AGENTIC REASONING COMPILATION: FROM TRACES TO EXECUTABLE REPRESENTATIONS FOR SMALL LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model reasoning combines semantic interpretation with logical and numerical computation. Errors may therefore arise before computation begins: a model selects the wrong entities, omits a constraint, or confuses current state with history. We present agentic reasoning compilation, which combines external executors for supported computations with an orchestration pipeline guided by solution traces and execution feedback. The resulting system couples reusable typed representations and admission checks with deterministic execution, then uses planning, verification, and answer selection to handle remaining inputs. Reusable capabilities and control rules organize inference while language-model weights remain fixed. On the full BBEH suite, the system raises micro accuracy from 48.4% to 64.9% for `gpt-5.6-luna` and from 47.2% to 64.0% for `qwen3.7-flash`. Their adjusted harmonic means reach 47.7 and 47.9, exceeding `gpt-5.6-sol`'s 37.2 at 67.2% and 97.7% lower estimated inference cost. A 100-item component comparison shows that orchestration preserves all 27 executor successes and solves 12 additional items. These results support a division of labor in which external execution supplies exact computation and orchestration extends reasoning coverage.

1 INTRODUCTION

Small language models can fail on reasoning problems whose final computation is elementary. We study 20 examples randomly sampled from BBEH-mini to identify recurring reasoning failures. One example asks for the total number of cars and musical instruments owned by the narrator. The narrative also mentions relatives' possessions, mobile phones, and earlier purchases. The `gpt-5.6-sol` solution reaches the reference answer of 1,240, while `gpt-5.4-nano` answers 1,230 and mixes the requested categories. The difficulty lies in connecting semantic interpretation to computation: whose possessions count, which categories are requested, and how historical purchases relate to current holdings.

An explicit ledger makes these decisions available to an external executor. Parsing recovers 35 first-person records, comprising 11 car records, 12 musical-instrument records, and 12 mobile-phone records. Selecting the requested categories yields 629 cars and 611 instruments, giving $629 + 611 = 1{,}240$. Once the problem state is represented correctly, deterministic execution carries out the sum. This separation also applies to logical inference, constraint satisfaction, and state tracking: a model can organize the reasoning while an executor performs supported operations on an explicit representation.

External computation and orchestration play complementary roles. Executors perform repeatable calculations within their supported semantics, while orchestration pipelines coordinate problem interpretation, exploration of alternative plans, verification, and answer selection when a complete executable solution is unavailable. Solution traces reveal useful problem decompositions, and execution outcomes expose missing constraints and ineffective reasoning steps. Recurring decompositions suggest that some reasoning operations may be reused across problems rather than generated anew for each one. We therefore ask whether these operations can be compiled into reusable system artifacts that enable lightweight backends to solve such problems more reliably.

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.

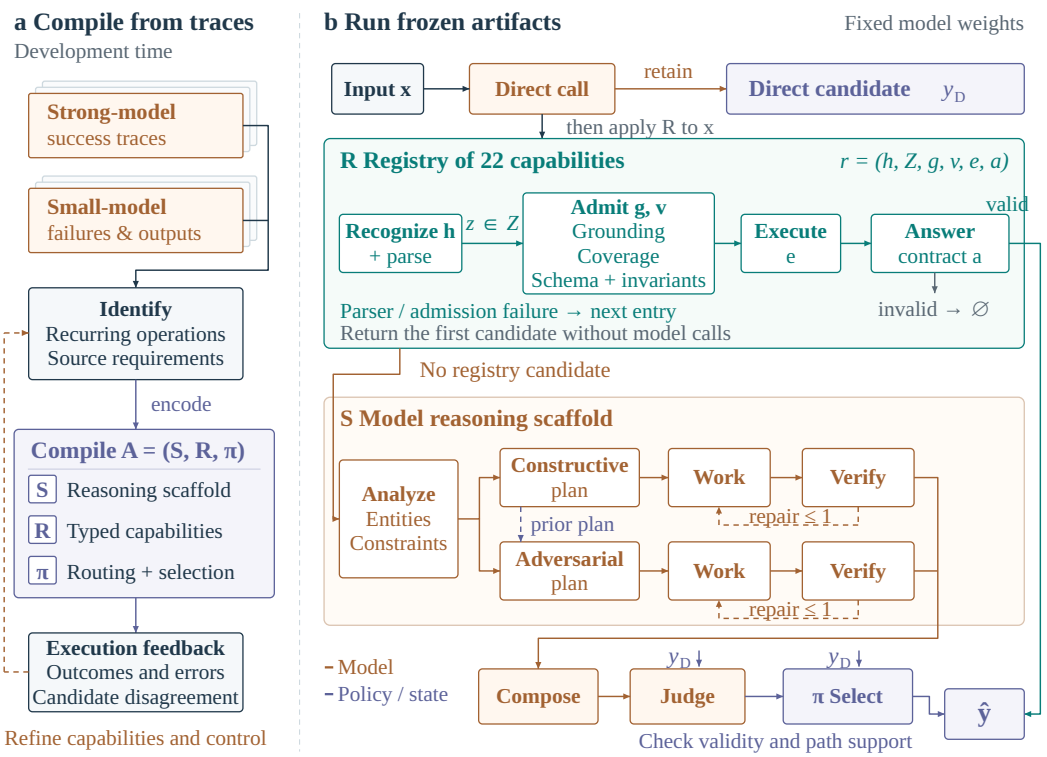

Figure 1: Agentic reasoning compilation. (a) Strong-model success traces, small-model failure traces, and execution feedback guide construction of the scaffold S, capability registry R, and routing and selection policy π. (b) With these artifacts frozen and model weights fixed, inference retains a Direct candidate y_D before applying the ordered registry. A valid executable answer is returned; a failed final answer contract yields an empty answer. Registry abstention invokes constructive and adversarial reasoning paths with verification and at most one repair per path. Composition, judgment, and selection use path support and answer validity, with y_D available to both judge and selector.

Existing approaches establish the basis for this division of labor. Chain-of-thought prompting exposes intermediate reasoning (Wei et al., 2022); program-aided reasoning delegates computation to an interpreter (Gao et al., 2023; Chen et al., 2023); and stronger models can construct reusable tools for lighter models to invoke (Cai et al., 2024). Applying these ideas across varied inputs requires an interface that preserves the information needed by each computation and a policy for handling inputs beyond the available executors.

We introduce *agentic reasoning compilation*, a framework for constructing executable capabilities and an orchestration pipeline using successful strong-model traces, small-model failures, and execution feedback. Each capability couples a typed representation, admission checks, a deterministic executor, and an answer contract. Figure 1 separates trace-guided construction (a) from inference with the frozen artifacts (b). At inference, the system retains a Direct candidate, attempts registry execution, and invokes bounded planning, verification, repair, and answer selection when the registry abstains. The reusable artifacts comprise representations, executable implementations, and control rules; model weights remain fixed.

The contributions are:

- **Reusable external computation.** We compile recurring logical and numerical operations into typed representations and executors, with admission checks connecting source inputs to supported computations.
- **Trace-guided orchestration.** We construct a reusable pipeline that combines executor outputs with model planning, verification, and answer selection, extending coverage beyond the executor registry.

- **Evidence for the combined system.** Full-suite BBEH comparisons show gains across three backends, and an evaluation excluding the construction and refinement samples shows a 14.20 pp accuracy gain for `gpt-5.6-luna`. Component comparisons explain the complementary solution coverage of execution and orchestration. MuSR, LiveBench Reasoning, and cost analyses characterize performance across tasks and inference resources.

2 Related Work

Constructing language-model pipelines. DSPy optimizes composable language-model programs from examples and a metric (Khattab et al., 2024). ADAS searches over agent designs expressed in code, and AFlow searches over executable workflows (Hu et al., 2025; Zhang et al., 2025). Our framework uses solution traces and execution feedback to guide the construction of reusable execution interfaces and orchestration. The retained system combines external computation with structured planning, verification, and answer selection.

Executable reasoning and reusable tools. PAL and Program of Thoughts delegate computation to executable programs (Gao et al., 2023; Chen et al., 2023); Faithful Chain-of-Thought separates translation from deterministic solving (Lyu et al., 2023). LATM constructs reusable tools with a stronger model for use by a lighter model (Cai et al., 2024). LLMCompiler schedules function calls according to their dependencies (Kim et al., 2024). We couple reusable executors with an orchestration pipeline that handles inputs outside their supported scope. Typed representations and admission checks connect each input to the appropriate computation, while the pipeline coordinates the remaining model reasoning.

Verification and neuro-symbolic reasoning. ReAct interleaves reasoning and actions, and CRITIC uses external tools to guide correction (Yao et al., 2023; Gou et al., 2024). Studies of intrinsic self-correction motivate the use of external checks alongside model judgments (Huang et al., 2024). Recent work extends symbolic reasoning in different directions: ARGOS supplies missing commonsense premises through abductive search, while Mixture-of-Thought trains and combines natural-language and symbolic reasoning modes (Cotnareanu et al., 2026; Zheng et al., 2026). Our system routes inputs through capability-specific admission checks and uses bounded model-driven fallback for rejected inputs. Agreement between the two fallback paths contributes to answer selection.

3 Agentic Reasoning Compilation

3.1 External execution and trace-guided orchestration

The construction procedure produces an artifact $A = (S, R, \pi)$: an orchestration scaffold S, an executable-capability registry R, and a routing and answer-selection policy π. As shown in Figure 1a, traces reveal recurring operations and their source requirements, while execution outcomes, errors, and candidate disagreements guide refinement. The resulting artifacts are frozen for inference (Figure 1b). The scaffold organizes entities, constraints, intermediate claims, and verification outcomes; the registry performs supported computations; and the policy governs routing and final answer selection.

A capability is a tuple $r = (h, \mathcal{Z}, g, v, e, a)$. The recognizer h identifies a supported operation, and $\mathcal{Z}$ defines its typed intermediate representation (IR). The checks g and v enforce source grounding, coverage, schema validity, and semantic invariants. The executor e computes the result, and a checks its answer contract. Parsing supplies $z \in \mathcal{Z}$ for registry execution. Together, these elements make the interface between interpretation and computation explicit.

3.2 Constructing reusable capabilities and control structure

Successful `gpt-5.6-sol` traces and small-model failures illustrate reasoning decompositions and execution requirements. Development uses an initial 20-example BBEH-mini sample for pipeline construction and a separate 100-example sample for capability refinement. The resulting control structure and capability registry are reused at inference. Appendix A distinguishes these development samples from the component and backend-reuse test sets.

The framework organizes construction around three operations: identifying recurring computations and their required source information; expressing these computations through typed representations, parsers, executors, and checks; and using execution outcomes to guide refinement. Accepted answers, rejected inputs, incorrect computations, and candidate disagreements provide feedback about the capabilities and their orchestration. The registry contains 22 entries spanning expression evaluation, state replay, constraints, logical inference, aggregation, and supported language or spatial transformations.

The orchestration scaffold gives these capabilities a reusable context. It requires plans to identify relevant entities and constraints, workers to carry out reasoning steps, and verifiers to inspect intermediate claims before composition and selection. This structure preserves the distinction between an executable result and a model's judgment of a reasoning path. The reusable artifact therefore includes both what can be computed externally and how model-driven reasoning proceeds when the registry cannot solve the complete input.

Worked case: ownership counting. The ownership example instantiates the registry route in Figure 1b. The representation is a ledger of quantity, item, and category, with first-person ownership enforced during parsing. The parser checks that all recognized first-person ownership clauses are consumed, rejects unknown item categories, and requires both requested categories. It treats parenthetical purchase histories as explanations of the stated quantities. The executor selects the car and instrument records and returns $629 + 611 = 1{,}240$, matching the reference answer. The same parse–check–execute sequence can be reused for other inputs satisfying these semantics.

3.3 Admission and execution

Admission links an input to a supported computation. *Grounding* associates representation fields with source content. *Coverage* checks that the required inputs have been consumed. *Invariants* enforce the executor's syntactic and semantic preconditions. The *answer contract* checks the permitted answer form and applicable uniqueness requirements. Failure of an admission check makes that capability abstain. Registry routing follows the order of these implemented checks.

The resulting separation makes correctness inspectable at two levels: whether the representation captures the required problem state, and whether execution correctly computes on that state. For a dataset D, registry coverage is

$$C(R, D) = \frac{1}{|D|} \sum_{x \in D} \mathbb{1}\{R \text{ accepts } x\}. \tag{1}$$

Coverage describes the reach of the executable route. Correctness among accepted cases describes its output quality. Model-driven orchestration handles inputs outside that route, so end-to-end accuracy depends on both executable coverage and the effectiveness of the remaining reasoning paths.

3.4 Inference and answer selection

The system first requests and retains an internal Direct answer y_D, then applies the registry to the original problem. Registry execution adds no model calls. If the registry abstains, the scaffold analyzes the entities and constraints and generates constructive and adversarial plans in sequence, passing the earlier plan as context to the later one. Workers execute each plan and verifiers inspect the resulting claims. A repair verdict permits one repair followed by reverification per path. The candidates are then composed and judged, with y_D supplied to both the judge and the final selector. Verification outcomes, candidate agreement, and answer validity determine whether a process answer replaces Direct. Appendix B specifies the execution and proposal configurations.

This design allocates supported computations to deterministic execution and uses model calls for broader interpretation and reasoning. The full system attempts Direct before registry execution. Successful execution uses one model call on the registry route and 10–14 calls on the model-driven route. The component comparison also evaluates the registry alone and a model-proposed IR interface with Direct fallback, allowing their answers to be compared with those of the complete pipeline.

4 EXPERIMENTAL SETUP

The experiments examine overall reasoning performance, performance after excluding recorded development samples, the complementary coverage of execution and orchestration, and the resources required across backends and benchmarks. Large-scale evaluations establish the performance of the complete system; the component comparison examines how its executable and model-driven routes contribute solutions.

4.1 DATASETS AND EVALUATION PROTOCOL

The primary evaluation uses all 4,520 BBEH examples across 23 task families (Kazemi et al., 2025). A 4,400-example evaluation excludes the 20-example construction sample and 100-example refinement sample by exact input matching. Additional evaluations use the complete 756-item MuSR test set (Sprague et al., 2024) and a fixed 200-item LiveBench Reasoning snapshot (White et al., 2025). Two mutually disjoint 100-item BBEH samples, both excluding BBEH-mini inputs, support component and backend-reuse analyses.

BBEH uses task-specific answer comparison and the benchmark aggregation metrics. All three conditions in the development-exclusion evaluation use the same explicit-option and numeric-answer extraction; Appendix A details its relation to full-suite scoring. MuSR and LiveBench use benchmark-specific answer normalization and comparison. Missing or failed answers count as incorrect, and accuracy uses the full evaluation population.

4.2 METHODS AND COMPARISON ROLES

Direct answers each input with the same backend used by the proposed system. BBEH Direct uses a structured-answer prompt; Direct on the additional benchmarks uses single-sample reasoning with the benchmark's answer contract. The complete system generates its internal Direct candidate with a structured-answer prompt, so this candidate and the standalone Direct baseline on the additional benchmarks differ in prompt format. **Compiled orchestration** is the complete system in Section 3: an internal Direct candidate, registry execution, and the reasoning and selection pipeline. The primary comparisons use `gpt-5.4-nano`, `gpt-5.6-luna`, and `qwen3.7-flash`. A stronger-model reference, `gpt-5.6-sol`, receives the original BBEH input without the structured system prompt.

The component comparison evaluates **executor only**, **Direct**, **model-proposed IR with Direct fallback**, and **compiled orchestration** using `gpt-5.4-nano` for model calls. The backend-reuse comparison also includes `gpt-5.4-mini`; the additional benchmark comparison includes `deepseek-v4-flash`. Supplementary `Qwen3.5-9B` experiments compare answer-only inference, nine-answer voting, ReAct, and three-sample chain-of-thought voting. These sampling and tool-use baselines have their own prompts and budgets, reported in Appendix C.

Model-proposed representations. This condition tests a model-mediated interface to the executable capabilities. The input is segmented into source units, and the model proposes a representation type, source bindings, and a query. Validation checks complete, unique source coverage, semantic bindings, and the query type before invoking the registered parser and executor on the original input. Selected validation failures permit one constrained repair. A rejected proposal falls back to Direct. This condition therefore measures how much executable coverage the proposal interface obtains, while compiled orchestration additionally uses planning, verification, composition, and selection to solve inputs beyond that interface.

4.3 METRICS AND RESOURCE ACCOUNTING

BBEH micro accuracy weights items equally, while macro accuracy averages the 23 task-family accuracies. Adjusted harmonic mean (HM) gives greater weight to low-accuracy families:

$$\mathrm{HM} = \frac{23}{\sum_{i=1}^{23}(a_i + 1)^{-1}}, \tag{2}$$

where a_i is task accuracy in percentage units and the scorer retains the one-point offset in the final score. Accuracy differences are percentage points (pp); HM differences are score points. Additional benchmarks use item accuracy. The reported values are aggregate point estimates.

Table 1: Results on the full BBEH suite with published BBEH references shown separately. Compiled orchestration improves all three same-backend comparisons. Macro and micro accuracy are percentages; Adjusted HM is the BBEH official metric. Bold marks the best result.

Backend	Method	Macro	Micro	Adjusted HM (BBEH official metric)
`gpt-5.4-nano`	Direct	12.3	12.0	3.5
	Compiled orchestration	38.5	38.5	17.1
`gpt-5.6-luna`	Direct	48.6	48.4	27.1
	Compiled orchestration	64.7	64.9	47.7
`qwen3.7-flash`	Direct	47.4	47.2	27.6
	Compiled orchestration	63.9	64.0	**47.9**
`gpt-5.6-sol`	Benchmark-input direct	67.7	67.7	37.2
Gemini-2.0-Flash (Ikenoue et al., 2025)	Adaptive prompt (23 tasks)	28.50	–	13.30
Gemini-2.0-Flash (Stein et al., 2025)	PIPS (80%/task)	27.99	–	17.10
o3-mini-high (Kazemi et al., 2025)	CoT (full)	–	54.2	44.8
Qwen3-235B-A22B (Stein et al., 2025)	PIPS (80%/task)	47.66	–	38.59

Inference resources are reported as recorded model calls, tokens, and estimated API expense. Call totals count completions retained by the runners and exclude transport retries and attempts without retained completions. Token estimates use reconstructed traces for the lightweight backends and sampled API usage for the stronger-model reference. Appendix D specifies the accounting sources, output-token treatment, and fixed rates used in the comparison.

5 Results and Analysis

5.1 Full-suite reasoning gains

Compiled orchestration improves both overall accuracy and task-family HM across all three principal backends (Table 1). Micro accuracy rises by 26.5 pp for `gpt-5.4-nano`, 16.5 pp for `gpt-5.6-luna`, and 16.8 pp for `qwen3.7-flash`. Their HM gains are 13.6, 20.6, and 20.3 points, respectively. The same execution-and-orchestration design therefore benefits backends with substantially different Direct performance.

With compiled orchestration, `gpt-5.6-luna` and `qwen3.7-flash` achieve HM scores of 47.7 and 47.9, above the stronger-model reference's 37.2. Their micro accuracies of 64.9% and 64.0% approach its 67.7%. The combination thus changes the performance profile across task families while closing much of the overall accuracy gap. Section 5.5 relates these results to inference expense.

Completion reliability also improves. For `qwen3.7-flash`, the number of error records falls from 497 under Direct to 66 under compiled orchestration, reducing the error rate from 11.0% to 1.5%. Since failed answers count as incorrect, the full-suite scores capture both answer correctness and successful completion. This reduction accompanies 759 additional correct answers across the 4,520 inputs.

5.2 Evaluation after development-sample exclusion

After excluding the 120 construction and refinement inputs, `gpt-5.6-luna` improves from 50.39% to 64.59% micro accuracy and from 27.41 to 47.58 HM (Table 2). The gains of 14.20 pp in accuracy and 20.17 HM points establish that the benefit extends beyond these two development samples. The comparison includes 625 additional correct answers across the 4,400 remaining inputs.

On the same population, compiled orchestration with `gpt-5.6-luna` achieves 47.58 HM, exceeding `gpt-5.6-sol` by 10.16 points. Under the primary task-family metric, compiled orchestration thus enables a lightweight backend to outperform the substantially more expensive stronger model even after development-sample exclusion.

Table 2: BBEH results after excluding the 120 construction and refinement inputs, leaving 4,400 examples. All rows use the same development-exclusion scoring protocol. Macro and micro accuracy are percentages; HM is a score.

Backend	Method	Macro	Micro	HM
`gpt-5.6-luna`	Direct	50.47	50.39	27.41
	Compiled orchestration	64.36	64.59	47.58
`gpt-5.6-sol`	Benchmark-input direct	67.74	67.75	37.42

Table 3: Component comparison on a separate 100-item BBEH sample. Model-driven conditions use `gpt-5.4-nano`; calls are recorded totals over the sample. Compiled orchestration retains all 27 executor successes and adds 12.

Method	Accuracy (%)	Recorded calls
Executor only	27	0
Direct	16	100
Model-proposed IR + Direct	17	130
Compiled orchestration	39	943

5.3 Complementary execution and orchestration

The component comparison makes the division of labor visible (Table 3). On the 100-item component sample, the executor registry solves 27 items without model calls, compared with 16 for Direct. The complete system solves 39. Comparing the answers item by item reveals how the executable and model-driven routes contribute to this improvement.

Figure 2 resolves the aggregate gains into paired outcomes. Direct and compiled orchestration share 13 correct answers; orchestration corrects 26 Direct errors and loses three Direct successes, giving a net gain of 23 answers. Relative to the executor, the complete system preserves all 27 successes and adds 12. Thus the gain combines a stable executable contribution with additional solutions from the remaining reasoning routes.

The model-proposed IR condition accepts six proposals, all correct, and reaches 17% overall accuracy with Direct fallback. Its call total of 130 reflects proposal generation and fallback, whereas the complete pipeline uses 943 calls to reach 39%. Admission gives the proposal interface a precise executable subset in this sample; the broader coverage comes from the complete pipeline's reasoning and selection. The paired outcomes distinguish these solution sets even when aggregate accuracy alone obscures their overlap.

Executor reuse also persists across backends. On the separate backend-reuse sample, the same capabilities solve the same 28 items with `gpt-5.4-nano`, `gpt-5.4-mini`, and `gpt-5.6-luna`. The complete system raises their accuracies from 18% to 42%, 16% to 39%, and 51% to 65%, respectively. The executable solution set remains unchanged across these backends, while model-driven reasoning determines the additional coverage.

5.4 Performance across reasoning benchmarks

The additional benchmark evaluations test performance across reasoning tasks beyond BBEH (Table 4). On the 200-item LiveBench Reasoning snapshot, `gpt-5.6-luna` improves from 165 to 180 correct answers, a gain of 7.5 pp. On the complete MuSR test set, it improves from 480 to 498 correct answers, while `deepseek-v4-flash` improves from 521 to 586. These correspond to gains of 2.4 and 8.6 pp, respectively. The MuSR results use the selection and error-handling refinements described in Appendix A.

For `gpt-5.6-luna`, compiled orchestration improves both LiveBench and MuSR; on MuSR, the gain extends to `deepseek-v4-flash`. On LiveBench, both DeepSeek conditions achieve 96.5%, corresponding to 193 correct answers out of 200 in the reported evaluation.

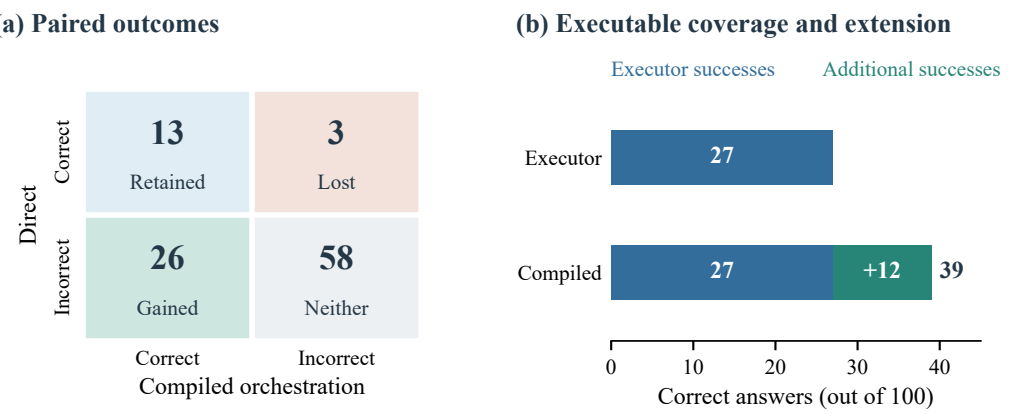

Figure 2: Complementary solution coverage on the 100-item component sample with `gpt-5.4-nano`. (a) Paired correctness separates 26 gains from three regressions relative to Direct. (b) All 27 executor successes remain correct in the complete system, which solves 12 additional items. Table 3 reports the corresponding inference calls.

Table 4: Accuracy (%) on the fixed LiveBench Reasoning snapshot and the complete MuSR test set. Gains are compiled orchestration minus Direct in percentage points.

Dataset	Backend	Direct	Compiled	Gain
LiveBench (200)	`gpt-5.6-luna`	82.5	90.0	+7.5
	`deepseek-v4-flash`	96.5	96.5	0
MuSR (756)	`gpt-5.6-luna`	63.5	65.9	+2.4
	`deepseek-v4-flash`	68.9	77.5	+8.6

5.5 Inference cost

Table 5 compares estimated inference expense over the full BBEH suite. Compiled orchestration has estimated costs of $64.03 with `gpt-5.6-luna` and $4.57 with `qwen3.7-flash`, compared with $195.42 for direct inference with `gpt-5.6-sol`. These correspond to reductions of 67.2% and 97.7%, respectively. Paired with Table 1, the comparison shows higher task-family HM at 67.2%–97.7% lower estimated expense while maintaining 64.0%–64.9% micro accuracy.

At the stated rates, the lower token prices of the lightweight backends offset the additional inference used for planning, verification, and answer selection. Lightweight-backend token totals are reconstructed from prompt and response traces; the stronger-model estimate weights sampled API usage by task size. Appendix D gives the rates and calculation. Together with the component call counts, these estimates describe the resources used to extend solution coverage.

Sampling, tools, and task-family balance. Figure 3 compares four inference strategies for `Qwen3.5-9B` on full-suite BBEH. Nine-answer voting increases average token use from 1,316 to 11,362 per item, while micro accuracy rises from 13.78% to 15.07%. Three-sample CoT voting reaches 30.02% using 17,617 tokens per item. ReAct uses 45,312 tokens and reaches 23.72% micro accuracy, with the highest HM of 11.6 among these strategies. CoT voting and answer-only Direct both have HM near 3.8 despite their different micro accuracies.

These results show why both metrics matter: a strategy can improve overall accuracy substantially while low-accuracy task families continue to constrain HM. Inference resources and task-family coverage therefore describe different aspects of performance. This distinction also clarifies the compiled system's full-suite results, where all three principal backends improve both micro accuracy and HM.

Table 5: Estimated API inference expense on full-suite BBEH. Reductions are relative to `gpt-5.6-sol` direct inference. Token totals and pricing assumptions appear in Appendix D.

Backend	Method	Cost ($)	Reduction (%)
`gpt-5.6-sol`	Benchmark-input direct	195.42	—
`gpt-5.6-luna`	Compiled orchestration	64.03	67.2
`qwen3.7-flash`	Compiled orchestration	4.57	97.7

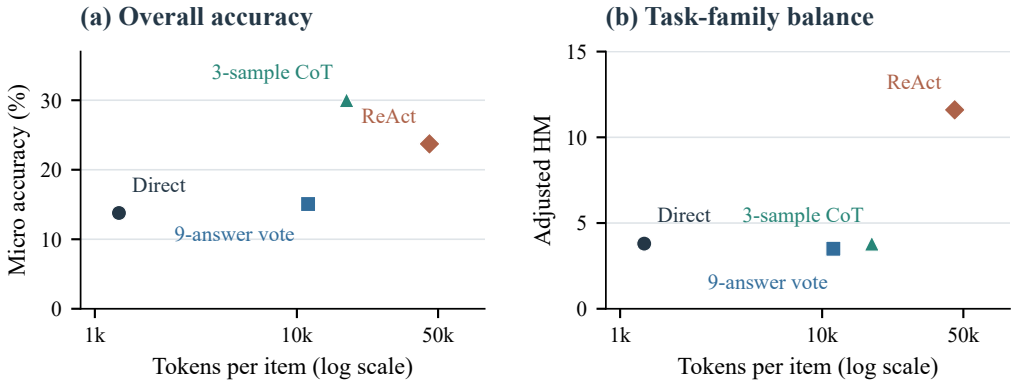

Figure 3: Inference resources and performance for four `Qwen3.5-9B` strategies on all 4,520 BBEH items. Tokens include input and output and are averaged per item; the horizontal axes are logarithmic. Three-sample CoT voting gives the highest micro accuracy, while ReAct gives the highest adjusted HM. Methods use their respective prompts and budgets; Appendix C provides exact values and configurations.

6 Limitations

External execution guarantees exact computation for problem states admitted by the supported representations and checks; end-to-end correctness additionally depends on faithful grounding. The ownership capability, for example, checks recognized first-person clauses and treats parenthetical text as explanatory history.

The component comparison measures the combined effect of inference structure and call budget; isolating individual pipeline stages requires matched-budget comparisons. MuSR evaluates a configuration refined through task-specific failure analysis. Resource estimates cover inference at the stated rates; development expense, latency, and transport-retry charges are outside this accounting.

7 Conclusion

Agentic reasoning compilation separates supported logical and numerical computations from language-model generation and combines them with a trace-guided orchestration pipeline. The executor registry supplies reusable computations, while planning, verification, and selection extend reasoning coverage. Full-suite BBEH gains across three backends persist for `gpt-5.6-luna` after excluding the construction and refinement samples. Component outcomes explain the complementary solution coverage; LiveBench extends the evidence for `gpt-5.6-luna`, while MuSR demonstrates gains for both `gpt-5.6-luna` and `deepseek-v4-flash`. Higher task-family harmonic scores at lower estimated API expense show how executable structure and orchestration can make lightweight backends more effective reasoning systems.

AI USE STATEMENT

Generative AI tools supported capability construction, implementation and code editing, experimental analysis, literature review, figure preparation, and manuscript editing. Development used model-generated reasoning traces and execution feedback. The authors take responsibility for the final method, claims, code, and manuscript.

ETHICS STATEMENT

The study uses existing reasoning benchmarks. Responsible use requires checking that each input falls within a capability's supported semantics and validating the resulting answers for the intended application.

REPRODUCIBILITY STATEMENT

The method and appendices describe the inference routes, data roles, scoring definitions, prompt formats, and cost assumptions. Component summaries include paired outcomes and call counts. The model identifiers correspond to the evaluation configurations; private infrastructure identifiers are omitted for anonymous review.

REFERENCES

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.17126.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2211.12588.

Joseph Cotnareanu, Didier Chételat, Yingxue Zhang, and Mark Coates. A balanced neuro-symbolic approach for commonsense abductive logic. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Yohei Ikenoue, Hitomi Tashiro, and Shigeru Kuroyanagi. Automatic prompt generation via adaptive selection of prompting techniques. *arXiv preprint arXiv:2510.18162*, 2025. URL https://arxiv.org/abs/2510.18162.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-Bench Extra Hard. *arXiv preprint arXiv:2502.19187*, 2025. URL https://arxiv.org/abs/2502.19187.

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. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023. URL https://aclanthology.org/2023.ijcnlp-main.20/.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.16049.

Adam Stein, Neelay Velingker, Mayur Naik, and Eric Wong. Once upon an input: Reasoning via per-instance program synthesis. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c828f33af3c3f669690c2e28ae7af5e2-Abstract-Conference.html.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.19314.

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*, 2023. URL https://arxiv.org/abs/2210.03629.

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. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Tong Zheng, Lichang Chen, Sophia Han, R. Thomas McCoy, and Heng Huang. Learning to reason via mixture-of-thought for logical reasoning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/af80ce1011eb35fe1023c320158c2ad9-Abstract-Conference.html.

A Evaluation protocol and capability families

Development and evaluation data. We use two disjoint BBEH-mini samples: 20 examples for pipeline construction and 100 for capability refinement. The 100-item backend-reuse sample excludes BBEH-mini. The 100-item component sample additionally excludes the backend-reuse sample. Both evaluation samples are disjoint from the two development samples and from each other.

Development-sample exclusion. Table 2 evaluates the 4,400 examples remaining after exact input matching removes 120 full-suite rows from the construction and refinement samples. Whitespace-normalized matching identifies one additional row; the table uses exact matching. All three conditions use the same explicit-option and numeric-answer extraction.

Scoring. Full-suite results use rescored predictions for `gpt-5.6-luna` orchestration and `gpt-5.6-sol` benchmark-input Direct. The development-exclusion evaluation also updates Direct answer extraction, so differences between Tables 1 and 2 reflect both population and extraction changes. Macro, micro, and HM follow Section 4, with failed answers counted as incorrect.

Component outcomes. Each condition has one result per item on the same 100 inputs. Pairing correctness by item gives the outcomes in Figure 2. Calls are summed over each condition: 0 for executor only, 100 for Direct, 130 for model-proposed IR with Direct fallback, and 943 for compiled orchestration. All six accepted IR proposals produce correct answers.

Capability families. The registry contains 22 entries for expression evaluation, state replay, temporal constraints, logical inference, tabular and ownership aggregation, language transformations, and spatial or causal operations. Each entry defines its parser preconditions and representation checks. Routing selects the first candidate produced by the ordered registry and returns it if the final answer contract is satisfied.

Additional benchmarks. LiveBench results use the final evaluation outputs, including selected-error retries for `deepseek-v4-flash`. MuSR uses the refined configuration with task-specific answer contracts, object-answer selection, and error-handling updates. The complete 756-item evaluation includes all three MuSR task categories.

B Inference configuration

The complete pipeline attempts to obtain its internal Direct candidate before registry execution. On registry abstention, two planning paths execute and verify claims, with at most one repair and reverification per path, followed by composition, judgment, and answer selection. The final policy determines whether a process candidate replaces Direct. The successful control flow uses one model call on an accepted registry route and 10–14 logical calls on the two-path reasoning route. Executor-only evaluation omits all model calls.

The model-proposed IR condition in Section 4 requests Direct after a rejected proposal; compiled orchestration obtains its Direct candidate before routing. Both conditions use the registered parser and executor on the original input. Call totals count retained completions, with failed attempts and transport retries excluded.

C Sampling and tool-use baselines

The `Qwen3.5-9B` comparison evaluates four inference strategies on the full BBEH suite (Table 6). Answer-only Direct uses one response. Nine-answer voting samples nine answer-only responses and selects by normalized vote. ReAct interleaves reasoning and tool use. Three-sample chain-of-thought voting requests explicit reasoning and combines three final answers by plurality, breaking ties by first occurrence. This last condition uses independent model calls without the capability registry or the complete orchestration pipeline.

Table 6: `Qwen3.5-9B` inference strategies on full-suite BBEH. Token counts are averages per item; output includes reasoning and final-answer tokens. Macro and micro accuracy are percentages; HM is a score.

Method	Input	Output	Total	Macro	Micro	HM
Answer-only Direct	1,262	54	1,316	13.9	13.78	3.8
Nine-answer voting	11,127	236	11,362	15.2	15.07	3.5
ReAct	29,419	15,893	45,312	24.0	23.72	11.6
Three-sample CoT voting	3,983	13,634	17,617	30.4	30.02	3.8

Three-sample reasoning uses 13.39 times the tokens of answer-only Direct and 1.55 times those of nine-answer voting. It gives the highest micro accuracy in this comparison, while ReAct gives the highest HM.

The answer-only and three-sample conditions differ in prompt and response format as well as sample count. The three-sample runner uses a reasoning prompt and final-answer marker, with a default 8,192-token output ceiling per call. The answer-only and ReAct runner uses a default ceiling of 32,768 tokens. Its ReAct protocol permits a second run when the first uses fewer calls than the target average, so realized budgets differ across methods. Table 6 reports the observed token use. The separately evaluated original-input Direct condition for `gpt-5.4-nano` obtains 4.2% macro accuracy, 4.3% micro accuracy, and 1.4 HM, compared with the structured Direct values in Table 1.

D API COST ACCOUNTING

Table 7 gives the estimated token totals used in the full-suite cost comparison. Counts for `gpt-5.6-luna` and `qwen3.7-flash` are reconstructed from saved prompt and response traces. The `gpt-5.6-sol` estimate uses a later stratified sample of 20 examples per BBEH task. Of the 460 sampled requests, 448 return valid usage records; their task-wise mean usage is weighted by full-suite task size. Sampled API usage includes reasoning and final-answer output, while reconstructed counts use the text available in the traces.

The fixed USD rates per million input/output tokens used in the recorded comparison are \$5.00/\$30.00 for `gpt-5.6-sol`, \$0.20/\$1.20 for `gpt-5.6-luna`, and \$0.030/\$0.130 for `qwen3.7-flash` at the stated $\leq$ 32K tier.

Table 7: Estimated input and output token totals, in millions (rounded), for the full-suite cost comparison in Table 5. Lightweight-backend counts are reconstructed from traces; the stronger-model reference is extrapolated from sampled API usage.

Backend	Input	Output
`gpt-5.6-sol`	5.03	5.68
`gpt-5.6-luna`	162.01	26.36
`qwen3.7-flash`	92.54	13.82

For token counts $T_{\text{in}}, T_{\text{out}}$ and rates $p_{\text{in}}, p_{\text{out}}$ per million tokens, estimated expense is

$$\widehat{C} = (T_{\text{in}} p_{\text{in}} + T_{\text{out}} p_{\text{out}})/10^6. \tag{3}$$