SHUQI: SUPPORT-AWARE EXECUTABLE REASONING FOR REPRESENTATION–EXECUTION DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Final-answer accuracy reveals whether a reasoning system is wrong, but not whether the error arises from constructing a usable representation or from executing semantics that are already explicit. We study this distinction with a controlled Raw-NL→Same-IR→deterministic diagnostic interface and semantic-stability audits that separately expose representation-associated recovery, residual execution-side error, and instability under fixed semantic content. These diagnostics reveal substantial effects: residual execution gaps reach 23.0 percentage points, while lossless serializations of identical canonical representations disagree on up to 30.8% of disjoint-source examples. Based on these findings, we propose SHUQI, a task-grounded support-aware executable reasoning system that compiles supported inputs into executable representations, performs deterministic task-defined execution with replay-checked evidence, and releases the symbolic result only when a task-specific support contract succeeds; otherwise, it falls back to a frozen generative reasoner. On ProofWriter, SHUQI preserves all 41 symbolic fixes while introducing no regressions, reaching 98.0% accuracy, and achieving 100% on both ProntoQA and LogicalDeduction. These results show that reliable reasoning requires controlling not only what semantic representation is constructed, but also how that representation is executed and when an executable result should be released.

1 INTRODUCTION

A wrong final answer does not reveal where reasoning failed. A system may fail while constructing a task-relevant representation, or while executing a problem whose relevant semantics are already explicit. These are successive operational stages, $x_{\mathrm{NL}} \to z_{\mathrm{IR}} \to y$, yet generative approaches such as chain-of-thought typically use one model process for both (Wei et al., 2022). Final-answer accuracy therefore collapses representation and execution into a single outcome.

We find that separating these stages matters in practice. Some errors disappear when the same generative executor is supplied with canonical task semantics, indicating *representation-associated recovery*; others remain even after those semantics are explicit. In our controlled comparisons, the residual gap between Same-IR generative execution and deterministic execution reaches **23.0 percentage points**. Moreover, fixing canonical semantics does not guarantee stable generative execution: predictions can change under semantics-preserving transformations, fail to track validated semantic changes, or disagree across lossless serializations of identical canonical IR. In a disjoint-source replication, such identical-IR disagreement reaches **30.8%**. Figure 1 summarizes these three observed patterns, which we treat as operational diagnostics rather than a causal decomposition of model internals.

To expose this boundary, we introduce a controlled *representation–execution interface*. Raw-NL and Same-IR hold the generative executor fixed while changing whether canonical semantic content is supplied; Same-IR and deterministic execution then hold that content fixed while changing how it is executed. This lets us separately test sensitivity to representation and reliability of execution once canonical semantics are available. Program- and solver-assisted methods already delegate computation to external mechanisms (Lyu et al., 2023; Chen et al., 2022); our contribution is not symbolic delegation itself, but making the representation–execution boundary experimentally

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.

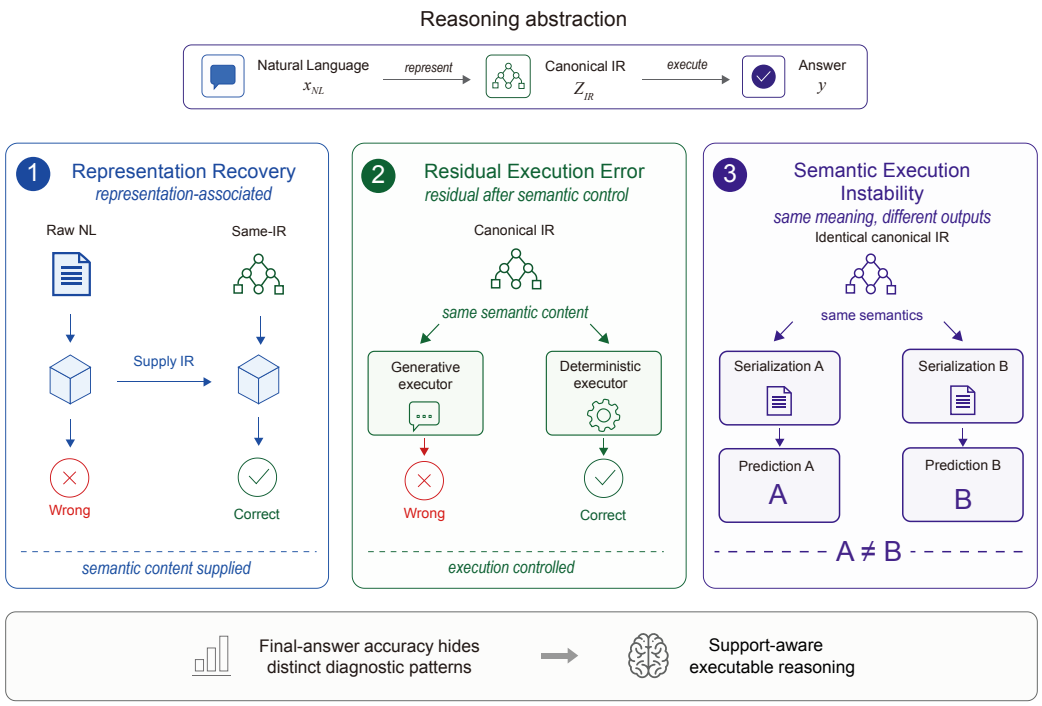


Figure 1: **Final-answer accuracy hides distinct reasoning failure patterns.** Controlled comparisons along $x_{\mathrm{NL}} \to z_{\mathrm{IR}} \to y$ expose representation-associated recovery, residual execution-side error, and semantic execution instability.

controllable and using executable support to determine when a deterministic result should govern the final prediction.

Executable coverage, however, can be incomplete, so forcing every instance through a symbolic path can introduce new errors. We therefore propose SHUQI (**S**tructured **H**ybrid **U**nderstanding and **Q**uery-grounded **I**nference), a task-grounded support-aware executable reasoning system following a compile–execute–replay–release workflow. A deterministic compiler constructs executable IR when supported, a task-specific executor produces a decision and replay evidence, and a release contract determines whether the deterministic result is admissible. Compilation or release failure instead invokes a frozen generative fallback on the original input.

Across ProofWriter, ProntoQA, and LogicalDeduction, the controlled diagnostics expose both representation-associated recovery and residual execution-side error, while semantic-preservation, semantic-change, and identical-IR audits reveal post-IR instability. Under incomplete ProofWriter coverage, support-aware routing preserves all **41** symbolic fixes while introducing **0 regressions**, reaching **98.0%** accuracy on ProofWriter and **100%** on both ProntoQA and LogicalDeduction. Detailed effect sizes, confidence intervals, and representative cases appear in Section 4.

Our contributions are:

- **Controlled representation–execution diagnosis.** We introduce a matched-content Raw-NL→Same-IR→deterministic framework that distinguishes representation-associated recovery from residual execution-side error without claiming a causal decomposition of model internals.
- **Semantic execution stability audits.** We test invariance under validated semantic equivalence, responsiveness to independently validated semantic change, and agreement across lossless serializations of identical canonical IR.
- **SHUQI: support-aware executable reasoning.** We release deterministic execution only when a task-specific support and replay contract succeeds and otherwise fall back to a frozen

generative reasoner, retaining symbolic gains while avoiding regressions from unsupported handoff.

2 Related Work

Generative, tool-assisted, and formal reasoning. Chain-of-thought and least-to-most prompting elicit intermediate natural-language reasoning (Wei et al., 2022; Zhou et al., 2022), while later methods add symbolic intermediates, premise selection, or solver-like traces (Saparov & He, 2022; Sun et al., 2024; Xu et al., 2024). Program- and solver-assisted systems already translate language into programs or formal representations for external execution (Gao et al., 2023; Chen et al., 2022; Paranjape et al., 2023; Yao et al., 2022; Lyu et al., 2023; Ye et al., 2023; Pan et al., 2023; Olausson et al., 2023; McGinness & Baumgartner, 2024). Our contribution is therefore not symbolic delegation, but controlled intervention at the representation–execution boundary: Raw-NL and Same-IR hold the generative executor fixed, while Same-IR and deterministic execution use matched canonical semantic content. Recent systems such as Symbolic-Aided CoT, AAI, VERUS-LM, and MenTaL provide complementary mechanisms (Nguyen et al., 2025; 2026; Callewaert et al., 2025; Li et al., 2026). MenTaL is closest to our representation-side question, but primarily targets $x_{\mathrm{NL}} \to z$ stability; our audits instead test post-representation $z \to y$ behavior under semantic-preservation, semantic-change, and identical-IR controls.

Selective prediction and routing. Selective classification and learning-to-defer allow abstention or delegation (Geifman & El-Yaniv, 2017; Mozannar & Sontag, 2020), while LLM cascades and routers trade off quality and cost (Chen et al., 2023; Ong et al., 2025). SHUQI instead releases symbolic results using a deterministic contract over compilation support, terminal validity, and replay evidence rather than learned confidence, preference, or cost.

Faithfulness and execution reliability. Generated rationales need not reflect the true basis of model predictions (Turpin et al., 2023; Lanham et al., 2023), while theorem-prover and proof-carrying approaches motivate machine-checkable evidence (Qi et al., 2025; Necula, 1997). SHUQI uses replay only to validate execution relative to compiled IR, not the upstream $x_{\mathrm{NL}} \to z$ mapping; our stability tests similarly evaluate behavioral invariance, responsiveness to semantic change, and agreement across lossless identical-IR serializations.

3 Method

3.1 Problem Formulation and the Representation–Execution Interface

For task t, let Σ denote the token alphabet, $\mathcal{X}_t \subseteq \Sigma^*$ the space of natural-language inputs, and $\mathcal{Y}_t$ the task-specific answer space. For an instance $x \in \mathcal{X}_t$ and a frozen model snapshot m, we write

$$G_{m,t} : \Sigma^* \to \mathcal{Y}_t \cup \{\texttt{invalid}\}$$

for the complete frozen generative protocol, including prompting, decoding, and answer parsing. Here `invalid` denotes failure to produce a valid task answer, so direct generation predicts $y^{(m)}_{\mathrm{Raw}} = G_{m,t}(x)$.

SHUQI introduces an explicit boundary between constructing task-relevant semantics and executing them. Let $\mathcal{Z}_t$ denote the task-specific executable-representation space. A deterministic task-grounded compiler

$$C_t : \mathcal{X}_t \to \mathcal{Z}_t \cup \{\bot_C\}$$

either constructs an executable representation $z = C_t(x)$ or returns the distinguished symbol $\bot_C$ when the input lies outside its declared compilation fragment. To avoid confounding downstream execution analysis with errors introduced by a learned semantic parser, the deployed compilers are exact-template parsers for bounded controlled languages rather than learned formalizers.

For successfully compiled $z \in \mathcal{Z}_t$, let $\mathcal{S}_t$ be the terminal semantic-state space and $\mathcal{R}_t$ the replay-evidence space. The deterministic executor $E_t : \mathcal{Z}_t \to \mathcal{S}_t \times \mathcal{R}_t$ returns a terminal semantic state s and replayable evidence r. The task decoder $D_t : \mathcal{S}_t \to \mathcal{Y}_t \cup \{\texttt{invalid}\}$ maps the terminal state

to an answer, and $V_t : \mathcal{Z}_t \times \mathcal{S}_t \times \mathcal{R}_t \to \{0,1\}$ checks whether the recorded computation replays relative to z:

$$(s, r) = E_t(z), \qquad y_{\text{sym}} = D_t(s), \qquad V_t(z, s, r) \in \{0, 1\}. \tag{1}$$

Thus s contains the task-specific semantic information required to determine an answer, while r contains the evidence needed to replay the computation. Compilation and replay certify different boundaries: compilation determines whether a supported executable object has been constructed, whereas replay checks computation relative to that object. In particular, replay validates $z \to (s, r) \to y_{\text{sym}}$, not the semantic faithfulness of the upstream mapping $x \to z$.

Diagnostic interface. Raw-NL, Same-IR, and deterministic execution are controlled experimental conditions rather than three deployed SHUQI branches. Let $\sigma_t : \mathcal{Z}_t \to \Sigma^*$ denote the frozen serializer, which is lossless with respect to the permitted canonical semantic fields. For a fixed model m and successfully compiled representation $z = C_t(x)$,

$$y^{(m)}_{\text{Raw}} = G_{m,t}(x), \qquad y^{(m)}_{\text{IR}} = G_{m,t}(\sigma_t(z)), \qquad (s, r) = E_t(z), \;\; y_{\text{Det}} = D_t(s). \tag{2}$$

Here $y^{(m)}_{\text{Raw}}$ is the prediction from the original natural-language input, $y^{(m)}_{\text{IR}}$ is the prediction from its lossless Same-IR serialization, and y_{Det} is the corresponding deterministic-execution answer. Raw-NL→Same-IR measures representation-associated changes under a fixed generative executor, while Same-IR→deterministic measures the remaining execution-side gap under matched canonical content. Neither comparison is interpreted as a causal decomposition of internal model computation.

3.2 SHUQI: SUPPORT-AWARE EXECUTION AND FALLBACK

SHUQI turns this interface into a task-specific release mechanism (Figure 2). Let

$$B_t : \mathcal{X}_t \to \mathcal{Y}_t \cup \{\texttt{invalid}\}$$

denote the frozen generative fallback, and let

$$H_t : \mathcal{X}_t \times \mathcal{Z}_t \times \mathcal{S}_t \times \mathcal{R}_t \to \{0, 1\}$$

denote the task-specific release predicate, defined only after successful compilation and execution.

Compilation failure invokes the fallback directly. If $C_t(x) = z \in \mathcal{Z}_t$, we first compute $(s, r) = E_t(z)$ and then apply the release predicate. The final SHUQI prediction is therefore

$$\hat{y} = \begin{cases} B_t(x), & C_t(x) = \bot_C, \\ D_t(s), & C_t(x) = z \in \mathcal{Z}_t,\ (s, r) = E_t(z),\ H_t(x, z, s, r) = 1, \\ B_t(x), & C_t(x) = z \in \mathcal{Z}_t,\ (s, r) = E_t(z),\ H_t(x, z, s, r) = 0. \end{cases} \tag{3}$$

Hence H_t is never evaluated when compilation fails, and no nonexistent z, s, or r is introduced in that case. The release predicate uses only compilation and execution artifacts—never gold labels, branch correctness, model confidence, or evaluation outcomes. SHUQI therefore does not choose whichever branch appears more likely to be correct; it releases deterministic execution only when its declared contract is satisfied.

ProofWriter release contract. ProofWriter provides the partial-coverage deployment. Let

$$\text{Complete}_{\text{PW}} : \mathcal{X}_{\text{PW}} \times \mathcal{Z}_{\text{PW}} \to \{0, 1\}, \qquad \text{Integrity}_{\text{PW}} : \mathcal{Z}_{\text{PW}} \to \{0, 1\}$$

denote the compilation-coverage and compiled-IR integrity checks, respectively. Let

$$V_{\text{proof}}, V_{\text{sat}} : \mathcal{Z}_{\text{PW}} \times \mathcal{S}_{\text{PW}} \times \mathcal{R}_{\text{PW}} \to \{0, 1\}$$

denote replay predicates for proof-bearing and saturation-based decisions. The ProofWriter decoder $D_{\text{PW}} : \mathcal{S}_{\text{PW}} \to \{A, B, C, \texttt{invalid}\}$ returns A for query entailment, B for opposite entailment, C when neither side is derivable, and `invalid` for the contradictory BOTH condition.

For a successfully compiled instance, the deployed release contract is

$$\begin{aligned} H_{\text{PW}}(x, z, s, r) = \text{Complete}_{\text{PW}}(x, z) \wedge \text{Integrity}_{\text{PW}}(z) \\ \wedge \Big[\big(D_{\text{PW}}(s) \in \{A, B\} \wedge V_{\text{proof}}(z, s, r) \big) \\ \vee \big(D_{\text{PW}}(s) = C \wedge V_{\text{sat}}(z, s, r) \big) \Big]. \end{aligned} \tag{4}$$

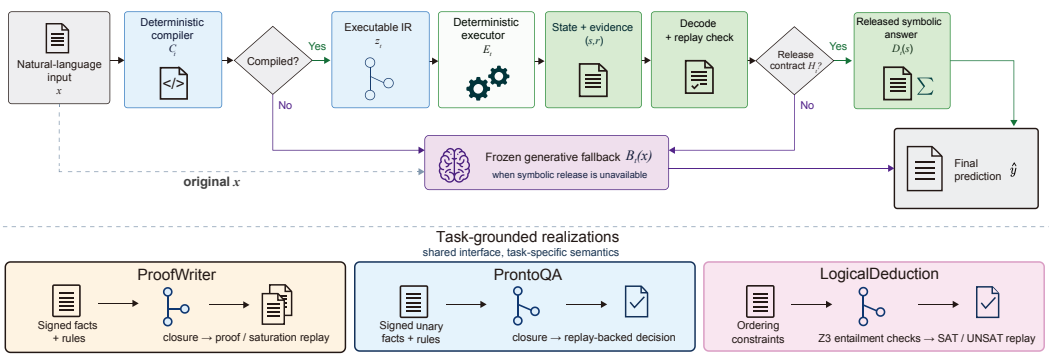


Figure 2: **SHUQI overview.** A task-grounded deterministic compiler constructs executable IR when possible. Deterministic execution produces a terminal semantic state and replay evidence, and the symbolic answer is released only when the task-specific release contract succeeds. Compilation or release failure invokes the frozen generative fallback on the original input. Bottom: the shared interface is instantiated with task-specific representations and execution semantics.

Here $\mathrm{Complete}_{\mathrm{PW}}(x, z) = 1$ requires all source statements needed by the supported fragment to be recognized and lowered into z, while $\mathrm{Integrity}_{\mathrm{PW}}(z) = 1$ requires the frozen schema, grounding, and query fields to be internally consistent. $V_{\mathrm{proof}} = 1$ means that the proof supporting an A/B decision replays successfully, and $V_{\mathrm{sat}} = 1$ means that the saturated closure supporting a C decision replays successfully. The contradictory BOTH condition maps to `invalid` and is never released. Successful compilation is therefore necessary but not sufficient for accepted symbolic release.

For ProofWriter,

$$B_{\mathrm{PW}}(x) \equiv B_0(x),$$

where B_0 denotes the unchanged frozen multi-stage Qwen2.5-14B-Instruct generative baseline, reused as the fallback. It receives only the original natural-language input x; neither compiled IR nor execution evidence is injected into the fallback. ProntoQA and LogicalDeduction instantiate the same release pattern but satisfy their contracts on all primary-set examples, so fallback is not exercised there. Full compiler conditions and replay procedures appear in Appendices A and I.

3.3 TASK-GROUNDED REALIZATIONS

SHUQI shares a compile–execute–replay–release interface across task-specific IRs (Figure 2).

Signed-rule tasks. ProofWriter and ProntoQA share the same typed signed-rule executor; coverage differences arise in compilation. Their executable representation has the form

$$z = (F, R, q, \mathcal{U}_z),$$

where F is the set of positive or explicit-negative ground facts, R is the set of signed implication rules, q is the signed ground query, and $\mathcal{U}_z$ is the finite set of constants visible in the compiled instance.

Each rule $\rho \in R$ is a finite Horn-style implication

$$\bigwedge_{\ell \in b(\rho)} \ell \;\rightarrow\; h(\rho),$$

where $b(\rho)$ is its finite set of signed body literals and $h(\rho)$ is its signed head literal. Let $\mathrm{Var}(\rho)$ denote the variables occurring in ρ. A grounding substitution for ρ is a type-respecting map

$$\theta : \mathrm{Var}(\rho) \rightarrow \mathcal{U}_z,$$

and $\Theta_z(\rho)$ denotes the finite set of all substitutions permitted by the visible constants and frozen type constraints.

Let $\mathcal{C}_k$ denote the set of ground signed literals derived after k closure iterations. Execution begins from the compiled facts and repeatedly applies grounded rules:

$$\mathcal{C}_0 = F, \qquad \mathcal{C}_{k+1} = \mathcal{C}_k \cup \{h(\rho)\theta \mid \rho \in R,\ \theta \in \Theta_z(\rho),\ b(\rho)\theta \subseteq \mathcal{C}_k\}\,. \tag{5}$$

Because the domain and rule set are finite and no derived literal is removed, this monotone process terminates at a fixed point $\mathcal{C}^\star$ satisfying $\mathcal{C}_{k+1} = \mathcal{C}_k = \mathcal{C}^\star$. The signed-rule terminal semantic state is

$$s_{\mathrm{SR}} = (\mathcal{C}^\star, q),$$

Let $\bar{q}$ denote the explicit signed opposite of q. From $s_{\mathrm{SR}} = (\mathcal{C}^\star, q)$, define four query outcomes: A when $q \in \mathcal{C}^\star$ and $\bar{q} \notin \mathcal{C}^\star$; B when $\bar{q} \in \mathcal{C}^\star$ and $q \notin \mathcal{C}^\star$; C when neither belongs to $\mathcal{C}^\star$; and BOTH when both are present. These are *decoded query outcomes*, not execution states.

For ProofWriter, D_{PW} maps outcomes A, B, and C to the corresponding benchmark answers and maps BOTH to `invalid`. For ProntoQA, D_{PQA} maps $A \mapsto \mathrm{true}$ and $B, C \mapsto \mathrm{false}$, with BOTH again mapped to `invalid`. ProntoQA covers all primary examples; 23 ProofWriter examples use fallback.

Running example. For frozen `ProntoQA_60`, compilation maps the rule chain $\mathrm{Wumpus} \to \cdots \to \mathrm{Dumpus} \to \neg\mathrm{Mean}$ and query $+\mathrm{Mean}(\mathrm{sam})$ into z. Closure derives $-\mathrm{Mean}(\mathrm{sam})$; replay verifies the derivation, and the release contract returns *false*. Unsupported compilation or replay would instead invoke $B_t(x)$ (Appendix C.2).

Finite-order task. LogicalDeduction uses a different executable representation,

$$z = (\Phi, O), \qquad O = (o_1, \ldots, o_K),$$

where Φ is the conjunction of compiled finite-domain ordering constraints and O is the ordered list of K candidate answer propositions. Let $\mathcal{I} = \{1, \ldots, K\}$ be the option-index set; we identify $\mathcal{Y}_{\mathrm{LD}}$ with these indices (or equivalently, with their fixed benchmark option labels).

Each entity is represented by an integer position variable constrained by ALLDIFFERENT together with the ordering relations in Φ. For every candidate o_i, $i \in \mathcal{I}$, Z3 evaluates the countermodel query $\Phi \wedge \neg o_i$. Let

$$\tau_i \in \{\texttt{SAT}, \texttt{UNSAT}, \texttt{UNKNOWN}\}$$

denote its solver status and let $s_{\mathrm{LD}} = (\tau_i)_{i \in \mathcal{I}}$ be the terminal execution state. Then

$$\Phi \models o_i \quad \Longleftrightarrow \quad \tau_i = \texttt{UNSAT} \quad \Longleftrightarrow \quad \Phi \wedge \neg o_i \text{ is UNSAT.} \tag{6}$$

The LogicalDeduction decoder closes the loop from solver states to the task answer:

$$D_{\mathrm{LD}}(s_{\mathrm{LD}}) = \begin{cases} i^\star, & \exists!\, i^\star \in \mathcal{I} : \tau_{i^\star} = \texttt{UNSAT}, \\ & \text{and } \tau_j \in \{\texttt{SAT}, \texttt{UNSAT}\} \text{ for all } j \in \mathcal{I}, \\ \texttt{invalid}, & \text{otherwise.} \end{cases} \tag{7}$$

Thus exactly one option must be certified MUST-BE-TRUE and all candidate queries must resolve normally. The corresponding SAT/UNSAT evidence is stored in r and must replay successfully before the release predicate accepts the answer. Zero or multiple MUST-BE-TRUE options, any `UNKNOWN` or timeout status, replay failure, and invalid encodings are never coerced into predictions.

Full grammars and grounding rules are given in Appendix A, while certificate schemas and replay procedures are detailed in Appendix I.

4 EXPERIMENTS

Evaluation Benchmarks and Models. We evaluate on three controlled deduction benchmarks: **ProofWriter** (Tafjord et al., 2021), **ProntoQA** (Saparov & He, 2022), and **LogicalDeduction** (Srivastava et al., 2022), covering signed-rule deduction, unary taxonomic reasoning, and finite-domain ordering constraints, respectively. Our primary matched evaluation sets contain 200, 500, and 300 source examples. The controlled representation–execution and stability experiments use **Qwen2.5-14B** (Team, 2024) and **DeepSeek-R1-14B** (Guo et al., 2025); the end-to-end comparison evaluates recent baselines on a common **Qwen3-14B** BF16 checkpoint (Yang et al., 2025). Table 1 summarizes task structure and executable coverage.

Dataset	N	Output	Structure / example	Reasoning regime	Symbolic support
ProofWriter	200	3-way truth	8.58 facts + 5.95 rules	Signed rules; unary + binary	177/200 cert.
ProntoQA	500	Binary truth	1 fact + 16.32 rules	Unary chains; target depth 5	500/500
LogicalDeduction	300	3/5/7-way MC	5.53 objects + 4.55 constraints	Finite ordering; 4 surface families	300/300

Table 1: Structural characteristics of the frozen evaluation universes. Statistics use only the frozen 200/500/300 examples. ProofWriter support denotes final certified symbolic routes; the remaining 23 examples use the frozen fallback.

Method	ProofWriter ($N = 200$)	ProntoQA ($N = 500$)	LogicalDeduction ($N = 300$)
SymbCoT[2024]	88.50	97.60	13.00
Symbolic-Aided CoT[2025]	82.00	99.20	85.67
AAI[2026]	81.00	99.80	84.67
VERUS-LM[2025]	77.00	31.40	82.67
MenTaL[2026]	64.00	38.60	73.67
SHUQI (ours)	**98.00**	**100.00**	**100.00**

Table 2: Local frozen-pipeline accuracy (%) on identical evaluation universes. Superscripts denote publication year. The five reproduced methods use the same Qwen3-14B BF16 backbone but retain method-specific inference protocols. SHUQI is task engineered and not inference-resource matched; the table provides end-to-end context rather than a matched-resource ranking.

Controls and Statistics. Gold labels and branch correctness are unavailable to compilation, execution, replay, routing, and transformation selection; confirmatory configurations are fixed before scoring. Source-clustered bootstrap is used when multiple conditions share a source, and invalid generations remain in accuracy denominators (Appendix K).

4.1 RQ1: Does SHUQI Improve End-to-End Reasoning?

We first ask whether SHUQI delivers strong end-to-end performance before turning to the controlled experiments that explain where its gains arise. On the same evaluation sets, we reproduce five recent reasoning systems: SymbCoT (Xu et al., 2024), Symbolic-Aided CoT (Nguyen et al., 2025), AAI (Nguyen et al., 2026), VERUS-LM (Callewaert et al., 2025), and MenTaL (Li et al., 2026). The reproduced baselines use the same Qwen3-14B BF16 checkpoint while retaining their method-specific prompting, decoding, intervention, and solver or tool protocols. Because the compared systems instantiate different inference mechanisms, we interpret this as a system-level comparison and report computational cost separately in Appendix J. Invalid or unparsable outputs count as incorrect; reproduction details and invalid-output accounting appear in Appendix B.

SHUQI achieves **98.0%**, **100.0%**, and **100.0%** accuracy on ProofWriter, ProntoQA, and LogicalDeduction, respectively. The strongest reproduced baseline on each task reaches 88.5% with SymbCoT, 99.8% with AAI, and 85.7% with Symbolic-Aided CoT, corresponding to gains of **+9.5**, **+0.2**, and **+14.3** percentage points. Paired tests support the improvements on ProofWriter and LogicalDeduction, whereas the one-example difference on ProntoQA is not statistically resolved.

These system-level results do not explain where errors are corrected; RQ2 therefore controls representation and execution directly.

4.2 RQ2: Why Does Separating Representation and Execution Help?

RQ2 asks where error remains after canonical semantics are supplied, whether generative execution is stable, and why executable reasoning should be support aware.

Where does reasoning error remain? We compare Raw-NL generation, Same-IR generation, and deterministic execution for Qwen2.5-14B and DeepSeek-R1-Distill-Qwen-14B. Raw-NL and Same-IR hold the generative executor fixed, while Same-IR supplies a lossless serialization of the canonical task representation; deterministic execution operates on the corresponding structured IR.

Component	Controlled effect
Representation	Raw-NL → Same-IR: PQA **+35.8/+12.6 pp**; LD **+17.7/+16.0 pp**
Execution	Same-IR → deterministic: PQA **+2.0/+0.6 pp**; LD **+23.0/+18.0 pp**
Routing	Forced symbolic → SHUQI: **88.5%** → **98.0%**, **19** → **0** regressions
Replay	Corruption audit: **1,406/1,406** rejected

Table 3: **Component attribution.** Qwen/DeepSeek effects are reported in that order.

These comparisons are operational diagnostics under matched semantic content rather than a causal decomposition of model internals.

Both executors show the same qualitative task contrast. On ProntoQA, Same-IR raises Qwen from 62.2% to 98.0% and DeepSeek from 86.8% to 99.4%, leaving only 2.0- and 0.6-point gaps to deterministic execution. On LogicalDeduction, Same-IR reaches 77.0% and 82.0%, leaving much larger 23.0- and 18.0-point residual gaps. Deterministic execution reaches 100% on both tasks and corrects all remaining Same-IR errors. Thus, supplying canonical semantics produces substantial representation-associated recovery, but the remaining execution-side bottleneck is strongly task dependent.

Is generative execution stable once semantics are explicit? High Same-IR accuracy does not imply consistent execution of the supplied semantics. Under independently validated semantics-preserving transformations, predictions flip on $292/2{,}400$ Qwen conditions (12.2%, 95% CI [10.5, 13.8]) and $165/2{,}400$ DeepSeek conditions (6.9%, [5.7, 8.1]). When a validated semantic change requires a different answer, Qwen and DeepSeek track it on $2{,}856/3{,}484$ (82.0%, [80.2, 83.7]) and $3{,}187/3{,}484$ (91.5%, [90.8, 92.1]) conditions, respectively. Separately measured stale-prediction rates are 18.7% and 7.9%. Because multiple transformed conditions may originate from one source example, the intervals use source-clustered bootstrap.

As a reference control, deterministic execution yields $0/2{,}400$ semantics-preserving flips and tracks all $194/194$ independently validated semantic changes in the frozen primary set. A stricter audit fixes the canonical IR itself and varies only its lossless serialization. On 1,200 additional disjoint source examples, clean valid-answer disagreement reaches $370/1{,}200$ for Qwen (30.8%, [28.3, 33.5]) and $94/1{,}200$ for DeepSeek (7.8%, [6.4, 9.5]). The primary-set breakdown appears in Appendix E. Together, these controls show that, across the tested settings, high Same-IR accuracy does not guarantee stable semantic execution. A targeted cross-family replication with Mistral-Nemo-Instruct-2407 further shows that this post-IR instability is not confined to the Qwen lineage: identical-IR disagreement reaches 59.4% on ProntoQA and 72.7% on LogicalDeduction (Appendix M).

Why make execution support-aware? Replacing generative post-IR computation is useful when an executable representation is supported, but forcing symbolic execution under incomplete coverage can introduce new failures. On ProofWriter, forced symbolic execution improves the frozen generative baseline and fallback B_0 from 77.5% to 88.5%, fixing 41 baseline errors but introducing 19 regressions. SHUQI instead releases deterministic execution only for the 177 examples satisfying its support contract and falls back to B_0 on the remaining 23, reaching 98.0%. It therefore preserves all 41 symbolic fixes while introducing no regressions.

Case study. Figure 3 makes the aggregate diagnostics concrete: (a) `ProntoQA_301` is corrected by supplying canonical semantics; (b) `logical_deduction_32` remains wrong under Same-IR and is fixed only by deterministic execution; (c) `logical_deduction_297` yields different valid predictions across lossless presentations of identical canonical IR; and (d) `AttNonegNatLang-OWA-273_Q11` is compiler-incomplete, so forced symbolic execution fails while $H_{\mathrm{PW}} = 0$ preserves the correct B_0 fallback. The cases are illustrative rather than prevalence evidence.

4.3 RQ3: Which Components Drive SHUQI's Gains?

Representation and execution. Same-IR yields large representation-associated gains, especially on ProntoQA, but leaves 23.0- and 18.0-point LogicalDeduction gaps for Qwen and DeepSeek;

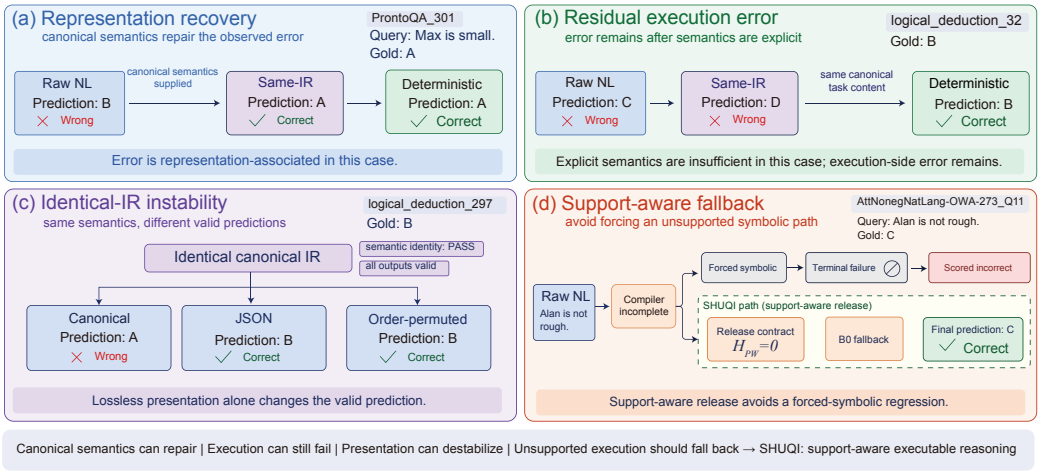


Figure 3: **Representative frozen cases behind the representation–execution diagnostics and support-aware routing.** (a) Canonical semantics repair a Raw-NL error. (b) Same-IR remains wrong while deterministic execution succeeds. (c) Lossless presentations of identical canonical IR yield different valid predictions despite passing semantic-identity checks. (d) Under incomplete compilation, forced symbolic execution terminates without a valid answer, whereas SHUQI rejects symbolic release and uses the correct fallback. Cases are selected mechanically within pre-specified outcome categories and are illustrative rather than prevalence evidence.

deterministic execution removes all remaining Same-IR errors. Their relative importance is therefore task dependent.

Support-aware routing. On ProofWriter, forced symbolic execution yields 41 fixes and 19 regressions, whereas SHUQI retains all 41 fixes with none. Two label-blind 177/23 matched-coverage controls average 87.22% and 87.21% (maximum 91.5%); none of 10,000 assignments per control reaches SHUQI's 98.0% ($p_{\mathrm{MC}} = 1/10001 < 10^{-4}$).

Replay validity. Replay does not change clean-set accuracy, but rejects all $1{,}406/1{,}406$ independently validated invalid certificate or symbolic mutations, with no false rejection in the bounded canonical audit. Thus replay provides execution validity rather than benchmark headroom (Appendix I).

Execution validation and cost. Deterministic execution agrees with independent reference semantics on all 849,684 tested decisions and requires no post-IR autoregressive calls (Appendices H and J).

5 Discussion and Limitations

Our results show that representation and execution contribute differently to reasoning error, motivating deterministic execution when supported and fallback otherwise. SHUQI assumes task-engineered executable representations and does not establish unrestricted language-to-IR coverage.

6 Conclusion

Final-answer accuracy is insufficient to diagnose reasoning. Our controlled representation–execution interface exposes distinct representation- and execution-side failures, while SHUQI uses deterministic execution when supported and generative fallback otherwise.

AI USE STATEMENT

Generative AI tools were used to assist with research ideation and hypothesis refinement; feedback on methodology and experimental design; code and software drafting, editing, and debugging; experimental-workflow support; organization and interpretation of results; and manuscript structuring, drafting, editing, citation checking, and submission preparation. The tools did not replace author review or verification. The authors reviewed and verified all AI-assisted code, experimental procedures, outputs, citations, interpretations, claims, and manuscript text, and take responsibility for the final content of this work.

Reproducibility Statement. Section 3 specifies the controlled representation–execution interface and SHUQI's task-grounded realization, while Section 4 documents the frozen evaluation universes, Same-IR diagnosis, semantic-stability controls, ProofWriter routing, and bounded implementation checks. The Appendix records task-specific compilation and execution (A), Same-IR and identical-IR protocols (C,E), semantic and routing controls (F,G), independent verification and replay audits (H,I), efficiency and statistics (J,K), frozen implementation details (L), and the Mistral-Nemo cross-family replication (M). Frozen manifests, serializer definitions, raw generations, hashes, and acceptance records are retained with the corresponding experiment artifacts.

REFERENCES

Benjamin Callewaert, Simon Vandevelde, and Joost Vennekens. Verus-lm: a versatile framework for combining llms with symbolic reasoning. *arXiv preprint arXiv:2501.14540*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

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*, pp. 10764–10799. PMLR, 2023.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Qingchuan Li, Jiatong Li, Zirui Liu, Mingyue Cheng, Yitong Zhou, Yuting Zeng, Qi Liu, and Tongxuan Liu. Are llms stable formal logic translators in logical reasoning across linguistically diversified texts? In *Proceedings of the ACM Web Conference 2026*, pp. 3633–3644, 2026.

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)*, pp. 305–329, 2023.

Lachlan McGinness and Peter Baumgartner. Automated theorem provers help improve large language model reasoning. *arXiv preprint arXiv:2408.03492*, 2024.

Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. In *International conference on machine learning*, pp. 7076–7087. PMLR, 2020.

George C Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 106–119, 1997.

Phuong Minh Nguyen, Tien Huu Dang, and Naoya Inoue. Non-interactive symbolic-aided chain-of-thought for logical reasoning, 2025. URL https://arxiv.org/abs/2508.12425.

Phuong Minh Nguyen, Dang Huu-Tien, and Naoya Inoue. Improving chain-of-thought for logical reasoning via attention-aware intervention. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 2917–2941, 2026.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176, 2023.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2025. URL https://arxiv.org/abs/2406.18665.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, 2023.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. Large language models meet symbolic provers for logical reasoning evaluation, 2025. URL https://arxiv.org/abs/2502.06563.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. Determlr: Augmenting llm-based logical reasoning from indeterminacy to determinacy. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9828–9862, 2024.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the association for computational linguistics: ACL-IJCNLP 2021*, pp. 3621–3634, 2021.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. Faithful logical reasoning via symbolic chain-of-thought. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 13326–13365, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Detailed Task-Specific Compilation and Execution

SHUQI uses task-grounded executable representations rather than a universal intermediate representation. For task t, the operational interface is

$$x_{\mathrm{NL}} \xrightarrow{C_t} z_t \xrightarrow{E_t} s_t \xrightarrow{D_t} y_t, \tag{1}$$

where C_t is the task-grounded compiler, z_t the executable representation, E_t the task-defined execution procedure, s_t its execution state, and D_t the terminal decision rule.

All three deployed compilers are deterministic task-engineered parsers for documented controlled languages. They are not learned formalizers and do not imply unrestricted natural-language coverage. Each compiler either constructs the task-specific representation required by the execution contract or exposes an unsupported condition rather than silently guessing an interpretation.

Execution evidence is defined relative to the trusted compiled symbolic input. It can establish that the represented problem was executed according to the task-defined semantics, but it does not establish that the source language was compiled faithfully. Thus, replay evidence addresses $z_{\mathrm{IR}} \to y$, not $x_{\mathrm{NL}} \to z_{\mathrm{IR}}$.

A.1 ProofWriter Compiler and Semantics

The ProofWriter compiler recognizes a fixed family of signed fact and rule templates. Supported atomic facts include positive and explicitly negated unary properties and binary relations. Supported implications include unary rules, two-antecedent unary rules, relational antecedents, and grounded conditional rules. The extended supported fragment additionally permits signed consequents and generic one-variable or grounded conditionals with at most two antecedents. Pronouns are resolved only through the rule-local *something*/*someone* binding introduced by the supported template; unlicensed or ambiguous references are rejected.

The compilation representation records the example identifier, signature, premises, query, source-formula mapping, constants, predicate symbols, diagnostics, and schema version. Grounding produces signed ground facts, ground implications, one-variable universal implications, the signed query, and the visible constant set.

Each signed literal records a predicate, arguments, and explicit polarity. Consequently, a negative premise is represented as an explicit negative literal rather than as absence of a positive fact.

Execution computes a finite additive fixed point. All source facts initialize the closure. Ground rules and all permitted instantiations of universal rules over visible constants are repeatedly applied until no new signed literal is derived. Execution never removes literals, does not introduce invisible domain constants, and rejects guard exhaustion rather than returning a partial closure.

For query literal q, let $\bar{q}$ denote the same atom with opposite polarity. The terminal symbolic state is

$$D_{\mathrm{signed}}(q) = \begin{cases} A, & q \in C,\ \bar{q} \notin C, \\ B, & q \notin C,\ \bar{q} \in C, \\ C, & q \notin C,\ \bar{q} \notin C, \\ \mathrm{BOTH}, & q \in C,\ \bar{q} \in C, \end{cases} \tag{2}$$

where C is the checked fixed-point closure.

A ProofWriter example is eligible for the symbolic route only after four separate conditions hold:

1. compilation is complete within the supported fragment;
2. integrity checks pass and the terminal prediction is valid;
3. the required proof or saturation replay succeeds; and
4. the routing gate accepts the resulting certified symbolic record.

These conditions are not interchangeable. Compilation can succeed without a route being accepted, for example if integrity, terminal-state, or replay requirements fail.

The frozen routing implementation distinguishes compiler-incomplete, integrity-failure, BOTH, and certificate-failure fallbacks. On the 200-example frozen ProofWriter universe, 177 examples pass the complete symbolic routing contract and 23 fall back to B_0. All 23 observed fallback cases are specifically compiler-incomplete fallbacks; the other defined fallback categories occur zero times in this frozen routing ledger.

A.2 PRONTOQA COMPILER AND SEMANTICS

The deployed ProntoQA adapter recognizes a narrower controlled language than the ProofWriter compiler. It supports fixed unary universal statements, plural-universal statements, and named signed facts. Explicit negative facts and negative consequents are represented by literal polarity. The deployed fragment does not support free paraphrase, binary predicates, function terms, equality, multi-antecedent rules, negative rule antecedents, or negation-as-failure.

The compilation-level representation records the example identifier, signature, premises, query, source-formula mapping, constants, predicate symbols, diagnostics, and schema version. The execution-level representation contains signed facts, ground rules, one-variable universal rules, the signed query, visible constants, and a canonical source-AST record.

Execution uses the same finite monotone closure contract as the supported signed-rule fragment above. Every one-variable rule is instantiated over the visible constants, and the computation terminates only after a checked fixed point is reached.

ProntoQA then applies its frozen task-specific decision policy. States A and B correspond to true and false respectively. For state C, where neither the query nor its explicit opposite is derivable, the checked fixed-point policy also returns false. Therefore, the supported policy is:

> If neither the query nor its explicit opposite is derivable at the checked fixed point, the answer is false.

This is a frozen ProntoQA task policy and should not be interpreted as a general assumption of implicit negation or unrestricted closed-world reasoning. State BOTH is treated as unresolved rather than forced into a task answer.

The frozen symbolic artifact contains an explicit `compiled=500` field, together with 500 certified and 500 correct examples, zero rejected examples, and zero failures in the denominator. Accordingly, the deployed compiler covers all 500 examples in the frozen ProntoQA evaluation universe. This statement is restricted to that frozen universe and does not imply general ProntoQA language coverage.

A.3 LOGICALDEDUCTION COMPILER AND SEMANTICS

The LogicalDeduction compiler recognizes the frozen 3-, 5-, and 7-object problem formats and the supported horizontal, age, price, and rank surface families. Entity declarations, fixed-position statements, relative-order statements, and answer options must match the documented templates. Production answer options compile to fixed-position constraints; unsupported or incompletely matched statements are rejected.

The canonical representation contains an ordering domain, entity variables, an ALLDIFFERENT constraint, the compiled ordering constraints, ordered answer options, and the frozen option-to-letter mapping.

Each entity is represented by an integer variable ranging from 0 to $N - 1$, where N is the number of entities. The deployed solver uses a bounded finite-domain encoding with a frozen solver seed and explicitly distinguishes satisfiable, unsatisfiable, unknown, timeout, and invalid states.

For a base constraint system B and answer option o_i, the MUST-BE-TRUE decision contract is

$$B \models o_i \quad \Longleftrightarrow \quad B \wedge \neg o_i \text{ is UNSAT.} \tag{3}$$

A final answer is returned only when exactly one option satisfies this task-defined acceptance condition. Multiple accepted options, no accepted option, or unresolved solver states are not coerced into an answer.

Method	Fidelity status	PW invalid	PQA invalid	LD invalid
SymbCoT	Qualified adaptation	1	0	78
Symbolic-Aided CoT	Faithful adaptation	0	0	9
AAI	Faithful adaptation	2	0	11
VERUS-LM	Faithful adaptation	37	322	20
MenTaL	Faithful adaptation*	67	107	78

Table 4: Reproduction fidelity and invalid-output accounting. All methods use the same Qwen3-14B BF16 non-quantized checkpoint but retain method-specific inference protocols. Invalid outputs remain in the evaluation denominator and count as incorrect. *MenTaL additionally requires the implementation qualifications described in the text.

The frozen 300-example artifact completes and certifies all 300 examples, reports 300 correct decisions, and records zero invalid-constraint, unsupported-question, solver-unresolved, missing, failed, or certificate-failure cases. Thus, the deployed compiler and execution interface support the complete 300-example frozen LogicalDeduction universe.

A.4 Formal Execution Contracts

The signed-rule tasks assume a trusted compiled signed theory satisfying the documented compiler contract, complete finite closure under the supported rule fragment, successful termination of the fixed-point guard, and application of the task-specific decision policy above. These requirements are execution contracts rather than claims about arbitrary natural-language inputs.

LogicalDeduction assumes a trusted compiled finite-ordering problem satisfying the documented compiler contract, preservation of the ordered answer options and option-to-letter mapping, and a unique terminal answer under the MUST-BE-TRUE test.

For all tasks, a valid execution certificate is defined only relative to the trusted compiled problem and the task-defined execution contract. Certificate validity therefore does not imply language-to-IR semantic faithfulness.

B Recent-Method Reproduction Details

The main end-to-end comparison uses local frozen reproductions rather than literature-reported benchmark scores. All reproduced methods are evaluated on exactly the same frozen universes used for SHUQI: 200 ProofWriter examples, 500 ProntoQA examples, and 300 LogicalDeduction examples.

All competitor runs use the same Qwen3-14B checkpoint in BF16 without quantization. This matches model weights and numerical precision, but not prompting, thinking mode, decoding, intervention, token budget, or solver/tool protocol. Those components remain method specific.

Invalid, unparsable, truncated, solver-failed, or otherwise non-answer outputs remain in the full denominator and count as incorrect. No failed record is removed or repaired for the main comparison.

SymbCoT. The local reproduction retains the staged translation, planning, solver, and verification procedure. The final reproduction audit classifies it as a qualified adaptation rather than an exact paper-and-code reproduction because the local answer parser rejects ambiguous multiple-label outputs that are handled more permissively by the released implementation. The frozen LogicalDeduction run contains 78 invalid outputs: 62 parsing failures and 16 generation truncations. The ProofWriter and ProntoQA runs contain one and zero invalid outputs, respectively.

Symbolic-Aided CoT. The reproduction retains the method-specific symbolic-aided inference structure, released formatting conventions, raw-completion behavior, and answer extraction while adapting the model interface to the common local backbone. Invalid counts are 0, 0, and 9 on ProofWriter, ProntoQA, and LogicalDeduction, respectively.

Method	ProofWriter	ProntoQA	LogicalDeduction
SymbCoT	177/200 (88.50%)	488/500 (97.60%)	39/300 (13.00%)
Symbolic-Aided CoT	164/200 (82.00%)	496/500 (99.20%)	257/300 (85.67%)
AAI	162/200 (81.00%)	499/500 (99.80%)	254/300 (84.67%)
VERUS-LM	154/200 (77.00%)	157/500 (31.40%)	248/300 (82.67%)
MenTaL	128/200 (64.00%)	193/500 (38.60%)	221/300 (73.67%)

Table 5: Frozen scores of the five reproduced recent methods on the matched evaluation universes. These are local results rather than literature-reported benchmark scores.

AAI. The local pipeline executes the released attention intervention rather than reducing the method to ordinary base prompting. Head statistics and selection, per-head intervention values, attention-mask modification, and adaptive attention registration are active in the frozen run. The intervention strength is 0.04. Invalid counts are 2, 0, and 11 on the three tasks.

VERUS-LM. The local reproduction preserves the released knowledge-base, question, information-extraction, inference, and solver pipeline while adapting model transport to the common Qwen checkpoint. Pipeline failures are retained rather than repaired. The frozen runs contain 37, 322, and 20 pipeline-invalid outputs on ProofWriter, ProntoQA, and LogicalDeduction, respectively. These invalids arise within the method-specific multi-stage pipeline rather than from post-hoc removal from the evaluation denominator; all such records remain counted as incorrect. A smaller solver/interface diagnostic subset is contained within these failures and is not added a second time.

MenTaL. The reproduction retains the released prompting and Prover9/Pyke/CSP solving paths. Implementation corrections address released task-dispatch behavior, 200-row truncation, and gold-bearing output serialization without replacing the method-specific solver or prompt. Invalid outputs are retained in the denominator rather than repaired or discarded. The frozen counts are 67, 107, and 78 on ProofWriter, ProntoQA, and LogicalDeduction, respectively; ProofWriter contains 3 parsing and 64 solver failures, with the remaining task-specific failure categories recorded in the frozen reproduction ledger.

C Same-IR Serialization and Inference Protocol

Same-IR controls represented task content while respecting each executor's native interface. For each supported ProntoQA or LogicalDeduction example, the production compiler first constructs canonical symbolic content. The deterministic executor receives the structured IR directly, while the generative executor receives a frozen lossless deterministic serialization of the same permitted instance-specific semantic fields.

Same-IR therefore matches canonical symbolic content but does not make the generative and deterministic executors identical in native modality. The Same-IR-to-deterministic comparison is interpreted as a residual execution-side gap under matched symbolic content, not as a pure executor-only causal intervention.

C.1 Permitted Information

Same-IR does not add the original natural-language problem, gold labels, deterministic predictions, derived closure facts, satisfying assignments, proof hints, replay certificates, counterfactual metadata, or any outcome-dependent information unless a field is part of the permitted canonical semantic representation itself.

Serialization, prompting, decoding, and parsing are frozen before confirmatory inference. No retry, repair, voting, prompt search, serializer selection, or result-conditioned regeneration is used.

C.2 Concrete Same-IR Example

For concreteness, we show frozen example `ProntoQA_60`, selected mechanically from the primary 500-example universe by minimum canonical serialization length, subject only to display constraints.

Raw natural-language input x.

```
Wumpuses are large. Each wumpus is an impus.
Each impus is not hot. Impuses are numpuses.
Every numpus is earthy. Zumpuses are mean.
Numpuses are vumpuses. Vumpuses are liquid.
Vumpuses are dumpuses. Dumpuses are not mean.
Dumpuses are tumpuses. Sam is a wumpus.

Question: Is the following statement true or false?
Sam is mean.
```

Permitted canonical content z (compact rendering).

```
entities = [sam]
facts    = [+Wumpus(sam)]

rules =
  +Wumpus(x) -> +Large(x)
  +Wumpus(x) -> +Impus(x)
  +Impus(x) -> -Hot(x)
  +Impus(x) -> +Numpus(x)
  +Numpus(x) -> +Earthy(x)
  +Zumpus(x) -> +Mean(x)
  +Numpus(x) -> +Vumpus(x)
  +Vumpus(x) -> +Liquid(x)
  +Vumpus(x) -> +Dumpus(x)
  +Dumpus(x) -> -Mean(x)
  +Dumpus(x) -> +Tumpus(x)

query = +Mean(sam)
decision_semantics = A_IF_QUERY_DERIVED_ELSE_B
```

Exact canonical Same-IR serialization $\sigma(z)$.

```
ENTITIES:
sam

FACTS:
+Wumpus(sam)

RULES:
+Wumpus(x) -> +Large(x)
+Wumpus(x) -> +Impus(x)
+Impus(x) -> -Hot(x)
+Impus(x) -> +Numpus(x)
+Numpus(x) -> +Earthy(x)
+Zumpus(x) -> +Mean(x)
+Numpus(x) -> +Vumpus(x)
+Vumpus(x) -> +Liquid(x)
+Vumpus(x) -> +Dumpus(x)
+Dumpus(x) -> -Mean(x)
+Dumpus(x) -> +Tumpus(x)

QUERY:
+Mean(sam)
```

Figure 4: **Concrete Same-IR walkthrough.** A frozen ProntoQA input is deterministically compiled into canonical signed semantics and serialized for generative Same-IR execution. The middle block is a compact rendering of the permitted canonical content; the final block is the exact stored canonical serialization. The Same-IR input excludes the gold answer, model predictions, derived closure, and replay evidence.

The middle block is a compact human-readable rendering of the permitted canonical content, whereas the final block reproduces the exact stored canonical Same-IR serialization supplied by the frozen serializer. Line breaks in the Raw-NL block are for display only. No gold label, model prediction, derived closure, or execution evidence is shown.

Here `A`/`B` in `decision_semantics` denote the frozen task-level decision vocabulary, not the gold answer for this example.

C.3 PRONTOQA SERIALIZATION

The frozen ProntoQA semantic projection contains the task/example type, entities, signed facts, ground and universal rules, the signed query, and the frozen task-level decision semantics. The canonical textual renderer exposes the instance-specific semantic fields through deterministic `ENTITIES`,

Task	Executor	Raw NL	Same-IR	Deterministic	Δ_{repr}	Δ_{exec}
ProntoQA	Qwen	62.2	98.0	100.0	+35.8	+2.0
ProntoQA	DeepSeek	86.8	99.4	100.0	+12.6	+0.6
LogicalDeduction	Qwen	59.3	77.0	100.0	+17.7	+23.0
LogicalDeduction	DeepSeek	66.0	82.0	100.0	+16.0	+18.0

Table 6: Complete controlled representation–execution trajectories. Values are accuracy percentages. Δ_{repr} is Same-IR minus Raw-NL accuracy and Δ_{exec} is deterministic minus Same-IR accuracy. These are operational diagnostic gaps rather than causal decompositions.

Task / Executor	Recovered	Regressed	Residual	Det. fixed
PQA / Qwen	183/189	4/311	10	10/10
PQA / DeepSeek	66/66	3/434	3	3/3
LD / Qwen	77/122	24/178	69	69/69
LD / DeepSeek	80/102	32/198	54	54/54

Table 7: Case-level transitions. "Recovered" denotes Raw-NL errors corrected under Same-IR; "Regressed" denotes Raw-NL-correct cases that become wrong; "Residual" denotes remaining Same-IR errors; "Det. fixed" reports how many residual errors are corrected by deterministic execution.

`FACTS`, `RULES`, and `QUERY` sections. The decision semantics are fixed by the task protocol rather than selected from the example outcome.

C.4 LogicalDeduction Serialization

The LogicalDeduction projection contains the ordering entities, constraints, ordered answer options, and option-to-letter mapping. Option order is not sorted or regenerated. The mapping remains fixed as A$\rightarrow$ 0, B$\rightarrow$ 1, and so on, and is explicitly checked by the serialization identity gate.

C.5 ProofWriter Boundary

The frozen Same-IR text serializer used in the controlled generative experiments is defined for ProntoQA and LogicalDeduction, not ProofWriter. ProofWriter instead exposes its canonical symbolic records through the typed theory, literal, rule, and certificate serialization primitives used by the symbolic and routing pipelines. We therefore do not report a ProofWriter Same-IR generative condition.

C.6 Canonical Reproduction

Under canonical Same-IR serialization, DeepSeek obtains 497/500 correct predictions on ProntoQA and 246/300 on LogicalDeduction. Qwen obtains 490/500 and 231/300, respectively. These frozen predictions are the Same-IR conditions used in the controlled representation–execution analysis.

D Complete Representation–Execution Results

The main paper visualizes the controlled Raw-NL$\rightarrow$Same-IR$\rightarrow$deterministic trajectories. Here we report the complete aggregate and case-level accounting.

Raw-NL-to-Same-IR changes are treated as representation-associated effects: the same generative executor is supplied with canonical symbolic content instead of the Raw-NL interface. Same-IR-to-deterministic differences are residual execution-side gaps under matched symbolic content. Neither comparison is interpreted as a causal decomposition of internal model mechanisms.

Task / Executor	Det.-only	IR-only	95% CI (pp)	McNemar p
PQA / DeepSeek	3	0	[0.00, 1.40]	0.25
PQA / Qwen	10	0	[0.8, 3.4]	1.95×10^{-3}
LD / DeepSeek	54	0	[13.67, 22.33]	1.11×10^{-16}
LD / Qwen	69	0	[18.33, 27.67]	3.39×10^{-21}

Table 8: Paired Same-IR versus deterministic execution. "Det.-only" and "IR-only" denote discordant correct cases; confidence intervals are for deterministic minus Same-IR accuracy.

D.1 Case-Level Error Transitions

For DeepSeek on ProntoQA, all 66 Raw-NL errors are recovered under Same-IR, while 3 of 434 Raw-NL-correct cases regress. These are the only three remaining Same-IR errors, and deterministic execution corrects all three.

For DeepSeek on LogicalDeduction, Same-IR recovers 80 of 102 Raw-NL errors, while 32 of 198 Raw-NL-correct cases regress. This leaves 54 Same-IR errors, all of which are corrected by deterministic execution.

Qwen exhibits the same qualitative task contrast: ProntoQA shows extensive Raw-error recovery and little residual execution-side error, whereas LogicalDeduction retains considerably more error after canonical symbolic content is supplied.

D.2 Paired Same-IR Versus Deterministic Statistics

The paired comparison is especially simple in the frozen universes because every Same-IR-correct example is also correct under deterministic execution: the IR-only discordant count is zero in all four cells. On ProntoQA, the remaining gap is small: deterministic execution corrects the 10 residual Qwen errors and the 3 residual DeepSeek errors. The Qwen difference is statistically resolved under the paired test, whereas the three-example DeepSeek difference is not. On LogicalDeduction, the residual gap is much larger: deterministic execution corrects all 69 Qwen and all 54 DeepSeek Same-IR errors, yielding 23.0- and 18.0-point gains respectively.

These results should not be read as evidence that the deterministic executor is a pure causal replacement for an otherwise identical model component. Same-IR and deterministic execution operate through different native interfaces. The controlled claim is narrower: once the permitted canonical task content is fixed, substantial error can remain under generative post-IR execution, and the task-defined deterministic executor resolves those residual errors in the studied universes.

E Identical-IR Serialization Stability

The preceding controlled interface matches canonical semantic content across generative and deterministic execution. This audit asks a stricter behavioral question: whether a generative executor remains invariant when the canonical IR itself is fixed but its lossless presentation changes.

E.1 Frozen Serializer Family

Three deterministic serializers were frozen before confirmatory inference:

1. canonical serialization;
2. JSON serialization of the permitted semantic projection; and
3. order-permuted canonical serialization with frozen seed 1729.

The JSON condition changes both serialization family and top-level presentation order. It is therefore not interpreted as a syntax-only intervention.

For every serialization, independent normalization and deserialization verify preservation of the permitted semantic fields. The source natural-language problem, gold label, deterministic answer,

Executor	Task	Clean disagreement
DeepSeek	ProntoQA	119/500 (23.8%)
Qwen	ProntoQA	50/500 (10.0%)
DeepSeek	LogicalDeduction	27/300 (9.0%)
Qwen	LogicalDeduction	95/300 (31.7%)

Table 9: Final strict source-level clean disagreement across the frozen lossless serializer family on the primary evaluation universes. Magnitude and sensitive presentation are model- and task-dependent.

Executor	Task	Alternative	Clean flip	Invalid-involved	C→W	W→C	W→diff. W
DeepSeek	PQA	JSON	117/500 (23.4%)	2/500 (0.4%)	117	2	0
DeepSeek	PQA	Order-permuted	5/500 (1.0%)	2/500 (0.4%)	4	3	0
DeepSeek	LD	JSON	17/300 (5.7%)	45/300 (15.0%)	13	47	1
DeepSeek	LD	Order-permuted	20/300 (6.7%)	41/300 (13.7%)	27	29	2
Qwen	PQA	JSON	30/500 (6.0%)	0/500 (0.0%)	21	9	0
Qwen	PQA	Order-permuted	30/500 (6.0%)	1/500 (0.2%)	22	9	0
Qwen	LD	JSON	68/300 (22.7%)	46/300 (15.3%)	57	34	13
Qwen	LD	Order-permuted	88/300 (29.3%)	17/300 (5.7%)	42	48	14

Table 10: Canonical-versus-alternative serialization contrasts. Clean flip and invalid-involved rates use the task universe. Correctness transitions count invalid as wrong; "diff. W" denotes a changed valid wrong answer.

derived closure, satisfying assignment, proof hint, certificate, and counterfactual metadata are excluded. For LogicalDeduction, answer-option order and option-to-letter mapping remain fixed across serializers.

Each primary generative executor is evaluated on three serializations of all 500 ProntoQA and 300 LogicalDeduction IRs, producing 2,400 serialized conditions per model. All 2,400 semantic-identity checks pass for each model. The canonical condition exactly reproduces the frozen Same-IR predictions: 497/500 and 246/300 for DeepSeek, and 490/500 and 231/300 for Qwen.

E.2 Primary-Set Clean Disagreement

We use the final strict source-level clean-disagreement definition. A source is counted as a clean disagreement only when all three frozen serializer outputs are valid parsed answers and at least two lossless presentations of the same canonical IR yield different answers. Invalid or truncated generations remain in the task universe but cannot create a clean disagreement.

The strict source-level summary above should be distinguished from the canonical-versus-alternative pairwise ledger below. The latter reports each pairwise condition directly and is retained without reinterpreting it as the strict all-three-valid source-level metric.

E.3 Disjoint-Source Replication

To test whether the primary-set instability is an artifact of the original evaluation universes, the final audit repeats the unchanged identical-IR protocol on 1,200 additional disjoint source examples. Clean valid-answer disagreement occurs for 370/1,200 Qwen examples (30.8%, 95% CI [28.3, 33.5]) and 94/1,200 DeepSeek examples (7.8%, [6.4, 9.5]). These replication sources are disjoint from the primary 500/300 task universes used in Tables 9–11.

These results do not imply that all lossless serializers are unstable, that all language models exhibit the same behavior, or that the observed differences identify an internal mechanism. They establish only that, within the frozen serializer family and tested executors, fixing canonical semantic content is insufficient to guarantee presentation-invariant generative prediction.

F Task-Semantic Stability Results

We test two complementary task-semantic requirements: predictions should remain invariant under reference-validated semantics-preserving transformations and should track independently validated

Executor	Task	Canonical	JSON	Order-permuted
DeepSeek	ProntoQA	0/500	2/500	2/500
DeepSeek	LogicalDeduction	38/300	15/300	31/300
Qwen	ProntoQA	0/500	0/500	1/500
Qwen	LogicalDeduction	6/300	42/300	11/300

Table 11: Invalid outputs by serializer. These condition-level counts remain in the experiment but cannot create clean valid-answer disagreement. DeepSeek invalids are max-token truncations or missing final answers; Qwen invalids are malformed strict final lines under the frozen parser.

Executor	Invariant	Prediction flips
Deterministic	2400/2400	0/2400
DeepSeek, 100-per-family subset	555/600	45/600 (7.5%)
Qwen, 100-per-family subset	507/600	93/600 (15.5%)

Table 12: Behavior under reference-validated semantics-preserving transformations. The deterministic row reports the exhaustive 2,400-condition control; the generative 600-condition rows are the frozen 100-per-family diagnostic subset used for the family-level breakdown below.

semantic changes. Transformation families and sampled subsets are frozen independently of model outcomes. Required inverse remappings are applied before prediction comparison.

F.1 Semantics-Preserving Transformations

ProntoQA transformations include entity renaming, predicate renaming, and fact/rule permutation. LogicalDeduction transformations include entity renaming, constraint permutation, and option permutation. These transformations are reference-validated as semantics preserving before model scoring. Prediction flips therefore measure behavioral failure of invariance under the tested equivalence controls; they do not identify an internal mechanism.

The frozen 100-per-family diagnostic subset contains 600 source–transformation conditions per model. Its breakdown is reported in Table 13.

The main-text aggregate additionally evaluates the same six transformation families over the complete frozen task universes: 500×3 ProntoQA conditions plus 300×3 LogicalDeduction conditions, for 2,400 transformed conditions per model. Under this exhaustive accounting, Qwen flips on 292/2,400 conditions (12.2%, 95% CI [10.5, 13.8]) and DeepSeek on 165/2,400 (6.9%, [5.7, 8.1]). Because multiple transformed conditions originate from the same source example, these intervals resample source examples as clusters. Deterministic execution produces 0/2,400 prediction flips under the same validated transformations.

F.2 Validated Semantic-Change Counterfactuals

Invariance alone is insufficient because a constant predictor could also remain invariant. We therefore construct counterfactual transformations intended to change the task-defined answer.

A transformed source example enters the primary counterfactual set only when an independent reference validates a different unique answer. Of 400 attempted source-level transformations, 194 satisfy this criterion. The remaining 206 are excluded before model scoring and are treated as not applicable rather than as model failures.

The main-text figure also reports an expanded condition-level semantic-change campaign containing 3,484 scored conditions per generative executor after independent semantic-change validation and frozen pre-score eligibility filtering. Under that condition-level accounting, Qwen tracks the required changed answer on 2,856/3,484 conditions (82.0%, 95% CI [80.2, 83.7]) and DeepSeek on 3,187/3,484 (91.5%, [90.8, 92.1]); stale-prediction rates are 18.7% and 7.9%, respectively. The 194-example tables above and the 3,484-condition aggregate correspond to different frozen accounting ledgers and are therefore reported separately rather than treated as interchangeable denominators. The

Task / Transformation	DeepSeek flip	Qwen flip
ProntoQA / Entity renaming	2/100	5/100
ProntoQA / Fact/rule permutation	2/100	7/100
ProntoQA / Predicate renaming	2/100	4/100
LogicalDeduction / Constraint permutation	15/100	33/100
LogicalDeduction / Entity renaming	17/100	24/100
LogicalDeduction / Option permutation	7/100	20/100

Table 13: Prediction flips for the frozen semantics-preserving transformation families. Each generative model–transformation cell contains 100 predeclared examples.

Executor	Tracks changed answer	Stale prediction
Deterministic	194/194 (100.0%)	0/194 (0.0%)
DeepSeek	172/194 (88.7%)	19/194 (9.8%)
Qwen	155/194 (79.9%)	30/194 (15.5%)

Table 14: Behavior on the frozen 194-example independently validated source-level semantic-change set. Tracking and staleness are not complements because a prediction may change to another incorrect or invalid answer.

exact condition membership is retained in the frozen semantic-change ledger used for the main-text aggregate.

Together, the invariance and semantic-change controls rule out the trivial interpretation that a stable executor merely emits a constant answer: deterministic execution is invariant when semantics are preserved and responsive when the validated answer must change, whereas both primary generative executors violate these requirements on nontrivial subsets.

G PROOFWRITER ROUTING PROTOCOL AND CONTROLS

ProofWriter is the partial-coverage setting in the main experiments. The frozen support-aware gate accepts a symbolic result only when compilation is complete, integrity checks pass, a valid non-BOTH terminal decision is produced, and the required replay certificate succeeds. Otherwise the unchanged B_0 generative prediction is used.

The routing input contains no gold label, branch-correctness signal, or model confidence. The gate therefore cannot compare candidate branches by their eventual correctness.

On the frozen 200-example universe, all 23 fallback decisions are compiler-incomplete fallbacks. No integrity, BOTH, or certificate-failure fallback occurs in this ledger.

G.1 MATCHED-COVERAGE CONTROLS

To test whether the result follows merely from using the symbolic branch on 177 of 200 examples, we evaluate two label-blind fixed-coverage randomizations:

1. direct random selection of 177 symbolic positions; and
2. permutation of the frozen 177/23 routing mask.

Each implementation is evaluated over 10,000 assignments. Their mean accuracies are 87.21755% and 87.21025%, respectively. The maximum observed accuracy is 91.5% in each run, and no sampled assignment reaches SHUQI's 98.0%.

The two procedures implement the same theoretical 177-of-200 matched-coverage null and are treated as replications of that null rather than as statistically independent evidence. Using the Monte Carlo $+1$ correction,

$$p_{\mathrm{MC}} = \frac{0+1}{10000+1} = \frac{1}{10001} < 10^{-4}. \tag{4}$$

We therefore do not report the empirical tail probability as $p = 0$.

Task	Counterfactual	DS track	DS stale	Qwen track	Qwen stale
ProntoQA	Removal	45/46	1/46	40/46	5/46
ProntoQA	Polarity change	91/100	9/100	86/100	13/100
LogicalDeduction	Reversal	36/48	9/48	29/48	12/48

Table 15: Breakdown of the independently validated 194-example semantic-change source set. Deterministic execution tracks all 194 validated changes, including all 48 LogicalDeduction reversals.

Route	Accuracy	Symbolic / fallback	Fixes / regressions
Always B_0	77.5%	0 / 200	0 / 0
Always symbolic	88.5%	200 / 0	41 / 19
Support-aware SHUQI	98.0%	177 / 23	41 / 0

Table 16: Complete routing comparison on the frozen ProofWriter universe. Fixes and regressions are measured relative to the frozen B_0 predictions.

G.2 Certificate Coverage and Deployment Accounting

Only the 177 supported symbolic routes carry execution evidence relative to their compiled symbolic inputs. The 23 B_0 fallback predictions are not certificate-covered.

In the frozen online-routing simulation, the 23 fallback examples correspond to 23 unique B_0 callbacks, while all 200 final routed predictions reproduce the frozen H1 outputs.

This deployment-style accounting should not be confused with the historical full-paired experimental campaign, which intentionally executed multiple branches and stages for diagnosis and accumulated 1,110 physical model/stage calls. Those calls describe the experimental campaign rather than deployment requirements.

The routing result is retrospective on one frozen ProofWriter universe and does not establish prospective routing generalization.

H Finite-Fragment Verification

Because deterministic execution serves as a diagnostic reference in the main paper, we independently validate its implementation over bounded symbolic fragments.

The reference implementations do not import the production transition or decision modules. A source-level self-test scans imports and fails if forbidden production modules are used. LogicalDeduction shares only the immutable constraint representation required as a serialization schema; its independent reference does not import the production Z3 encoder, solver, question decision, or model checker.

The verification begins from trusted canonical IR. It therefore tests implementation agreement over bounded symbolic semantics rather than language-to-IR compilation.

Across all 849,684 semantic decisions, the deployed and independent reference implementations disagree zero times.

The ProofWriter fragment is deliberately narrow, including bounded ground signed-Horn cases over a restricted entity and predicate universe. The ProntoQA and LogicalDeduction fragments are likewise bounded according to the frozen verification manifest. These checks are not formal verification of unrestricted benchmark correctness.

The frozen verification protocol SHA is

`6a2f3f6010aa850789de4ecfdea0d8f5cb971ae22e89fa6d36d2d262745e7891`.

I Certificate Replay and Corruption Audit

Execution produces task-specific replay artifacts rather than a universal certificate format.

Task	Worlds / sets	Semantic decisions	Mismatches
ProntoQA	51,803	766,664	0
LogicalDeduction	4,622	81,216	0
ProofWriter	451	1,804	0
Total	56,876	849,684	0

Table 17: Agreement between the deployed deterministic executors and independently implemented bounded reference semantics.

I.1 Signed-Rule Proof Certificates

A base-fact certificate records the signed literal, source fact identifier, and derivation depth zero. A rule-application certificate records:

- the source rule identifier;
- the source universal-formula identifier when applicable;
- the instantiated rule;
- the substitution;
- antecedent literals;
- references to antecedent certificates;
- the derived consequent; and
- derivation depth.

Replay rebuilds the trusted fact and rule lookup from the compiled symbolic problem, checks source binding, validates antecedent references and consequents, replays universal substitutions where required, detects missing or cyclic certificate references, and recursively validates dependency depth.

I.2 Saturation Certificates

For the signed-rule neither state, the saturation certificate records:

- the final closure;
- the ground rules examined;
- the universal-rule/constant pairs examined;
- the fixed-point pass count;
- a fixed-point confirmation flag;
- the query and opposite literals; and
- their absence from the final closure.

The checker binds the certificate to the trusted query and opposite, verifies that trusted source facts occur in the closure, reconstructs the complete trusted ground-rule and universal-instantiation universe, checks closure under every examined rule, confirms absence of both queried polarities, and verifies the claimed fixed point.

The rule ordering is part of the frozen certificate contract; incomplete or reordered examined-rule universes are not silently accepted as equivalent certificates.

I.3 LogicalDeduction Certificates

A satisfiable LogicalDeduction certificate records the problem hash, entity order, satisfying assignment, and solver fingerprint.

An unsatisfiable certificate records the problem hash, normalized problem, constraint hashes, solver fingerprint, unsatisfiable core, operation, and solver transcript.

SAT replay checks exact problem binding, solver identity, entity order, a complete permutation assignment, and satisfaction of the constraints using an independent Python evaluator.

UNSAT replay requires every referenced core constraint to belong to the trusted top-level problem and independently checks that the supplied witness is itself unsatisfiable. Full replay additionally binds the canonical problem, constraint hashes, solver identity, operation, transcript, and final bounded problem.

The verifier does not require one arbitrary unsatisfiable-core serialization: historical, non-minimal, reordered, and duplicate-containing valid witnesses are accepted when they satisfy the trusted witness property.

I.4 Canonical Replay Acceptance

Across the three benchmark fragments, the frozen audit contains

$$743{,}208 + 81{,}216 + 1{,}584 = 826{,}008 \tag{5}$$

canonical benchmark-fragment certificate checks with zero false rejects. Including 245 standalone canonical checks gives 826,253 total canonical checks, again with zero false rejects.

I.5 Invalid-Certificate Corruption Audit

The corruption audit generates 1,406 semantically invalid certificate or symbolic mutations; all 1,406 are rejected. An additional 125 candidates (40 saturation-binding candidates and 85 core candidates) are excluded because reference validation shows that they are not genuinely invalid under the relevant contract. These excluded candidates are not counted as verifier successes or failures.

I.6 Valid-Witness Metamorphic Audit

For LogicalDeduction, 155 alternative valid duplicate or reordered witness forms are accepted. Across 1,704 metamorphic checks, the audit observes zero genuine semantic violations.

One semantics-preserving saturation-order edit is rejected because ordering is part of the frozen certificate-binding contract; it is tracked separately and is not counted among the 1,406 semantic-invalid mutations.

I.7 Certificate Boundary

Replay validates

$$z_{\mathrm{IR}} \to y \tag{6}$$

relative to the trusted compiled symbolic input. It does not validate

$$x_{\mathrm{NL}} \to z_{\mathrm{IR}}. \tag{7}$$

A valid replay artifact can therefore certify correct execution of a miscompiled problem without certifying the source-language interpretation.

J Full Efficiency Accounting

The efficiency comparison begins only after canonical executable IR is available. It therefore measures post-IR execution cost rather than end-to-end system cost. Representation construction, compiler engineering, and upstream language processing are outside the comparison.

Once canonical IR is available, deterministic execution requires no additional autoregressive model calls or generated tokens. Generative Same-IR execution instead continues decoding from the same supplied symbolic content.

The recorded wall times arise from different execution mechanisms and environments and are therefore not divided into an "$\times$ faster" ratio.

Task	Executor	Calls	Generated tokens	Recorded time	Accuracy
ProntoQA	Same-IR DeepSeek	500	368,002	10,587.58 s	497/500 (99.4%)
ProntoQA	Deterministic	0	0	0.116457 CPU s	500/500
LogicalDeduction	Same-IR DeepSeek	300	594,437	17,806.33 s	246/300 (82.0%)
LogicalDeduction	Deterministic	0	0	4.090447 CPU s	300/300

Table 18: Post-IR execution accounting. Recorded times are absolute measurements from the frozen runs and are not presented as hardware-normalized or end-to-end speedups.

Task	Comparator	Both correct	S-only / C-only	Δ acc.	Exact p
ProofWriter	SymbCoT	174	22 / 3	+9.50	1.57×10^{-4}
ProntoQA	AAI	499	1 / 0	+0.20	1.0
LogicalDeduction	Symbolic-Aided CoT	257	43 / 0	+14.33	2.27×10^{-13}

Table 19: Paired comparison between SHUQI and the strongest reproduced comparator on each frozen task universe. "S-only / C-only" denotes cases correct only for SHUQI / only for the comparator. Both-wrong counts are 1, 0, and 0, respectively.

For ProofWriter, support-aware routing sends 177 examples through the symbolic path and 23 through the frozen B_0 fallback. The cached deployment-style simulation invokes B_0 for 23 unique fallback examples. The historical full-paired diagnostic campaign, by contrast, intentionally executed multiple branches and accumulated 1,110 physical model/stage calls, 653,747 generated tokens, and 20,801.633 seconds of recorded generation time. These two accounting regimes should not be conflated.

K Statistical Procedures and Complete Paired Results

All primary end-to-end comparisons are paired on identical frozen examples. Paired confidence intervals use 100,000 bootstrap resamples with frozen seed 1729. Exact two-sided McNemar tests use the discordant paired counts.

For the final recent-method comparison, Holm correction is applied over the predeclared family of 15 comparisons: five accepted reproduced methods across three tasks.

When multiple transformed conditions originate from the same source example, as in the expanded semantic-stability audits, confidence intervals resample source examples as clusters rather than treating derived conditions as independent observations.

K.1 Strongest Reproduced Comparator on Each Task

On ProofWriter, the paired difference relative to SymbCoT is +9.5 percentage points with 95% CI [+5.0, +14.5]. The exact two-sided McNemar value is $p \approx 1.5652 \times 10^{-4}$ and the Holm-adjusted value is approximately 6.2609×10^{-4}.

On ProntoQA, SHUQI and AAI differ on only one example. The accuracy difference is +0.2 points with 95% CI [0.0, 0.6] and exact McNemar $p = 1.0$; the Holm-adjusted value is also 1.0. This experiment therefore does not statistically resolve an advantage over AAI.

On LogicalDeduction, the difference relative to Symbolic-Aided CoT is +14.33 percentage points with 95% CI [+10.33, +18.33]. The exact McNemar value is approximately 2.2737×10^{-13} and the Holm-adjusted value is approximately 1.8190×10^{-12}.

K.2 Representation–Execution Comparisons

The Same-IR versus deterministic paired counts, confidence intervals, and McNemar tests are reported in Appendix D.2. Those statistics quantify residual execution-side differences under matched symbolic content and should not be interpreted as pure causal estimates of an internal executor component.

K.3 Matched-Coverage Routing Randomization

For ProofWriter, no assignment among 10,000 sampled matched-coverage routings attains SHUQI's 98.0%. Applying the Monte Carlo $+1$ correction gives

$$\frac{1}{10001} < 10^{-4}. \tag{8}$$

The value is not reported as zero. The two routing randomizations are alternative implementations of the same 177-of-200 null and are not counted as independent confirmatory tests.

L Implementation and Frozen Configuration

All primary experimental universes are frozen and record aligned before manuscript analysis. Gold labels are not inputs to compilation, deterministic execution, replay verification, Same-IR serialization, transformation selection, or support-aware routing.

L.1 Evaluation Universes

The main frozen universes contain:

- 200 clean disjoint ProofWriter examples;
- 500 ProntoQA examples; and
- 300 LogicalDeduction examples.

Invalid model outputs remain in the relevant evaluation denominator rather than being dropped.

L.2 Generative Executors for Controlled Diagnosis

The primary representation–execution and semantic-stability diagnostics use two generative executors: Qwen2.5-14B and DeepSeek-R1-Distill-Qwen-14B. Raw-NL and Same-IR comparisons hold the generative executor fixed within each model.

We additionally run a targeted cross-family robustness replication with Mistral-Nemo-Instruct-2407. The Mistral post-IR audit reuses the frozen serializer family, task-semantic transformations, semantic-change controls, and scoring rules. A separate matched-answer-channel diagnostic uses constrained legal-label decoding to qualify the representation-side comparison; full details appear in Appendix M.

Deterministic execution is task defined and does not depend on which generative executor is used for the corresponding generative condition.

L.3 Recent-Method Backbone

The five reproduced recent methods use the same local Qwen3-14B checkpoint in BF16 without quantization. Matched backbone does not mean matched inference: the reproduced methods retain different prompting, thinking settings, decoding procedures, attention interventions, generation budgets, and solver/tool interfaces.

The final frozen reproduction audit is accepted with qualification rather than as a claim of uniform exact reproduction across all methods.

L.4 Frozen Serializer Configuration

The identical-IR audit uses the frozen canonical, JSON, and order-permuted serializers. The order-permuted serializer uses seed 1729. Serializer choice is fixed before confirmatory inference and is not selected using prediction outcomes.

All permitted-field identity checks are performed before interpreting cross-serialization disagreement.

Task	Raw-NL	Same-IR	Deterministic	Δ_{repr}	Δ_{exec}
ProntoQA	96/500 (19.2%)	260/500 (52.0%)	500/500	+32.8	+48.0
LogicalDeduction	157/300 (52.3%)	95/300 (31.7%)	300/300	−20.7	+68.3

Table 20: Native Mistral-Nemo representation–execution trajectory. The Raw-NL condition contains 390/500 invalid outputs on ProntoQA and 38/300 on LogicalDeduction, whereas Same-IR contains zero invalid outputs. The Raw-NL–Same-IR differences are therefore not used as matched cross-family estimates of representation-associated recovery.

L.5 Statistical Configuration

Paired bootstrap confidence intervals use 100,000 resamples with seed 1729. Exact two-sided McNemar tests use the paired discordant counts. The final recent-method Holm family contains 15 method–task comparisons. Source-clustered bootstrap is used whenever multiple derived conditions originate from one source example.

L.6 Frozen-Artifact Policy

The primary Qwen/DeepSeek predictions, inference ledgers, routing decisions, serializer definitions, statistical outputs, verification summaries, and condition manifests used for the reported results were frozen before their manuscript interpretation. Intermediate recent-method campaign reports superseded by the final acceptance audit are not used as evidence for the final comparison.

The Mistral-Nemo robustness experiments are retained as separate artifact families. Their full diagnostic campaign used the runtime-calibrated continuation procedure described in Appendix M.4; no outcome or gold information entered source selection, prompts, caching, retries, or the runtime-only continuation decision.

Frozen serializer definitions, exact IR universes, semantic-identity ledgers, raw generations, frozen pre-score predictions, model-specific canonical reproductions, hashes, and independent acceptance records are retained with the corresponding experiment artifacts.

M Cross-Family Mistral-Nemo Replication

The primary controlled diagnostics use Qwen2.5-14B and DeepSeek-R1-Distill-Qwen-14B, which share the Qwen model family. We therefore conduct a targeted cross-family robustness replication with Mistral-Nemo-Instruct-2407. The purpose is not to rank executors or claim matched inference budgets, but to test whether the observed post-IR instability is confined to the original model family.

M.1 Native Full Diagnostic Campaign

Using the frozen task universes and canonical Same-IR protocol, the native Mistral trajectory is:

The corresponding case transitions are 207 recoveries, 43 regressions, and 240 residual Same-IR errors on ProntoQA, and 27 recoveries, 89 regressions, and 205 residual Same-IR errors on LogicalDeduction. Deterministic execution corrects all 240 and 205 residual errors, respectively.

Because Raw-NL invalidity differs sharply from Same-IR invalidity, particularly on ProntoQA, this native trajectory mixes semantic-input effects with answer-format compatibility. We therefore use a separate matched-answer-channel diagnostic below to qualify the representation-side comparison.

M.2 Post-IR Stability Replication

The post-IR replication reuses the frozen canonical task representations, lossless serializer family, semantics-preserving transformation manifests, validated semantic-change conditions, and scoring rules used in the primary audits.

For the identical-IR audit, all $2{,}400/2{,}400$ semantic-identity checks pass. The canonical, JSON, and order-permuted conditions contain zero invalid outputs and zero truncations.

Metric	ProntoQA	LogicalDeduction
Clean identical-IR disagreement	297/500 (59.4%)	218/300 (72.7%)
Canonical accuracy	260/500 (52.0%)	95/300 (31.7%)
JSON accuracy	268/500 (53.6%)	75/300 (25.0%)
Order-permuted accuracy	249/500 (49.8%)	104/300 (34.7%)

Table 21: Mistral-Nemo identical-IR cross-family replication. All three serializations preserve the permitted canonical semantic content and all 2,400 serialized outputs are valid.

Task	Raw-NL	Same-IR	Δ_{repr}	95% CI
ProntoQA	278/500 (55.6%)	272/500 (54.4%)	-1.20 pp	$[-5.80, +3.40]$
LogicalDeduction	117/300 (39.0%)	103/300 (34.3%)	-4.67 pp	$[-10.00, +0.67]$

Table 22: Format-controlled Mistral-Nemo Raw-NL versus Same-IR diagnostic. Raw-NL and Same-IR use the same constrained legal-label answer channel and both have zero invalid outputs.

Clean valid-answer disagreement occurs on 297/500 ProntoQA sources (59.40%, 95% CI [55.04, 63.62]) and 218/300 LogicalDeduction sources (72.67%, [67.36, 77.40]). Because all three serializer outputs are valid, these disagreements cannot be attributed to invalid parsing or truncation.

Under the frozen semantics-preserving transformations, Mistral-Nemo changes prediction on 709/2,400 conditions (29.54%, 95% CI [27.21, 31.92]). The corresponding post-IR conditions contain zero invalid outputs.

On the expanded validated semantic-change campaign, Mistral-Nemo tracks the required changed answer on 2,401/3,484 conditions (68.92%, 95% CI [64.81, 72.88]). The separately defined stale-prediction criterion fires on 2,987/3,484 conditions (85.73%, [83.31, 88.02]). Tracking and staleness are not complements: a prediction may change while remaining incorrect or may satisfy the stale criterion under the frozen condition-level accounting.

These results provide cross-family evidence that fixing canonical semantic content does not guarantee presentation-invariant or transformation-invariant generative execution. The magnitude is clearly model- and task-dependent, and the experiment is not interpreted as a matched-backbone comparison.

M.3 Matched-Answer-Channel Qualification

To separate native answer-format invalidity from the Raw-NL–Same-IR comparison, we run a second Mistral diagnostic in which both conditions use the same constrained legal-label output channel. The decoder implements an exact complete-label prefix trie with forced EOS. This yields zero invalid outputs in all four task/condition cells.

On ProntoQA, Same-IR yields 64 recoveries and 70 regressions relative to Raw-NL, leaving 228 residual errors; the exact two-sided McNemar test gives $p = 0.665950$. On LogicalDeduction, Same-IR yields 25 recoveries and 39 regressions, leaving 197 residual errors; McNemar gives $p = 0.103422$. Both paired bootstrap confidence intervals include zero.

Thus, under the matched answer channel, we find no statistically resolved Same-IR improvement for Mistral-Nemo on either task. Representation-associated recovery should therefore not be treated as universal across executors.

The constrained answer channel itself changes the decoding distribution and is not a neutral correction of the native run. Among outputs that were valid in the previous native campaign, agreement with the format-controlled predictions is 60/110 (54.55%) for ProntoQA Raw-NL, 366/500 (73.20%) for ProntoQA Same-IR, 118/262 (45.04%) for LogicalDeduction Raw-NL, and 139/300 (46.33%) for LogicalDeduction Same-IR. Accordingly, this experiment is interpreted as a matched-answer-interface diagnostic rather than a corrected native Mistral trajectory.

Despite the absence of a significant representation-side improvement, substantial residual gaps remain from matched-answer Same-IR to deterministic execution: 45.6 percentage points on ProntoQA and

65.7 points on LogicalDeduction. This is consistent with the cleaner post-IR stability replication above.

M.4 Runtime Calibration and Execution Order

The full Mistral diagnostic used a runtime-calibrated continuation procedure. After an initial batch, observed throughput was used to determine whether the complete frozen manifests could be evaluated within the available compute budget. An interim scoring artifact existed before this runtime-only continuation calculation was finalized.

The continuation decision itself used throughput only. Source sets, serializer definitions, transformation manifests, prompts, decoding configuration, and evaluation criteria were already fixed, and no gold label, correctness statistic, disagreement rate, or other outcome metric was consulted when deciding whether to continue. The campaign subsequently completed the full frozen universes and manifests.

We therefore treat the Mistral experiment as a targeted cross-family robustness replication rather than as part of the primary pre-score confirmatory campaign.