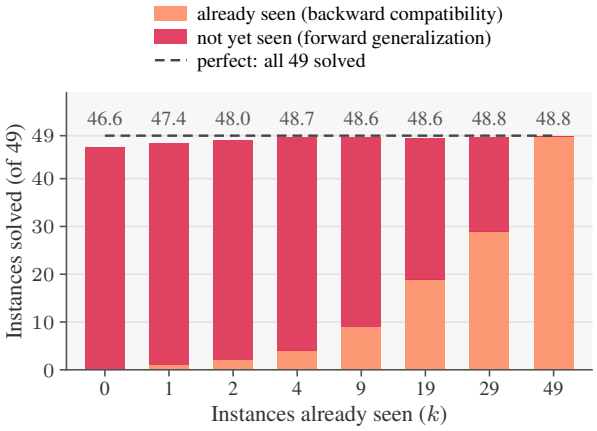


Figure 5: Instances of a GSM-Symbolic family that a program frozen after seeing k of its 50 instances solves among the other 49, split into instances it has already seen and instances it has not. Each bar's total is shown above it, and the dashed line marks all 49.

G Additional Analysis

How often programs defer to the FM, and how deeply. We find that programs defer a step to an FM sub-call only where the step resists code: 7.1% of stored GAIA programs defer such a step (251 of 3,541, steady at 6.3–10.3% across 16 runs), whereas none of the 4,999 GSM-Symbolic programs do. When a GAIA program does defer a step, it almost always uses the generic `delegate` call rather than a typed perception helper (304 of 313 calls), so the deferred step is an unconstrained FM sub-call. Although the design allows a delegated call to author a program of its own, nesting beyond

one level is rare in practice: it happened once in our runs, and a third level never occurred. Among GAIA tasks that defer a step, runs whose `delegate` call returned a result pass 75.4% of the time, against 58.5% where it failed ($p = 0.022$). We do note, however, that failed calls are plausibly harder sub-questions, and that the within-task comparison has only two discordant pairs, so the effect of a successful call lies between 0 and +17 points. These calls also failed often in our runs (2,231 failures, over half of them timeouts), and since a failed call returns an empty result, this biases the reported GAIA numbers against ABCAGENT if anything.

Failures on WorkArena L1. Of the 35 failures of ABCAGENT on the paired WorkArena L1 tasks, classified by the benchmark's own validator messages, 22 end with the browser on a page other than the one the validator reads, including 19 of the 28 that the neural agent solved: 14 catalog orders never reach the checkout confirmation that carries the request identifier, which appears in none of them but in all 13 successful orders of ABCAGENT, and 8 chart and sorting tasks leave the target page. The remaining failures are a wrong interface state (6), wrong record fields (4, three of which the neural agent also gets wrong) and wrong answers (3); none are execution errors.

Errors over loop length. Errors on WorkArena-CF do not accumulate with loop length: 87.5–100% of field errors fall in a field that is wrong on at least 80% of the records requesting it. Hence, what grows with N is not the chance of a new mistake on each record, but the number of records that a single authoring defect is replayed on.

Inspectability. Because ABCAGENT stores the program it executes, the reason a program does or does not transfer is legible in the program itself (Figure 6). In a GSM-Symbolic family, all instances share one derivation, so a sibling is solved by rebinding a handful of constants; in an augmented GAIA family built around Boggle boards, a new board calls for a new search rather than a new constant, which is why rebinding fails there. The neural agent, in contrast, leaves no such artifact to inspect.

(a) Rebinding

```
"""
Variants this script handles by changing
parameters in Section A:
 - Variant 1: 8 remoras instead of 7
     -> set NUM_REMORAS = 8
 - Variant 2: 84-inch remoras instead of 72
     -> set REMORA_LENGTH_INCHES = 84
"""
# Section A - parameters
WHALE_LENGTH_FEET    = 10   # sibling i39: 80
NUM_REMORAS          = 8
REMORA_LENGTH_INCHES = 3    # sibling i39: 30

# Section B - derivation (byte-identical
# across all 50 instances of this family)
combined_inches = (NUM_REMORAS
                   * REMORA_LENGTH_INCHES)
combined_feet   = combined_inches / 12
result = (combined_feet
          / WHALE_LENGTH_FEET) * 100
if result == int(result):
    result = int(result)
print(f"<final_answer>{result}</final_answer>")
```

(b) Misbinding

```
'''
Variants this script handles by changing
parameters in Section A:
 - As-is (this task): Find the SECOND longest
     word on board ABRL/EITE/IONS/FPEI
 - Variant 2: Find the longest word
     (first tier) -> set LENGTH_RANK = 1
'''
BOARD_STRING = "ABRLEITEIONSFPEI"
LENGTH_RANK  = 1   # task asks for the SECOND
```

Predicted `briniest`; gold `pointrel`. The program states its own contract in the header and then contradicts it in the parameter block. The error is one integer, and it is visible without running anything.

The neural agent solved the same task in 20 tool calls and left no program behind. Its entire artifact on the task in panel (a)'s family was:

```
print("unanswerable")
```

Figure 6: Programs authored by ABCAGENT, taken verbatim from run artifacts. In (a), all instances of a GSM-Symbolic family share one derivation, so a sibling is solved by rebinding the highlighted constants; in (b), rebinding fails on an augmented GAIA family, because a new Boggle board requires a new search rather than a new constant.

H Component Ablations by Benchmark

Table 2 restricts augmented GAIA to its 129 L1 child tasks, and uses the 8 loop composites of WorkArena-CF at $N = 30$ together with its 10 non-loop tasks. On WorkArena-CF, the success rate is computed over the tasks with a score, and cost efficiency over those scored for both systems; the non-loop tasks of the setting without reuse and rebinding coincide with the full configuration by construction, since a single task has no family to reuse, and its loop tasks run without the executor loop (Table 10). The setting without the variant check is cheaper than the neural agent only because its loop tasks commit records with wrong field values, none of them correct, and it also solves fewer tasks, so its cost efficiency should be read together with its success rate and records; all four settings solve every non-loop task. Cost efficiency is colored as in Table 1. On GSM-Symbolic, every setting in Table 2 and Table 9 is run on the same 1,000 tasks from 20 templates and costed from token usage in a single pass; the full configuration is the corresponding slice of the 5,000-task run in Table 1, run at a different code revision, and its binding and verification calls, which were not logged per task, are estimated at \$3.62 from the per-call cost of that run. τ^2-bench is omitted, since it has no task families to rebind across and its authored policy has no parameters for the variant check to vary; a task that does require authoring costs about the same for both systems, \$1.610 against \$1.592 on augmented GAIA. Table 2 keeps only the component settings that were measured on more than one benchmark. This section reports the settings that were measured on a single benchmark, which isolate one component at a time on GSM-Symbolic (Table 9) and the interaction between the variant check and the executor loop on WorkArena-CF (Table 10).

Setting	SR	Free	\$/task	Cost eff.
Neural agent	99.7	0.0%	0.1362	1.00×
ABCAgent, no components	*99.8*	0.0%	0.1408	0.97×
+ family reuse	*100.0*	0.0%	0.2138	0.64×
+ variant check	99.7	0.0%	0.2155	0.63×
+ cheap rebind	99.1	96.0%	0.0114	11.91×

Table 9: Component ablation on GSM-Symbolic over 1000 tasks from 20 templates, adding components cumulatively. Cheap rebinding is the only component that reduces cost, and accuracy is near ceiling for every setting.

	executor loop off	executor loop on
variant check off	*64.4*	0.0‡
variant check on	63.3	63.9⋆
neural agent	53.3	

Table 10: Records written correctly (%) on WorkArena-CF at $N = 30$, crossing the variant check with the executor loop, without family reuse. ⋆ is the configuration reported as ABCAgent; ‡without the variant check, the replayed programs commit records with wrong field values, including one program that ignores its per-record input entirely and submits the same values on every record.