SL ALGEBRA: A CONTROL IR FOR STRUCTURED, GROUNDED, AND AUDITABLE LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model inference increasingly combines heterogeneous controls, including structured-output constraints, retrieval, tool execution, and runtime policies. Existing systems typically implement these mechanisms as separate prompts, decoding processors, or callbacks, making their composition difficult to reason about and optimize. We introduce State-Logit (SL) Algebra, a control intermediate representation that separates state operators, which modify auxiliary state or conditioning context, from logit operators, which transform next-token scores. This representation enables formal reasoning about control plans, semantic-preserving rewrites, alternative physical lowerings, and execution traces.

We formalize non-anticipatory plans and multiple levels of equivalence, derive normalization laws for exact hard masks, and prove a tokenization-lift theorem for byte-level interface constraints. We further represent grounding as a logical join independent of its physical realization. Experiments across Qwen2.5 and Qwen3 models show that exact constraints achieve 100% structured-output validity, explicit grounding substantially improves QA exact match, and mask fusion provides a mean 1.55× runtime speedup. These results demonstrate that separating control semantics from physical realization enables both formal guarantees and practical optimization for controlled LLM inference. The implementation code is available at https://anonymous.4open.science/r/SL-Algebra-3A45.

1 INTRODUCTION

Deployed LLMs are rarely judged only by whether their text is plausible. In software systems, they must often emit a valid interface object, answer from approved evidence, and expose enough provenance for debugging or audit. A response that is semantically reasonable but returns malformed JSON, omits citations, or silently bypasses retrieval can still fail the application. This mismatch between "good text" and "correct controlled behavior" is now a first-order systems problem.

Current practice addresses these requirements with a heterogeneous stack of prompt instructions, constrained decoding, retrieval-augmented generation, tool calling, and custom logits processors (Lewis et al., 2020; Beurer-Kellner et al., 2023; Khattab et al., 2023; Zheng et al., 2024; Beurer-Kellner et al., 2024; Koo et al., 2024; Geng et al., 2025; Ugare et al., 2025; Dong et al., 2024). These mechanisms can work well in isolation, but in real systems they are typically composed as order-sensitive runtime logic. That makes three things difficult. First, *reasoning*: when do two controls commute, when does one subsume another, and what property does their composition preserve? Second, *optimization*: the logical control policy is specified in one layer, while the true costs arise elsewhere, e.g., Python callbacks, cache invalidation, device synchronization, and batching disruption (Yu et al., 2022; Kwon et al., 2023; Dao et al., 2022; Dao, 2024). Third, *auditability*: when outputs drive downstream software or high-stakes decisions, it matters not only what was generated, but also which controls fired and what evidence entered the computation.

We argue that the missing abstraction is an explicit *control intermediate representation (IR)* for autoregressive decoding. We propose **State-Logit (SL) Algebra**, whose key idea is to separate *state operators*, which modify auxiliary state or the effective conditioning context, from *logit operators*, which transform the next-token distribution. A retrieval call and a schema mask may both influence the next token, but they do so through different causal channels. Treating that distinction as a

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.

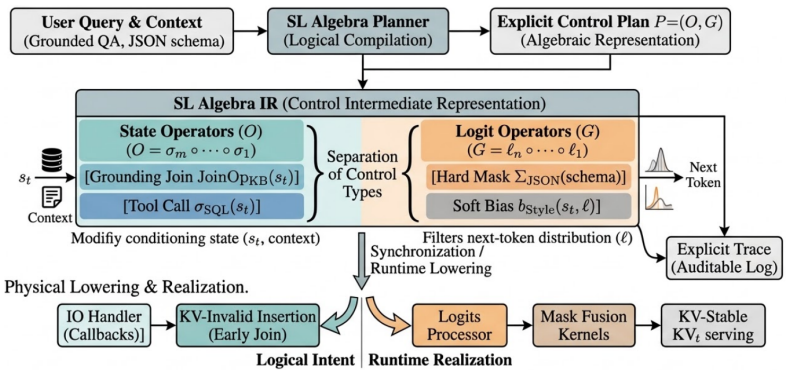

Figure 1: **SL Algebra at a glance.** A control plan separates logical *state operators* σ (e.g., retrieval, tools, memory updates) from *logit operators* ℓ (e.g., hard masks, finite biases). The same logical plan may admit multiple physical lowerings with different KV implications and runtime costs.

first-class semantic boundary turns heterogeneous runtime logic into an explicit plan that can be checked, rewritten, lowered, and traced.

The separation serves three purposes: formal reasoning about controlled decoding, a clean boundary between logical intent and physical lowering (especially for retrieval and KV reuse), and first-class provenance. Figure 1 illustrates this boundary. Our formal results isolate what the runtime guarantees; our experiments are diagnostic rather than leaderboard-oriented and answer three questions: (i) does treating formal constraints as first-class logit operators improve *strict* correctness when outputs must satisfy a language? (ii) does representing grounding as an explicit join improve grounded QA beyond constraints alone? and (iii) does an explicit control IR reveal optimizations and bottlenecks that otherwise remain hidden? Across seven Qwen2.5 and Qwen3 models, the answer to all three is yes: explicit hard constraints convert loose task capability into strict correctness on structured tasks; explicit joins substantially improve grounded QA; exact mask fusion yields a measurable operator-level speedup; and trace telemetry localizes the cost of Guarded Join to a single, cacheable lowering step rather than leaving it as an opaque slowdown.

Contributions.

- **C1 (representation):** Separating state operators from logit operators under non-anticipation yields a usable control IR for autoregressive inference (Sections 3–5); falsified if key controls cannot be represented without collapsing the distinction.
- **C2 (formal properties):** In the exact hard-mask fragment, normalization and tokenization-lift guarantees hold (Section 4, Appendix B); falsified by a counterexample to Proposition 4.2 or Theorem 4.3.
- **C3 (logical/physical separation):** Grounding as an explicit logical join, separate from its physical lowering, is empirically necessary (Tables 2 and 4); falsified if Join-Early and logit-additive join have indistinguishable quality/cost profiles under matched plans.
- **C4 (runtime value):** Explicit control yields measurable wins and diagnosable bottlenecks (Figures 3 and 4, Table 9); falsified if mask fusion shows no gain or if Guarded Join overhead cannot be localized.

2 Related Work

SL Algebra sits at the intersection of constrained decoding, LLM programming systems, retrieval-augmented generation, and efficient serving. The primitives are known; the contribution is a shared *control layer* that makes heterogeneous controls explicit, composable, optimizer-visible, and auditable.

Constrained decoding and structured generation. A large body of work enforces output structure by compiling schemas, regular languages, or grammars into token-level masks during decoding

(Hokamp & Liu, 2017; Geng et al., 2023; Beurer-Kellner et al., 2024; Koo et al., 2024; Geng et al., 2025; guidance-ai, 2026; dottxt-ai, 2026). This literature makes clear that formal validity and task correctness are distinct axes. SL Algebra builds on this line of work by treating hard constraints as first-class *logit operators* and making their algebraic properties explicit.

Programming systems for LLM pipelines. LMQL, DSPy, SGLang, and related systems improve the programmability of LLM pipelines by exposing higher-level control flow, prompt templates, or runtime abstractions (Beurer-Kellner et al., 2023; Khattab et al., 2023; Zheng et al., 2024). Recent tool-augmented benchmarks and training frameworks further emphasize explicit API interaction as a first-class systems concern (Li et al., 2023; Qin et al., 2024; Schick et al., 2023). We are complementary to these interfaces. Our focus is not the end-user language, but the lower control IR on which operator legality, equivalence, lowering, and trace semantics can be stated explicitly.

Retrieval and active grounding. RAG systems condition generation on external corpora to improve factuality and reduce unsupported responses (Lewis et al., 2020; Karpukhin et al., 2020; Izacard & Grave, 2021; Jiang et al., 2023; Asai et al., 2024; Izacard et al., 2023; Shi et al., 2024; Ye et al., 2024; Singh et al., 2025; Oche et al., 2025). Closely related lines on citation-aware generation and post-hoc revision also highlight the need to separate evidence access from generation-time control (Gao et al., 2023b;a). Many systems retrieve once and prepend the result to the prompt; others perform retrieval during generation. We make this design choice explicit by separating the *logical* join from its physical realization and by tracking its KV implications.

Serving systems and observability. Efficient LLM runtimes rely on batching, attention optimization, and KV reuse (Yu et al., 2022; Kwon et al., 2023; Dao et al., 2022; Dao, 2024). Meanwhile, production stacks increasingly expose traces and observability metadata (OpenAI, 2026; LangChain, 2026). SL Algebra connects these threads. Once control is represented explicitly, legality checks, rewrites, lowerings, and traces become properties of the plan rather than incidental implementation behavior.

Positioning. The closest prior systems expose either higher-level programming interfaces for LLM pipelines or specialized runtimes for one control mechanism at a time. Our goal is the missing middle layer: an IR on which one can simultaneously state admissibility guarantees, legality conditions for retrieval/tool actions, rewrite laws for optimizer passes, and trace semantics for observability. SL Algebra is therefore not a new grammar engine or retriever, but the representation in which heterogeneous controls can be reasoned about jointly (Appendix A.7 gives a qualitative comparison).

3 SL Algebra

We consider online autoregressive decoding for a decoder-only language model with vocabulary $\mathcal{V}$. At step t, the model has produced prefix $x_{1:t} \in \mathcal{V}^t$, maintains cache state KV_t, and may depend on auxiliary runtime state such as retrieved documents, tool outputs, grammar automata, and telemetry. We collect all non-model auxiliary information into ζ_t and define the decoding state as

$$s_t := (x_{1:t}, \mathrm{KV}_t, \zeta_t) \in \mathcal{S}.$$

Let $f_\theta : \mathcal{S} \to (\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|}$ denote the base forward pass that returns the next-token logits.

State and logit operators. A *state operator* is a map $o : \mathcal{S} \to \mathcal{S}$ that transforms the decoding state before token selection. A *logit operator* is a curried map

$$g : \mathcal{S} \to \big((\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|} \to (\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|}\big), \qquad g(s)(\ell) \in (\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|}$$

that transforms next-token logits while leaving the logical decoding state unchanged. Retrieval, tool execution, memory updates, and restart boundaries are naturally state operators; hard masks, finite biases, and retrieval-derived logits processors are naturally logit operators. Here $(\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|} \subseteq \overline{\mathbb{R}}^{|\mathcal{V}|}$ makes exact hard masks with $-\infty$ type-consistent by construction.

This distinction is semantic rather than implementational: retrieval alters the conditioning context, whereas schema masking constrains the admissible next tokens. Conflating them in a uniform runtime callback obscures both guarantees and optimization opportunities.

Effect annotations and plans. To constrain legal rewrites and lowerings, we attach lightweight effect annotations. State operators may be marked KVStable, KVInvalid, IO, or Pure; logit operators may be marked HardMask, SoftBias, StateDep, or StateIndep. A *control plan* is a pair

$$P = (O, G), \qquad O = o_m \circ \cdots \circ o_1, \qquad G = g_n \circ \cdots \circ g_1,$$

with a well-formedness judgment $\vdash P$ wf enforcing, for example, that KVInvalid operators appear only at explicit restart boundaries and that state-dependent logit operators read only materialized information in ζ_t. Composition of logit operators is evaluated at fixed state: $G(s) = g_n(s) \circ \cdots \circ g_1(s)$, so applying the plan to logits is $G(s)(\ell)$.

Derivable effect-typing judgment. To make $\vdash P$ wf mechanically checkable rather than purely descriptive, we use a lightweight derivation system parameterized by context Γ (materialized fields and restart status). We separate effect typing from legality checking:

$$\Gamma \vdash o : \epsilon_s,\ \Gamma \vdash g : \epsilon_\ell \quad \text{and} \quad \Gamma \vdash o\ \mathsf{wf},\ \Gamma \vdash g\ \mathsf{wf}.$$

Here ϵ_s and ϵ_ℓ range over state- and logit-effect labels, respectively; $\mathrm{annot}(\cdot)$ returns declared effect annotations, $\mathrm{reads}(g)$ returns the materialized fields read by g, and $\mathrm{update}(\Gamma, \cdot)$ denotes the context-transition function used by typed execution. Core constraints are annotation-driven and context-sensitive, e.g.

$$\mathsf{KVStable} \in \mathrm{annot}(o) \Rightarrow \Gamma \vdash o : \mathsf{KVStable},$$
$$\mathsf{KVInvalid} \in \mathrm{annot}(o) \wedge \Gamma(\mathsf{restart}) = \mathsf{open} \Rightarrow \Gamma \vdash o : \mathsf{KVInvalid},$$
$$\mathsf{StateDep} \in \mathrm{annot}(g) \wedge \mathrm{reads}(g) \subseteq \Gamma(\mathsf{mat}) \Rightarrow \Gamma \vdash g\ \mathsf{wf}.$$

Full inference rules (O-SEQ/G-SEQ/PLAN-COMPOSE, S-GUARD) and the Γ-update discipline are given in Appendix B.1.

Given a well-formed plan, one controlled decoding step is

$$s_t^+ = O(s_t), \qquad \tilde{\ell}_t = G(s_t^+)\big(f_\theta(s_t^+)\big), \qquad y_t \sim \mathrm{softmax}(\tilde{\ell}_t), \qquad s_{t+1} = \mathsf{step}(s_t^+, y_t).$$

The plan is a *logical* object: it specifies intended control behavior independently of how that behavior is implemented by a backend.

Typed execution invariance. Under the typed small-step semantics (Appendix B.1), side conditions are invariant under one-step execution: context updates are monotone on materialized fields and restart transitions are constrained to explicit KVInvalid boundaries. The formal proposition and proof sketch are given in Appendix B.1 (Proposition B.2).

Non-anticipation. An operator applied at step t is *non-anticipatory* if it is a measurable function of the current state s_t and does not depend on future tokens $x_{t+1:}$. This is the basic causality condition of the algebra. It rules out retrieval triggers, masks, or routing policies that depend on future generations.

Grounding as an explicit join. Let $\mathcal{M}$ denote an external memory space. For source $k \in K$, let

$$R_k : \mathcal{S} \to \mathcal{M}, \qquad J_k : \mathcal{S} \times \mathcal{M} \to \mathcal{S}$$

be a non-anticipatory retriever and join mechanism. The logical join is

$$\bowtie_k(s_t) := J_k\big(s_t, R_k(s_t)\big).$$

The key point is that this logical join does *not* specify its physical realization. The same join can be lowered as: (i) **Join-Early** (retrieve and prepend, typically KVInvalid if done mid-decode), (ii) **memory fusion** via a KV-stable architectural modification such as $h_t \leftarrow h_t + \mathrm{Attn}(q_t, K_M, V_M)$, or (iii) a **logit-additive join** $\ell \mapsto \ell + b(M_t)$, which remains KV-stable but depends on the quality of the compiled bias $b(M_t)$. Distinguishing logical join from lowering is essential, as it allows us to reason about semantics and systems consequences in one representation.

Guarded Join. Conditional retrieval is an explicit plan node,

$$\mathsf{Guard}[\phi](\bowtie_k) := \texttt{if } \phi(s_t) \texttt{ then } \bowtie_k \texttt{ else } \mathsf{Id},$$

where $\phi : \mathcal{S} \to \{0, 1\}$ is non-anticipatory. This makes retrieval policy visible to the optimizer and to the trace. In general, Guarded Join does not preserve stronger notions of equivalence, because whether retrieval occurs can change both state and downstream token probabilities. Representing the policy explicitly makes it measurable, auditable, and optimizable: SL Algebra turns what is often an implicit runtime heuristic into a first-class plan node that can be logged, ablated, and diagnosed when a seemingly sensible policy becomes physically inefficient.

4 FORMAL PROPERTIES

Three equivalence levels. A single equivalence notion is too coarse for controlled decoding. We therefore distinguish three levels. *S0 (support equivalence):* two logit plans G, G' are equivalent if they induce the same next-token support on every reachable pair (s, ℓ):

$$\mathrm{supp}(\mathrm{softmax}(G(s)(\ell))) = \mathrm{supp}(\mathrm{softmax}(G'(s)(\ell))).$$

S1 (stepwise logit equivalence): $G(s)(\ell) = G'(s)(\ell)$ pointwise. *S2 (trace-distribution equivalence):* two full plans induce the same distribution over generated token sequences and, when enabled, the same distribution over recorded traces. The stratification matters because some rewrites preserve only admissibility, while others preserve exact logits or full execution behavior.

Exact hard-mask fragment. We first study the fragment consisting of exact hard masks $m \in \{0, -\infty\}^{|\mathcal{V}|}$ and finite additive biases. In this regime, the following identities hold pointwise:

$$(\ell + m_A) + m_B = \ell + (m_A \wedge m_B), \qquad (\ell + m) + b = (\ell + b) + m, \qquad (\ell + m) + m = \ell + m.$$

Here $m_A \wedge m_B$ is the elementwise minimum, i.e., the mask admitting exactly the tokens admitted by both. Exact masks thus fuse by conjunction, commute with finite biases, and are idempotent. We summarize the exact rewrite laws in Appendix A.7 (Table 13); Appendix B gives the proofs. The immediate operational consequence is a compiler-style normalization result.

Proposition 4.1 (Well-formedness preservation under normalization)**.** *Let $P = (O, G)$ with $\vdash P$ wf. Let $\mathsf{normalize}(P)$ apply only the above exact hard-mask rewrites (mask fusion, bias-mask commutation, and mask idempotence) inside G, without changing O. Then*

$$\vdash \mathsf{normalize}(P)\ \mathsf{wf}.$$

Proof sketch. The rewrites are local to logit operators and do not introduce new state writes or restart transitions. In this normalization fragment, transformed operators are exact masks and finite biases annotated as HardMask/StateIndep, hence $\mathrm{reads}(g) = \emptyset$ before and after fusion/commutation/idempotence. Therefore transformed logit operators remain well-formed under the state-independent branch of the typing system, and because O is unchanged, the composed plan remains well-formed after normalization. The full rule-level derivation is given in Appendix B.1.

Proposition 4.2 (Canonical form for the exact hard-mask fragment)**.** *Let G be any finite composition of exact hard masks and finite state-independent biases. Then there exists an S1-equivalent plan*

$$G_{\mathrm{nf}}(\ell) = (\ell + b^\star) + m^\star,$$

where $m^\star$ is the conjunction of all hard masks in G and $b^\star$ is the sum of all finite biases in G.

Tokenization-lift soundness. Formal interface constraints are often specified over bytes or characters, while LLMs decode over tokens. Let

$$\mathrm{bytes} : \mathcal{V} \to \{0, \ldots, 255\}^*$$

be the token-to-byte emission map, and let $A_\Gamma = (Q, q_0, F, \delta_\Gamma)$ be a DFA over bytes recognizing target language L_Γ. We lift the DFA to tokens by defining

$$\delta_{\mathcal{V}}(q, v) = \begin{cases} \delta_\Gamma^*(q, \mathrm{bytes}(v)) & \text{if all intermediate transitions are defined,} \\ \bot & \text{otherwise.} \end{cases}$$

If q_t is the DFA state after consuming the bytes emitted by the current prefix, the corresponding hard mask is

$$m_{q_t}(v) = \begin{cases} 0 & \text{if } \delta_{\mathcal{V}}(q_t, v) \neq \bot \text{ and } (v \neq \mathrm{EOS} \text{ or } q_t \in F), \\ -\infty & \text{otherwise.} \end{cases}$$

Theorem 4.3 (Tokenization-lift soundness)**.** *Under exact masking, suppose decoding samples each next token from*

$$\mathrm{softmax}\big(f_\theta(s_t) + m_{q_t}\big)$$

and assume (i) the effective valid token set is non-empty at every nonterminal step, and (ii) decoding terminates almost surely (i.e., emits EOS *in finite time with probability* 1*). Then the emitted byte string belongs to L_Γ with probability* 1*.*

Logical plan
$\Sigma_{\text{JSON}} \circ \mathsf{Guard}[\phi](\bowtie_{\text{KB}})$

Lowering A (Join-Early)
retrieve $\rightarrow$ serialize docs $\rightarrow$ restart boundary $\rightarrow$ generate under Σ

Lowering B (logit-additive)
retrieve $\rightarrow$ build sparse bias $b(M_t) \rightarrow$ generate under $\ell + b(M_t) + m_t$

Trace fields
plan hash; gate outcome; retrieved doc_ids; mask statistics; wall-clock timings

Figure 2: **One logical plan, multiple physical realizations.** The IR fixes control semantics, while the backend determines join and mask implementation. The plan also defines an explicit trace schema.

The theorem isolates the exact guarantee of hard constrained decoding: if the byte-level language is compiled correctly and the runtime enforces the mask exactly, structural validity is a property of the decoding procedure itself rather than of the model's learned formatting behavior. This is precisely the kind of guarantee that tends to disappear when structured generation is described only informally as a prompt or parser wrapper. Appendix B also gives a finite-penalty leakage bound for the practical regime where $-\infty$ is approximated by a large negative constant, which connects the exact semantics to realistic fused-kernel implementations.

5 Reference Runtime and Evaluation Design

Reference runtime. The algebra is logical; to evaluate it we implement a simple reference runtime that checks plans, lowers operators to standard backends, executes them, and emits structured traces. The runtime is built on Hugging Face Transformers (Wolf et al., 2020). Hard constraints (Σ) are realized as exact token-admissibility masking during decoding, using the JSON-schema and regex parsers of LM Format Enforcer (Gat) to compute the admissible token set at each step. Logical joins ($\bowtie$) are lowered either as **Join-Early** (retrieve, serialize, prepend, restart) or as a KV-stable **logit-additive join** implemented as a sparse logits processor. Guarded Join plans are lowered as explicit conditional nodes whose predicates read only the materialized prefix. For each request, the runtime records plan hashes, operator firings, join outcomes, retrieved document identifiers, validity diagnostics, and timing counters. Figure 2 shows the intended interpretation: the plan is the stable logical object, while lowerings and trace schemas are backend-facing artifacts derived from it.

Tasks. Our core protocol uses three controlled testbeds, each chosen to isolate a different claim. **AGNews JSON** tests whether formal admissibility is a bottleneck when the output must satisfy a fixed JSON schema (Zhang et al., 2015). **SST-2 label generation** tests whether exact language constraints matter even for very small output languages (Socher et al., 2013; Wang et al., 2018). **SQuAD-RAG-JSON** tests grounded answering when the model must emit a JSON object containing an answer field and citation indices (Rajpurkar et al., 2016). Retrieval uses a dense FAISS index (Johnson et al., 2021) over Sentence-BERT embeddings (Reimers & Gurevych, 2019). We additionally report **Hotpot-RAG-JSON** (multi-hop grounded QA) as a broader stress test for retrieval-dependent reasoning (Yang et al., 2018).

Plans, models, and metrics. We compare **Prompt-only**, **Σ only**, **$\bowtie + \Sigma$** (always-join), and **Guarded $\bowtie + \Sigma$**. We evaluate seven instruction-tuned models: Qwen2.5-0.5B, -1.5B, -7B, -14B (Qwen et al., 2024) and Qwen3-1.7B, -4B, -8B (Yang et al., 2025). We report **validity** (the *complete* generated string must belong to the interface language, so a well-formed answer followed by extra text is invalid, as it would be for a downstream parser), **strict accuracy** (invalid outputs count as wrong), **loose accuracy** (semantic task accuracy ignoring interface validity), and, for grounded QA, **exact match (EM)**, **supported rate**, and **retrieval recall@k**. For grounded QA, **Supported%** is the percentage of all evaluated examples whose output is valid JSON with a non-empty answer that appears, after normalization, in a document the model cites by its 1-indexed rank; empty or out-of-range citations count as unsupported. It is a conservative lexical metric (no entailment model or LLM judge; Appendix C), and it is 0 by construction for plans without $\bowtie$, which retrieve no documents to cite. Systems measurements include wall-clock latency, ms/token, token counts, and trace counters.

Table 1: **Main results across structured generation and grounded QA tasks** (means over three seeds; 95% bootstrap intervals in Appendix A). Structured-task entries are *Prompt-only / Σ*; grounded-QA entries are Σ *only / ⋈ + Σ*. On AGNews and SST-2, Σ turns loose task capability into strict correctness; on grounded QA, adding ⋈ substantially improves answer quality over constraints alone. Supported is 0 by construction without ⋈ (no documents to cite).

Task	Plan / Metric	Qwen2.5				Qwen3		
		0.5B	1.5B	7B	14B	1.7B	4B	8B
AGNews JSON								
AGNews	Valid (Prompt / Σ)	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0
AGNews	Strict (Prompt / Σ)	0.0 / 68.8	0.0 / 78.5	0.0 / 78.3	0.0 / 84.7	0.0 / 38.3	0.0 / 80.5	0.0 / 82.3
AGNews	Loose (Prompt / Σ)	46.0 / 68.8	75.2 / 78.5	77.7 / 78.3	84.3 / 84.7	43.0 / 38.3	66.3 / 80.5	82.3 / 82.3
SST-2 label generation								
SST-2	Valid (Prompt / Σ)	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0
SST-2	Strict (Prompt / Σ)	0.0 / 57.7	0.0 / 92.3	0.0 / 92.7	0.0 / 90.0	0.0 / 53.0	0.0 / 93.2	0.0 / 92.0
SST-2	Loose (Prompt / Σ)	57.7 / 57.7	81.2 / 92.3	91.7 / 92.7	61.3 / 90.0	53.0 / 53.0	40.5 / 93.2	92.0 / 92.0
SQuAD-RAG-JSON								
SQuAD-RAG	Valid (Σ / Join)	98.2 / 98.2	87.3 / 95.7	82.0 / 94.7	82.3 / 97.3	97.3 / 99.0	99.2 / 98.8	96.7 / 98.7
SQuAD-RAG	EM (Σ / Join)	2.8 / 27.3	6.3 / 36.8	7.3 / 32.7	17.0 / 37.3	7.7 / 46.0	3.8 / 47.0	9.0 / 40.3
SQuAD-RAG	Supported (Σ / Join)	0.0 / 36.8	0.0 / 46.5	0.0 / 6.3	0.0 / 58.0	0.0 / 55.5	0.0 / 10.0	0.0 / 49.0
Hotpot-RAG-JSON								
Hotpot-RAG	Valid (Σ / Join)	96.8 / 99.8	77.3 / 99.8	92.0 / 96.7	92.7 / 95.3	98.0 / 100.0	98.5 / 95.0	94.7 / 96.0
Hotpot-RAG	EM (Σ / Join)	9.0 / 18.8	8.0 / 27.3	14.7 / 28.0	20.0 / 32.3	11.0 / 27.7	10.8 / 29.0	12.3 / 35.0
Hotpot-RAG	Supported (Σ / Join)	0.0 / 14.3	0.0 / 47.8	0.0 / 14.0	0.0 / 31.0	0.0 / 24.5	0.0 / 8.8	0.0 / 48.0

Three diagnostics are especially important for the IR claim. First, a *join-physicalization ablation* compares Join-Early with a KV-stable logit-additive join under the same logical plan, testing whether the logical/physical separation is empirically necessary rather than merely notational. Second, a *mask-fusion microbenchmark* tests whether the exact hard-mask laws expose a real optimizer-visible runtime win. Third, the guarded-join analysis asks whether making a retrieval policy explicit improves physical execution or reveals where the runtime spends time. The design principle is causal isolation rather than benchmark breadth.

Protocol. All experiments use greedy (deterministic) decoding and three seeds per model/task/plan; each seed selects a different deterministic subset of n evaluation examples, so seed-to-seed variation reflects example sampling. All plans share the same plain-text prompt and differ only in their control operators. Experiments are run on a workstation with four NVIDIA GeForce RTX 4080 GPUs (16 GB VRAM each). For memory efficiency, we use 4-bit NF4 quantization (Dettmers et al., 2023) for Qwen2.5-1.5B, Qwen3-4B, Qwen2.5-7B, Qwen3-8B, and Qwen2.5-14B. For Guarded Join, the gate is evaluated once at the pre-generation boundary (before any token is generated), using next-token entropy of the initial distribution. With $z_0 = f_\theta(s_0^+)$ the logits after state ops and A_0 the set allowed by Σ ($A_0 = \mathcal{V}$ without Σ), the gate uses the constrained entropy $H_0 = -\sum_{v \in A_0} p_0(v) \log p_0(v)$ with $p_0(v) \propto \exp z_{0,v}$ on A_0. Retrieval is triggered iff $H_0 \geq \tau$; the main guarded setting uses $\tau = 0.92$, the join-physicalization ablation uses $\tau = 0.75$, and Appendix A.5 sweeps $\tau \in [0.70, 0.85]$. A prompt-level necessity heuristic can force the join regardless of H_0. Constrained decoding uses exact masking rather than a finite $-C$ penalty.

6 RESULTS

RQ1: explicit constraints turn validity into strict correctness. In Table 1, prompt-only runs retain loose task capability but zero strict correctness, while Σ raises validity to 100% on AGNews and SST-2 at every scale. Stored traces explain why: every prompt-only output runs to the generation cap, and on AGNews and SST-2 most of them (79–100% for the three Qwen2.5 models with released traces) begin with a well-formed answer and then keep generating (Appendix A.6, Figure 6). This is precisely the failure that Σ rules out: under the lifted mask, EOS is admissible only in accepting states, and once the object closes only whitespace or EOS remain admissible (Theorem 4.3). Even against a generous post-hoc baseline that keeps only the first well-formed object, Σ matches or exceeds strict

Table 2: **Logical vs. physical join.** One logical $\bowtie$ lowered as *Join-Early* or as a KV-stable *logit-additive join* (SQuAD-RAG-JSON, k=3).

Model	Join-Early			Logit-additive		
	Valid	EM	p95 ms	Valid	EM	p95 ms
Qwen2.5-0.5B	98.2	27.3	4757	82.2	3.7	6582
Qwen2.5-1.5B	95.7	36.8	7627	48.5	5.5	8645
Qwen2.5-7B	94.7	32.7	7541	63.0	7.7	8283
Qwen2.5-14B	97.3	37.3	14494	65.7	15.0	17069
Qwen3-4B	98.8	47.0	11265	97.5	4.2	8426

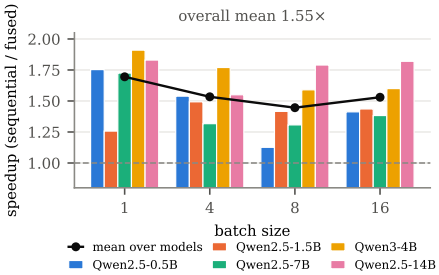

Figure 3: **Mask fusion.** Speedup of one fused conjunction mask over two sequential exact masks.

accuracy in all nine model–task pairs we can re-score (Table 10), without any task-specific repair code.

RQ2: explicit joins improve grounded QA. For grounded QA, Σ alone mainly enforces format, while adding $\bowtie$ drives quality. In Table 1, EM rises from low single digits to 27%–47% on SQuAD and from 9%–20% to 19%–35% on Hotpot, with aligned Supported-rate gains. The largest SQuAD gain is Qwen3-4B ($3.8\% \rightarrow 47.0\%$); the largest Hotpot EM is Qwen3-8B (35.0%). Σ-only validity below 100% is due to generation-length truncation before JSON closure, i.e., a violation of the termination assumption of Theorem 4.3 rather than of its conclusion. The low Supported% of Qwen2.5-7B on SQuAD reflects a citation-format mismatch, not ungrounded answers: in 213 of 300 outputs it cites the corpus identifier shown in the document header instead of the rank (Appendix A.6). Retrieval recall remains bounded (about 63% on SQuAD, 85%–86% on Hotpot), so retriever quality is an upstream ceiling.

Statistical robustness and cross-dataset consistency. Using three-seed bootstrap summaries, conservative interval subtraction shows positive Hotpot lower bounds for ΔEM in six of seven models and for ΔSupported in all seven (Appendix Table 7). Every model–task pair also lies above $\Delta\text{EM} = 0$ with model-dependent latency costs (Appendix Figure 5).

Logical joins and physical lowerings are empirically non-interchangeable. Table 2 shows a large lowering gap under the same logical plan: Join-Early improves EM by 22.3–42.8 points over the current logit-additive implementation (largest for Qwen3-4B) and improves validity in all listed models. Because both rows share one logical plan, this gap is invisible to any representation that does not separate semantics from physical realization.

RQ3: the IR exposes both exact optimizations and physical bottlenecks. First, exact mask fusion is consistently beneficial (Figure 3): the masking operator is $1.55\times$ faster on average, per-model averages range from $1.40\times$ to $1.75\times$, and every configuration is above $1.1\times$. The absolute saving per step is small (e.g., 25.3 to 14.4 μs for Qwen2.5-0.5B at batch size 1), but it is exact and licensed by an algebraic law rather than by case-by-case testing.

Second, Guarded Join reduces retrieval frequency but, in this prototype, lowers EM and increases latency; our hypothesis that an entropy guard improves the quality–latency trade-off is therefore *not supported*. Table 9 shows weaker quality at higher thresholds with limited latency relief, and Table 4 shows guarded variants slower than always-join under matched lowerings. Because every operator firing is traced, the slowdown is attributable (Figure 4): each Σ request spends 3.1–3.4 s compiling the constraint, the guarded plan compiles it a second time to compute the constrained gate entropy, and retrieval I/O is negligible.

The same telemetry shows that decoding under Σ takes 20–50% less time than prompt-only decoding, because exact masking terminates generation at an accepting state. The overhead is thus a property of one lowering step, not of the plan: the schema is fixed per task, so the compiled constraint can be cached and shared between gate and decoder.

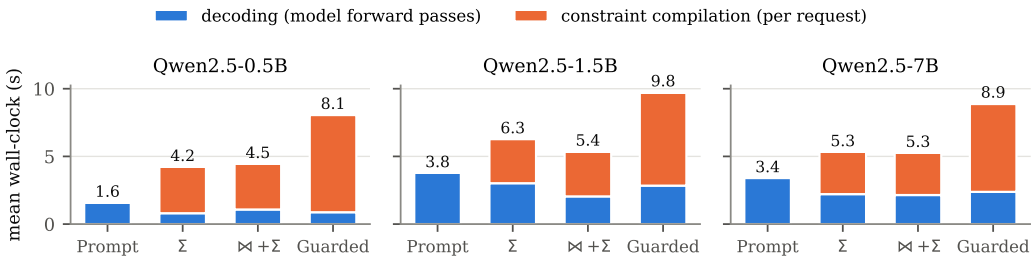

Figure 4: **Trace-derived latency decomposition on SQuAD-RAG-JSON** (mean seconds per request, three seeds; Qwen2.5 models with released traces). The overhead of Σ and of Guarded Join is per-request constraint compilation, which the guarded plan performs twice; retrieval ($\approx$5 ms) and the gate forward pass ($<$60 ms) are too small to be visible.

7 Discussion, Scope, and Limitations

SL Algebra is a *control abstraction*, not a finished serving stack. It supports exact rewrite reasoning, clarifies the scope of constrained-decoding guarantees, and separates a weak policy from a weak lowering: our negative Guarded Join result is diagnosable precisely because the plan and its trace are explicit.

The evidence is diagnostic; we do not claim an optimal retriever, guard policy, or serving kernel. Our typing results establish well-formedness preservation for the rewrites we use; we do not claim a full type-soundness theorem for arbitrary plans. Exact guarantees assume exact masking; practical runtimes may use finite penalties or mixed precision (Appendix B.4). The prompt-only baseline uses the same plain-text prompts as all other plans, without chat templates or stop sequences, which isolates the effect of control operators but understates what task-specific prompt engineering and output repair can recover (Table 10). The prototype recompiles constraints per request and runs host-side masking, so absolute latencies should be read as protocol-dependent. Empirical scope is targeted (structured generation and grounded QA over one model family), not exhaustive.

Finally, traces are *procedural* records of control-flow events, not evidence of model understanding or truthfulness, and the prototype is not a safety-certified decision system.

8 Conclusion

We introduced SL Algebra, a control IR for structured, grounded, and auditable LLM inference. Separating state operators from logit operators turns constraints, joins, and runtime policies into explicit plan objects with semantics, side conditions, and traces. Empirically, the abstraction matters: hard constraints raise validity from 0% to 100% on AGNews and SST-2 across all seven models; explicit joins raise SQuAD EM by up to 43 points and Hotpot EM by up to 23 points; mask fusion yields a mean $1.55\times$ operator speedup; and Join-Early exceeds the logit-additive lowering by up to 42.8 EM points under matched logical plans. Explicit plans also make bottlenecks legible, localizing the cost of Guarded Join to repeated constraint compilation. A control IR does not replace programming systems or production runtimes; it provides one language in which guarantees, lowerings, and bottlenecks can be reasoned about.

Ethics Statement

All authors have read and adhere to the ICLR Code of Ethics. This work involves no human subjects, crowdsourcing, or newly collected data. All experiments use public benchmarks (AG News, SST-2/GLUE, SQuAD, HotpotQA) under their original licenses and publicly released open-weight Qwen2.5 and Qwen3 models; we release no new dataset and process no personal or sensitive information. SL Algebra is a methodological contribution for making inference-time control explicit and auditable, and we see its main societal effect as positive: explicit constraints, grounding, and traces

make LLM behavior easier to verify. We note three risks. First, traces record which controls fired and which evidence was retrieved; they are procedural rather than epistemic and should not be presented as explanations of model reasoning or as guarantees of factual correctness (Section 7). Second, grounding improves but does not guarantee answer correctness, and our Supported% metric is lexical, so it can both under- and over-state faithful citation; the prototype must not be deployed in high-stakes settings without domain-specific validation. Third, hard constraints are value-neutral: they can enforce any output language, including one that encodes a harmful policy, and they inherit whatever biases are present in the underlying model and retrieval corpus; we evaluate neither fairness nor bias. Experiments ran on a single workstation with four consumer GPUs, keeping the computational footprint modest.

Reproducibility Statement

We provide anonymized source code at the link in the abstract, including the SL Algebra IR (operators, effect annotations, plan type-checker, and optimizer rewrites), the reference runtime, the evaluation harness, and all scripts used to produce the tables and figures (multi-seed runs, bootstrap aggregation, sweeps, join-mode and constraint ablations, the mask-fusion microbenchmark, and the trace-derived diagnostics of Appendix A.6). Section 5 and Appendix C specify models, quantization, datasets and splits, sample sizes, seeds and subset selection, decoding caps, retriever configuration, gate thresholds, metric definitions, and hardware. Per-example JSONL traces with telemetry and environment snapshots are released with the code, so every reported number and the post-hoc analyses can be recomputed without re-running the models. For the theoretical results, all assumptions are stated with each result in Section 4, and complete proofs, the full effect-typing rules, and the finite-penalty leakage bound are given in Appendix B.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. *Proc. ACM Program. Lang.*, 7(PLDI):1336–1360, 2023. doi: 10.1145/3591300. URL https://doi.org/10.1145/3591300.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: Fast, non-invasive constrained generation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3658–3673. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/beurer-kellner24a.html.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. *CoRR*, abs/2411.15100, 2024.

dottxt-ai. Outlines: Structured outputs. GitHub repository, 2026. URL https://github.com/dottxt-ai/outlines. Accessed 2026-02-24.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, et al. RARR: Researching and revising what language models say, using language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16477–16508, 2023a.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6465–6488, 2023b.

Noam Gat. LM Format Enforcer: Enforce the output format (json schema, regex, etc.) of a language model. GitHub repository. URL https://github.com/noamgat/lm-format-enforcer. Accessed 2026-02-24.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674. URL https://aclanthology.org/2023.emnlp-main.674/.

Saibo Geng, Hudson Cooper, Michal Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *CoRR*, abs/2501.10868, 2025. doi: 10.48550/ARXIV.2501.10868. URL https://doi.org/10.48550/arXiv.2501.10868.

guidance-ai. Guidance: A guidance language for controlling large language models. GitHub repository, 2026. URL https://github.com/guidance-ai/guidance. Accessed 2026-02-24.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1535–1546, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1141. URL https://aclanthology.org/P17-1141/.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th conference of the european chapter of the association for computational linguistics: main volume*, pp. 874–880, 2021.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7969–7992, 2023.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. *CoRR*, abs/2407.08103, 2024. doi: 10.48550/ARXIV.2407.08103. URL https://doi.org/10.48550/arXiv.2407.08103. Also circulated as a COLM 2024 camera-ready on arXiv.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP '23: Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 761–774. ACM, 2023. doi: 10.1145/3600006.3613135. URL https://doi.org/10.1145/3600006.3613135.

LangChain. LangSmith observability. Documentation, 2026. URL https://docs.langchain.com/langsmith/observability. Accessed 2026-03-04.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Agada Joseph Oche, Ademola Glory Folashade, Tirthankar Ghosal, and Arpan Biswas. A systematic review of key retrieval-augmented generation (RAG) systems: Progress, gaps, and future directions. *arXiv preprint arXiv:2507.18910*, 2025.

OpenAI. Tracing – OpenAI agents SDK. Documentation, 2026. URL https://openai.github.io/openai-agents-python/tracing/. Accessed 2026-03-04.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8371–8384, 2024.

Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V. Vasilakos. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*, 2025.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, 2013. URL https://aclanthology.org/D13-1170/.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. Syncode: LLM generation with grammar augmentation. *Transactions on Machine Learning Research*, 2025. Originally circulated as arXiv:2403.01632.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi (eds.), *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446/.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020. doi: 10.18653/v1/2020.emnlp-demos.6.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Fuda Ye, Shuangyin Li, Yongqi Zhang, and Lei Chen. R^2AG: Incorporating retrieval information into retrieval augmented generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11584–11596, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.678. URL https://aclanthology.org/2024.findings-emnlp.678/.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

USE OF LARGE LANGUAGE MODELS

Large language models were used to assist with polishing the writing of this manuscript. All technical content, experiments, and conclusions are the authors' own, and the authors take full responsibility for the paper.

A EXTENDED EXPERIMENTAL RESULTS

This appendix provides the empirical detail omitted from the main paper for space: scaling results, join-physicalization ablations, grounded-QA and structured-task metrics across model sizes, structured-task cost profiles, guarded-join threshold sweeps, trace-derived diagnostics, and implementation notes relevant to reproducibility.

A.1 SCALING AND JOIN PHYSICALIZATION

This subsection complements the main-text results with full model-scale detail. Table 3 reports the scaling trend under a fixed evaluation protocol, while Table 4 isolates the effect of physical join realization (Join-Early vs. logit-additive join) under matched logical plans.

Table 3: **Scaling study.** We report strict quality under Σ for structured tasks and strict EM under $\bowtie + \Sigma$ for grounded QA at $k = 3$. n is the number of evaluation examples per seed.

Task	**Qwen2.5**				**Qwen3**		
	0.5B	**1.5B**	**7B**	**14B**	**1.7B**	**4B**	**8B**
n	200	200	100	100	200	200	100
AGNews (Σ), reported as Valid/Strict							
AGNews	100.0/68.8	100.0/78.5	100.0/78.3	100.0/84.7	100.0/38.3	100.0/80.5	100.0/82.3
SST-2 (Σ), reported as Valid/Strict							
SST-2	100.0/57.7	100.0/92.3	100.0/92.7	100.0/90.0	100.0/53.0	100.0/93.2	100.0/92.0
SQuAD-RAG ($\bowtie + \Sigma$), reported as Valid/EM							
SQuAD-RAG	98.2/27.3	95.7/36.8	94.7/32.7	97.3/37.3	99.0/46.0	98.8/47.0	98.7/40.3
Hotpot-RAG ($\bowtie + \Sigma$), reported as Valid/EM							
Hotpot-RAG	99.8/18.8	99.8/27.3	96.7/28.0	95.3/32.3	100.0/27.7	95.0/29.0	96.0/35.0

Consistent with the claims in Section 6, Table 3 shows that strict structured performance generally improves with scale under Σ, whereas grounded QA remains sensitive to retrieval quality and join realization across both SQuAD and Hotpot. We therefore report the physicalization ablation separately in Table 4, which reveals that quality and latency are not determined by logical plan structure alone.

A.2 STRUCTURED TASKS ACROSS MODEL SCALE

Table 5 expands the structured-task view from the main text by listing per-model means and 95% bootstrap intervals for both prompt-only and Σ conditions. This table is intended to support cross-model comparisons while preserving uncertainty estimates.

Across models, Table 5 reinforces the same qualitative pattern: prompt-only generation can retain non-trivial loose accuracy while failing strict validity, and Σ systematically converts this into strict end-to-end correctness by enforcing admissible outputs.

A.3 GROUNDED QA ACROSS MODEL SCALE

Table 6 provides the full grounded-QA breakdown for both SQuAD-RAG-JSON and Hotpot-RAG-JSON, including validity, EM, supported rate, retrieval recall@k, and latency. We include these fields jointly to separate answer quality from retrieval coverage and generation cost.

Table 4: **Join-physicalization ablation on SQuAD.** Guarded rows use threshold 0.75. Join-Early substantially outperforms the current logit-additive join prototype, highlighting the gap between logical join expressivity and current KV-stable realizations.

Model	Join mode / policy	Valid%	EM%	Wall (ms)	p95 (ms)	Calls
Qwen2.5-0.5B	Join-Early / always	98.2	27.3	4311	4757	200.0
	Join-Early / guarded	98.3	22.7	7921	8534	147.0
	Logit-additive / always	82.2	3.7	4607	6582	200.0
	Logit-additive / guarded	86.2	3.5	8150	10135	147.0
Qwen2.5-1.5B	Join-Early / always	95.7	36.8	5155	7627	200.0
	Join-Early / guarded	88.7	11.0	9509	11830	27.0
	Logit-additive / always	48.5	5.5	7678	8645	200.0
	Logit-additive / guarded	82.8	6.2	9721	11774	27.0
Qwen2.5-7B	Join-Early / always	94.7	32.7	5278	7541	100.0
	Join-Early / guarded	82.3	10.7	8938	11639	17.3
	Logit-additive / always	63.0	7.7	6086	8283	100.0
	Logit-additive / guarded	77.7	7.3	9033	11623	17.3
Qwen3-4B	Join-Early / always	98.8	47.0	8582	11265	200.0
	Join-Early / guarded	99.2	9.2	12643	15124	27.0
	Logit-additive / always	97.5	4.2	7111	8426	200.0
	Logit-additive / guarded	99.0	4.0	12641	14771	27.0
Qwen2.5-14B	Join-Early / always	97.3	37.3	9820	14494	100.0
	Join-Early / guarded	83.7	17.7	16087	23066	15.3
	Logit-additive / always	65.7	15.0	11412	17069	100.0
	Logit-additive / guarded	79.0	16.3	15557	21508	15.3

The detailed results in Table 6 are consistent with the main-text interpretation: Σ alone largely improves formatting validity, while explicit joins drive the main EM gains. At the same time, recall@k remains bounded, indicating an upstream retrieval ceiling that limits further grounded-QA improvement.

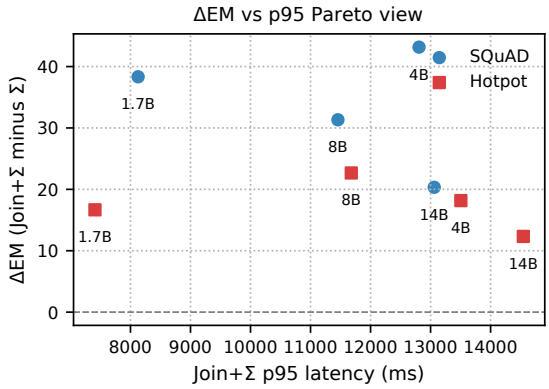

Figure 5: **ΔEM vs. p95 latency for grounded QA.** Each point compares $\bowtie + \Sigma$ with Σ for one model–task pair (x-axis: $\bowtie + \Sigma$ p95 latency). All points lie above $\Delta\text{EM} = 0$.

A.4 Detailed Cost Profiles

Table 8 reports fine-grained runtime diagnostics for structured tasks, including p50/p95 latency, ms/token, and token counts. We include all plan variants to make the latency decomposition auditable rather than relying on a single aggregate number. For AGNews/SST-2 specifically, join-enabled

Table 5: **Structured tasks across model scale.** Values are means with 95% bootstrap confidence intervals over three seeds.

Task	Model	Plan	Valid%	Strict%	Loose%
AGNews	Qwen2.5-0.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	46.0 [44.0–48.0]
		Σ	100.0 [100.0–100.0]	68.8 [67.5–70.0]	68.8 [67.5–70.0]
	Qwen2.5-1.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	75.2 [73.5–76.5]
		Σ	100.0 [100.0–100.0]	78.5 [77.0–80.0]	78.5 [77.0–80.0]
	Qwen2.5-7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	77.7 [76.0–80.0]
		Σ	100.0 [100.0–100.0]	78.3 [77.0–81.0]	78.3 [77.0–81.0]
	Qwen3-1.7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	43.0 [40.0–45.0]
		Σ	100.0 [100.0–100.0]	38.3 [36.0–40.0]	38.3 [36.0–40.0]
	Qwen3-4B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	66.3 [65.5–67.5]
		Σ	100.0 [100.0–100.0]	80.5 [78.5–82.5]	80.5 [78.5–82.5]
	Qwen3-8B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	82.3 [81.0–84.0]
		Σ	100.0 [100.0–100.0]	82.3 [81.0–84.0]	82.3 [81.0–84.0]
	Qwen2.5-14B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	84.3 [82.0–86.0]
		Σ	100.0 [100.0–100.0]	84.7 [83.0–86.0]	84.7 [83.0–86.0]
SST-2	Qwen2.5-0.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	57.7 [53.0–62.5]
		Σ	100.0 [100.0–100.0]	57.7 [53.0–63.5]	57.7 [53.0–63.5]
	Qwen2.5-1.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	81.2 [79.0–84.5]
		Σ	100.0 [100.0–100.0]	92.3 [91.0–94.0]	92.3 [91.0–94.0]
	Qwen2.5-7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	91.7 [89.0–94.0]
		Σ	100.0 [100.0–100.0]	92.7 [89.0–97.0]	92.7 [89.0–97.0]
	Qwen3-1.7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	53.0 [49.0–59.0]
		Σ	100.0 [100.0–100.0]	53.0 [49.0–59.0]	53.0 [49.0–59.0]
	Qwen3-4B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	40.5 [39.5–41.5]
		Σ	100.0 [100.0–100.0]	93.2 [91.5–95.5]	93.2 [91.5–95.5]
	Qwen3-8B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	92.0 [89.0–97.0]
		Σ	100.0 [100.0–100.0]	92.0 [89.0–97.0]	92.0 [89.0–97.0]
	Qwen2.5-14B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	61.3 [59.0–65.0]
		Σ	100.0 [100.0–100.0]	90.0 [86.0–95.0]	90.0 [86.0–95.0]

rows do not inject retrieved context (no retrieval index is configured for these tasks), and therefore primarily expose control/runtime overhead rather than retrieval-content effects.

As shown in Table 8, constrained plans increase per-request latency relative to prompt-only baselines. The trace telemetry in Appendix A.6 attributes almost all of this increase to per-request constraint compilation (about 3.1–3.4 s, independent of model size), whereas decoding itself is faster under Σ because generation stops at an accepting state (compare the Gen tok column). Because the schema is fixed per task, this cost is amortizable by caching; we report it unamortized to keep the measurement protocol identical across plans.

A.5 Guarded-Join Sensitivity

To characterize the policy frontier for Guarded Join, Table 9 sweeps the entropy threshold while holding $k = 3$ fixed. This isolates how the gating threshold trades retrieval frequency against answer quality and latency.

The sweep in Table 9 indicates a consistent trade-off: higher thresholds reduce join frequency and usually lower EM, while wall-clock latency does not improve proportionally in the current

Table 6: **Grounded QA across datasets (SQuAD-RAG-JSON and Hotpot-RAG-JSON).** Values are means with 95% bootstrap confidence intervals over three seeds when available. Recall@k is shown when available. Guarded-Join policy comparisons are reported separately in Table 4 and Table 9.

Dataset	Model	Plan	Valid%	EM%	Supp.%	Rec@k%	p95 ms
SQuAD	Qwen2.5-0.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	1755 [1575–1849]
		Σ	98.2 [97.5–99.0]	2.8 [2.5–3.5]	0.0 [0.0–0.0]	–	4646 [4346–4868]
		Join+Σ	98.2 [98.0–98.5]	27.3 [22.5–30.5]	36.8 [35.5–38.0]	63.0 [61.5–64.5]	4992 [4687–5166]
	Qwen2.5-1.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	4217 [3946–4503]
		Σ	87.3 [86.0–89.5]	6.3 [6.0–6.5]	0.0 [0.0–0.0]	–	8559 [8047–9078]
		Join+Σ	95.7 [95.5–96.0]	36.8 [34.0–38.5]	46.5 [45.5–47.5]	63.0 [61.5–64.5]	8012 [7675–8381]
	Qwen2.5-7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	3618 [3600–3646]
		Σ	82.0 [81.0–83.0]	7.3 [6.0–9.0]	0.0 [0.0–0.0]	–	8050 [7970–8120]
		Join+Σ	94.7 [94.0–95.0]	32.7 [26.0–36.0]	6.3 [5.0–8.0]	61.0 [59.0–63.0]	7711 [7526–7817]
	Qwen3-1.7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	3137 [2837–3704]
		Σ	97.3 [97.0–97.5]	7.7 [6.5–8.5]	0.0 [0.0–0.0]	–	7771 [7063–8433]
		Join+Σ	99.0 [98.0–99.5]	46.0 [43.5–47.5]	55.5 [52.5–58.0]	63.0 [61.5–64.5]	8127 [7550–8640]
	Qwen3-4B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	9262 [9026–9595]
		Σ	99.2 [98.5–100.0]	3.8 [3.0–5.5]	0.0 [0.0–0.0]	–	9430 [7354–10923]
		Join+Σ	98.8 [97.5–99.5]	47.0 [43.5–51.0]	10.0 [9.5–10.5]	63.0 [61.5–64.5]	12808 [9801–14901]
	Qwen3-8B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	7766 [7002–8352]
		Σ	96.7 [96.0–97.0]	9.0 [8.0–10.0]	0.0 [0.0–0.0]	–	12249 [11170–13864]
		Join+Σ	98.7 [98.0–99.0]	40.3 [36.0–43.0]	49.0 [43.0–56.0]	61.0 [59.0–63.0]	11456 [9655–14432]
	Qwen2.5-14B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	9231 [8895–9462]
		Σ	82.3 [79.0–85.0]	17.0 [16.0–18.0]	0.0 [0.0–0.0]	–	18259 [17444–19836]
		Join+Σ	97.3 [97.0–98.0]	37.3 [34.0–39.0]	58.0 [56.0–61.0]	61.0 [59.0–63.0]	13061 [12128–13711]
Hotpot	Qwen2.5-0.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	3032 [2889–3229]
		Σ	96.8 [96.5–97.5]	9.0 [7.5–10.0]	0.0 [0.0–0.0]	–	9414 [8912–10069]
		Join+Σ	99.8 [99.5–100.0]	18.8 [18.0–19.5]	14.3 [13.5–15.5]	86.3 [84.0–89.0]	7803 [7552–8217]
	Qwen2.5-1.5B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	6210 [5876–6513]
		Σ	77.3 [77.0–78.0]	8.0 [7.5–9.0]	0.0 [0.0–0.0]	–	11140 [10407–11768]
		Join+Σ	99.8 [99.5–100.0]	27.3 [24.5–32.5]	47.8 [42.5–52.0]	86.3 [84.0–89.0]	8064 [7693–8267]
	Qwen3-1.7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	4457 [3919–5476]
		Σ	98.0 [97.0–98.5]	11.0 [9.0–13.0]	0.0 [0.0–0.0]	–	7588 [7296–7902]
		Join+Σ	100.0 [100.0–100.0]	27.7 [23.0–31.0]	24.5 [23.5–26.5]	86.3 [84.0–89.0]	7408 [7262–7511]
	Qwen3-4B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	8315 [8247–8423]
		Σ	98.5 [97.5–99.0]	10.8 [10.0–12.0]	0.0 [0.0–0.0]	–	7726 [7597–7859]
		Join+Σ	95.0 [93.0–97.0]	29.0 [25.5–33.5]	8.8 [7.0–11.0]	86.3 [84.0–89.0]	13505 [9469–15996]
	Qwen2.5-7B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	6678 [6467–7067]
		Σ	92.0 [89.0–94.0]	14.7 [11.0–17.0]	0.0 [0.0–0.0]	–	14220 [13760–14672]
		Join+Σ	96.7 [95.0–98.0]	28.0 [19.0–33.0]	14.0 [9.0–18.0]	85.3 [82.0–90.0]	10787 [9011–12175]
	Qwen3-8B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	8667 [8311–9179]
		Σ	94.7 [94.0–95.0]	12.3 [11.0–14.0]	0.0 [0.0–0.0]	–	12390 [9581–15660]
		Join+Σ	96.0 [94.0–98.0]	35.0 [34.0–36.0]	48.0 [45.0–51.0]	85.3 [82.0–90.0]	11679 [9044–15886]
	Qwen2.5-14B	Prompt	0.0 [0.0–0.0]	0.0 [0.0–0.0]	0.0 [0.0–0.0]	–	10755 [10537–10947]
		Σ	92.7 [90.0–96.0]	20.0 [12.0–24.0]	0.0 [0.0–0.0]	–	17980 [16910–18736]
		Join+Σ	95.3 [95.0–96.0]	32.3 [23.0–38.0]	31.0 [28.0–35.0]	85.3 [82.0–90.0]	14546 [12923–17046]

implementation. This supports the bottleneck analysis in Section 6. An exception is Qwen2.5-1.5B over thresholds 0.70–0.85, where EM and join rate are effectively unchanged; this indicates that, in that interval, the guarded decision boundary is not crossed on the evaluated set (consistent with concentrated gate statistics and/or necessity-override dominance for the same examples). For this model, the remaining latency drift across thresholds is therefore best interpreted as run-order systems variation (e.g., warmup/caching effects under sequential sweep execution) rather than a causal threshold effect.

Table 7: **Conservative Join-over-Σ improvement bounds on Hotpot-RAG-JSON.** For each model, Δ is computed as (mean under $\bowtie + \Sigma$ minus mean under Σ); the bracketed interval is obtained by subtracting the corresponding 95% bootstrap CI endpoints, i.e., a conservative bound from existing summaries.

Model	ΔEM (Join$-\Sigma$)	ΔSupported (Join$-\Sigma$)
Qwen2.5-0.5B	9.8 [8.0, 12.0]	14.3 [13.5, 15.5]
Qwen2.5-1.5B	19.3 [15.5, 25.0]	47.8 [42.5, 52.0]
Qwen2.5-7B	13.3 [2.0, 22.0]	14.0 [9.0, 18.0]
Qwen2.5-14B	12.3 [-1.0, 26.0]	31.0 [28.0, 35.0]
Qwen3-1.7B	16.7 [10.0, 22.0]	24.5 [23.5, 26.5]
Qwen3-4B	18.2 [13.5, 23.5]	8.8 [7.0, 11.0]
Qwen3-8B	22.7 [20.0, 25.0]	48.0 [45.0, 51.0]

A.6 Trace-Derived Diagnostics

The runtime stores, for every request, the raw output, the executed trace, and timing telemetry. This allows post-hoc analyses without re-running any model. We report them for the three Qwen2.5 models (0.5B, 1.5B, 7B) whose per-example traces are released with the code; the script is `scripts/plot_diagnostics_from_traces.py`.

Anatomy of prompt-only failures. Figure 6 classifies each prompt-only output as strictly valid, as a well-formed answer followed by trailing text, or as a schema/label violation. No output is strictly valid, and 100% of prompt-only outputs reach the generation cap. On AGNews and SST-2 the dominant failure is non-termination after a correct-looking answer; violations (e.g., the non-schema label "Science/Tech") are concentrated in the smallest model. On SQuAD, Qwen2.5-7B wraps 93% of its answers in Markdown code fences, which no strict JSON consumer accepts.

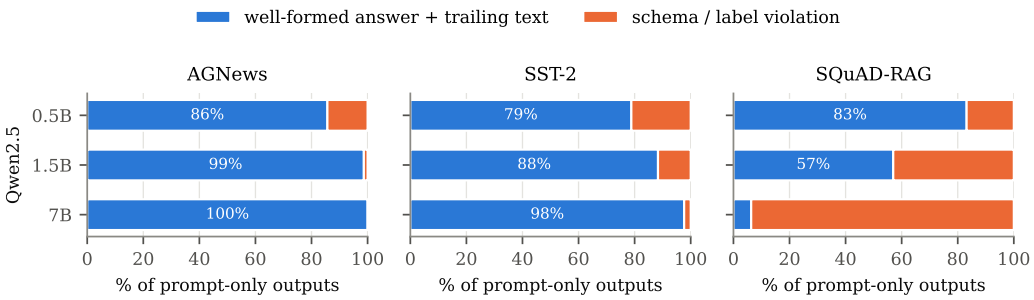

Figure 6: **Anatomy of prompt-only failures** (three seeds pooled; n=600 for 0.5B/1.5B and n=300 for 7B). Blue: the output begins with an answer that would satisfy the interface but continues with extra text; orange: the first answer already violates the schema or label language. Strictly valid outputs are 0% in all cells.

Comparison with post-hoc output repair. A natural objection is that a lenient parser could repair most prompt-only outputs. Table 10 scores prompt-only outputs with a generous post-hoc extractor that keeps only the first JSON object (AGNews, SQuAD) or the first line (SST-2) and then applies the same strict checks. Repair recovers much of the accuracy of the larger models on AGNews, but Σ matches or exceeds it in all nine model–task pairs, with the largest gaps for the smallest model and on SQuAD. Moreover, Σ achieves this by construction, without task-specific repair code, and certifies validity rather than hoping for it.

Latency decomposition. Table 11 gives the numbers behind Figure 4. "Other" is wall-clock time minus decoding, compilation, and retrieval; for guarded plans it is dominated by the gate's single forward pass. The guarded plan compiles the constraint twice (for the constrained gate entropy and

Table 8: **Latency and token diagnostics on structured tasks.** Values are means with 95% bootstrap confidence intervals over three seeds. For AGNews/SST-2, $\bowtie+\Sigma$ and Guarded $\bowtie+\Sigma$ are control-path overhead probes with no retrieval corpus attached (content-wise no-op join), so prompt-token counts remain aligned with non-join plans.

Task	Model	Plan	p50 ms	p95 ms	ms/tok	Prompt tok	Gen tok
AGNews	Qwen2.5-0.5B	Prompt-only	400 [380–410]	464 [406–524]	16.8	133	24.0
		Σ	3762 [3615–3945]	4109 [3860–4269]	279.3	133	13.7
	Qwen2.5-1.5B	Prompt-only	919 [869–963]	1103 [1061–1165]	38.9	133	24.0
		Σ	3966 [3814–4130]	4300 [4045–4449]	316.9	133	12.6
	Qwen2.5-7B	Prompt-only	865 [859–876]	948 [940–961]	36.6	134	24.0
		Σ	3829 [3799–3855]	4008 [3994–4034]	297.2	134	13.1
	Qwen3-1.7B	Prompt-only	741 [648–799]	876 [699–1005]	31.74	132.8	24.0
		Σ	6123 [5425–6748]	6531 [5772–7308]	489.65	132.8	12.4
		$\bowtie+\Sigma$	6087 [5486–6610]	7169 [6819–7481]	491.73	132.8	12.4
		Guarded $\bowtie+\Sigma$	6109 [5675–6585]	6638 [6057–7309]	490.99	132.8	12.4
	Qwen3-4B	Prompt-only	1834 [1740–2012]	1990 [1829–2266]	77.04	132.8	24.0
		Σ	6497 [5814–7494]	7595 [6735–8789]	896.44	132.8	7.4
		$\bowtie+\Sigma$	6531 [5944–7466]	8131 [6696–9259]	913.56	132.8	7.4
		Guarded $\bowtie+\Sigma$	6447 [5893–7312]	7118 [6357–8343]	884.62	132.8	7.4
	Qwen3-8B	Prompt-only	1796 [1738–1854]	1964 [1793–2104]	76.36	133.6	24.0
		Σ	6999 [5945–7642]	7584 [6419–8342]	617.51	133.6	11.4
		$\bowtie+\Sigma$	7071 [6139–7654]	7677 [6703–8276]	625.25	133.6	11.4
		Guarded $\bowtie+\Sigma$	7043 [6024–7654]	7620 [6474–8414]	620.14	133.6	11.4
	Qwen2.5-14B	Prompt-only	2239 [2037–2409]	2404 [2238–2616]	93.86	133.6	24.0
		Σ	6004 [5750–6257]	6728 [6281–7379]	805.10	133.6	7.8
		$\bowtie+\Sigma$	6037 [5741–6355]	6705 [6229–7432]	809.46	133.6	7.8
		Guarded $\bowtie+\Sigma$	6101 [5823–6338]	7104 [6379–7480]	826.70	133.6	7.8
SST-2	Qwen2.5-0.5B	Prompt-only	143 [137–148]	196 [191–199]	18.5	43	8.0
		Σ	3620 [3540–3670]	3924 [3748–4026]	453.9	43	8.0
	Qwen2.5-1.5B	Prompt-only	321 [308–333]	416 [402–444]	41.5	43	8.0
		Σ	3757 [3650–3845]	4103 [3842–4281]	472.1	43	8.0
	Qwen2.5-7B	Prompt-only	305 [303–308]	345 [336–350]	38.9	43	8.0
		Σ	3633 [3619–3656]	3765 [3752–3787]	453.7	43	8.0
	Qwen3-1.7B	Prompt-only	242 [226–253]	272 [265–281]	30.28	43.5	8.0
		Σ	5681 [5508–5960]	6427 [5969–6921]	715.66	43.5	8.0
		$\bowtie+\Sigma$	5658 [5416–5958]	6097 [5806–6436]	706.30	43.5	8.0
		Guarded $\bowtie+\Sigma$	5643 [5403–6025]	6069 [5806–6491]	703.19	43.5	8.0
	Qwen3-4B	Prompt-only	616 [574–675]	697 [620–841]	77.73	43.5	8.0
		Σ	6505 [6266–6900]	7624 [6808–8480]	933.27	43.5	7.2
		$\bowtie+\Sigma$	6685 [6319–7129]	7578 [6754–8280]	955.19	43.5	7.2
		Guarded $\bowtie+\Sigma$	6611 [5962–7316]	7807 [6630–8913]	960.25	43.5	7.2
	Qwen3-8B	Prompt-only	620 [609–639]	689 [630–773]	78.77	43.2	8.0
		Σ	6722 [5688–7376]	7285 [6240–7963]	842.48	43.2	8.0
		$\bowtie+\Sigma$	6678 [5604–7410]	7222 [6015–8033]	839.63	43.2	8.0
		Guarded $\bowtie+\Sigma$	6460 [5697–7412]	7130 [6016–7940]	811.72	43.2	8.0
	Qwen2.5-14B	Prompt-only	810 [734–853]	937 [752–1034]	102.10	43.2	8.0
		Σ	6114 [5832–6294]	6885 [6676–7035]	770.57	43.2	8.0
		$\bowtie+\Sigma$	6211 [6016–6390]	7480 [6735–8536]	795.73	43.2	8.0
		Guarded $\bowtie+\Sigma$	6229 [6202–6269]	7553 [6772–9032]	798.55	43.2	8.0

for decoding), which explains why it is slower than always-join even though it retrieves for only 14–32% of requests.

Citation format and Supported%. Supported% requires the model to cite retrieved documents by their 1-indexed rank. The serialized documents also display their corpus identifier in the header (`[DOC 1] (id=3935)`), and some models cite that identifier instead. In the $\bowtie+\Sigma$ SQuAD runs, out-of-range citations occur in 255 of 600 outputs for Qwen2.5-0.5B, 150 of 600 for Qwen2.5-1.5B, and 213 of 300 for Qwen2.5-7B; for Qwen2.5-7B, all 213 out-of-range citations equal the corpus

Table 9: **Guard-threshold sweep on SQuAD under Guarded $\bowtie + \Sigma$ at $k = 3$.** Lower thresholds increase join frequency; higher thresholds reduce retrieval more aggressively but also reduce EM.

	Qwen2.5-0.5B				Qwen2.5-1.5B				Qwen2.5-7B			
th	**EM%**	**Join%**	**wall ms**	**p95 ms**	**EM%**	**Join%**	**wall ms**	**p95 ms**	**EM%**	**Join%**	**wall ms**	**p95 ms**
0.70	24.7	82.5	8390	9482	11.0	13.5	9993	12562	11.3	20.3	8872	11623
0.75	22.7	73.5	7887	8553	11.0	13.5	9527	11871	10.7	17.3	8869	11590
0.80	20.8	64.5	7880	8508	11.0	13.5	9519	11827	10.0	16.0	8888	11531
0.85	15.7	51.3	7844	8476	11.0	13.5	9373	11578	10.0	15.7	8853	11514

Table 10: Σ **versus prompt-only decoding with post-hoc repair.** Strict accuracy (AGNews, SST-2) or strict EM (SQuAD-RAG-JSON, without $\bowtie$), in %, computed from the stored outputs of the runs in Table 1. "Repair valid" is the fraction of prompt-only outputs whose repaired answer passes the schema or label check.

Task	**Model**	**Repair valid**	**Prompt + repair**	**Σ (no repair)**
AGNews	Qwen2.5-0.5B	85.7	59.7	**68.8**
	Qwen2.5-1.5B	98.7	78.0	**78.5**
	Qwen2.5-7B	100.0	**78.3**	**78.3**
SST-2	Qwen2.5-0.5B	78.8	42.3	**57.7**
	Qwen2.5-1.5B	88.3	81.0	**92.3**
	Qwen2.5-7B	97.7	91.7	**92.7**
SQuAD-RAG	Qwen2.5-0.5B	83.2	2.0	**2.8**
	Qwen2.5-1.5B	57.0	3.2	**6.3**
	Qwen2.5-7B	6.3	0.3	**7.3**

identifier of a retrieved document. This explains the low Supported% of Qwen2.5-7B despite its high EM, and indicates that Supported% is a lower bound on citation support.

A.7 Additional Positioning and Law Tables

For readability of the main text, we place two reference tables here. Table 12 summarizes representation-level positioning relative to adjacent system classes, and Table 13 consolidates the exact rewrite laws used by the formal development.

A.8 Implementation and Reproducibility Notes

Why the structured tasks are informative despite their simplicity. AGNews and SST-2 are not used as challenge benchmarks; they are used as clean testbeds for the distinction between *semantic capability* and *interface-correct behavior*. Their role in the paper is causal isolation: prompt-only generation often reveals task knowledge in loose scoring while still failing the interface language, which allows us to cleanly test whether formal admissibility is a bottleneck for strict end-to-end correctness.

Why traces matter. The runtime records plan hashes, operator firings, gate outcomes, retrieved document IDs, validity diagnostics, and timing counters. These traces are procedural rather than epistemic: they reveal which control actions occurred, not whether the final answer is factually correct. This distinction is important both scientifically and operationally.

What the negative guarded-join result does and does not show. The guarded plan is a stress test for the abstraction, not a claim of a production-ready optimization. The scientific point is that once control is explicit, the slowdown becomes localizable to gate evaluation and repeated compilation rather than being hidden inside end-to-end latency.

Table 11: **Mean per-request latency components on SQuAD-RAG-JSON** (ms; three seeds).

Model	Plan	Wall	Decode	Compile	Retrieval	Other	Join%
Qwen2.5-0.5B	Prompt-only	1596	1596	0	0.0	0	0
	Σ	4235	803	3432	0.0	0	0
	$\bowtie + \Sigma$	4457	1078	3373	5.6	0	100
	Guarded $\bowtie + \Sigma$	8083	875	7184	2.0	22	32
Qwen2.5-1.5B	Prompt-only	3784	3783	0	0.0	0	0
	Σ	6288	3019	3268	0.0	0	0
	$\bowtie + \Sigma$	5373	2045	3322	5.6	0	100
	Guarded $\bowtie + \Sigma$	9754	2841	6859	1.0	53	14
Qwen2.5-7B	Prompt-only	3410	3410	0	0.0	0	0
	Σ	5334	2213	3121	0.0	0	0
	$\bowtie + \Sigma$	5277	2146	3125	5.2	0	100
	Guarded $\bowtie + \Sigma$	8918	2381	6492	0.8	44	15

Table 12: **Qualitative positioning.** ✓ indicates explicit first-class support, △ partial or implicit support, and ○ little or no explicit support. This is a schematic comparison of *representations*, not an exhaustive survey of what can be engineered in specific systems.

Approach	Hard Const.	Explicit $\bowtie$	Effect Annots.	Eq. Laws	Cost Visibility	Trace Artifact
Prompt-only	○	○	○	○	○	○
Constraint libs	✓	○	○	○	○	△
RAG pipelines	△	△	○	○	△	△
Prompt DSLs	△	△	△	○	△	△
SL Algebra	✓	✓	✓	✓	✓	✓

B Additional Proofs

B.1 Effect Typing and Typed Execution

This subsection provides the full inference-rule fragment referenced in Section 3. We use context Γ to track materialized fields and restart status, and separate effect typing from legality checking.

State-operator effect typing.

$$\frac{\mathsf{KVStable} \in \mathrm{annot}(o)}{\Gamma \vdash o : \mathsf{KVStable}} \text{ (S-KVStable)}$$

$$\frac{\mathsf{KVInvalid} \in \mathrm{annot}(o) \quad \Gamma(\mathsf{restart}) = \mathsf{open}}{\Gamma \vdash o : \mathsf{KVInvalid}} \text{ (S-KVInvalid)}$$

Logit-operator effect typing and local well-formedness.

$$\frac{\mathsf{StateDep} \in \mathrm{annot}(g)}{\Gamma \vdash g : \mathsf{StateDep}} \text{ (L-StateDep)}$$

$$\frac{\mathsf{StateIndep} \in \mathrm{annot}(g)}{\Gamma \vdash g : \mathsf{StateIndep}} \text{ (L-StateIndep)}$$

$$\frac{\Gamma \vdash g : \mathsf{StateDep} \quad \mathrm{reads}(g) \subseteq \Gamma(\mathsf{mat})}{\Gamma \vdash g\ \mathsf{wf}} \text{ (L-WF-Dep)}$$

$$\frac{\Gamma \vdash g : \mathsf{StateIndep}}{\Gamma \vdash g\ \mathsf{wf}} \text{ (L-WF-Indep)}$$

$$\frac{\Gamma \vdash o : \mathsf{KVStable}}{\Gamma \vdash o\ \mathsf{wf}} \text{ (S-WF-Stable)}$$

$$\frac{\Gamma \vdash o : \mathsf{KVInvalid}}{\Gamma \vdash o\ \mathsf{wf}} \text{ (S-WF-Invalid)}$$

Table 13: **Exact rewrite laws in the hard-mask fragment.** Each law is exact in the stated regime and justifies a concrete compiler or runtime optimization.

Law	Equation	Level	Interpretation
Mask fusion	$(\ell + m_A) + m_B = \ell + (m_A \wedge m_B)$	S1, S2	Sequential exact masks can be fused into one conjunction mask.
Bias–mask commutation	$(\ell + m) + b = (\ell + b) + m$	S1, S2	Finite biases may be reordered across exact masks.
Mask idempotence	$(\ell + m) + m = \ell + m$	S1, S2	Reapplying the same exact mask changes nothing.
Join–mask commutation	$(\ell + b(M_t)) + m = (\ell + m) + b(M_t)$	S1, S2	A finite logit-additive join can be reordered with masking.
Restart enforcement	$\bowtie^{\mathrm{Early}} \Rightarrow$ explicit restart boundary	—	A cache-invalidating early join is legal only across an explicit restart.

Sequence composition and guarded join.

$$\frac{\Gamma_0 \vdash o_1 \ \mathsf{wf} \quad \Gamma_1 = \mathrm{update}(\Gamma_0, o_1) \quad \cdots \quad \Gamma_{m-1} \vdash o_m \ \mathsf{wf} \quad \Gamma_m = \mathrm{update}(\Gamma_{m-1}, o_m)}{\Gamma_0 \vdash O \ \mathsf{wf} \triangleright \Gamma_m} \ (\text{O-SEQ})$$

$$\frac{\Gamma_m \vdash g_1 \ \mathsf{wf} \quad \cdots \quad \Gamma_m \vdash g_n \ \mathsf{wf}}{\Gamma_m \vdash G \ \mathsf{wf}} \ (\text{G-SEQ})$$

$$\frac{\Gamma_0 \vdash O \ \mathsf{wf} \triangleright \Gamma_m \quad \Gamma_m \vdash G \ \mathsf{wf}}{\vdash (O, G) \ \mathsf{wf}} \ (\text{PLAN-COMPOSE})$$

$$\frac{\Gamma \vdash \phi : \mathsf{NA} \quad \Gamma \vdash \bowtie_k \ \mathsf{wf}}{\Gamma \vdash \mathsf{Guard}[\phi](\bowtie_k) \ \mathsf{wf}} \ (\text{S-GUARD})$$

where update materializes newly produced fields and enforces restart-discipline transitions around KVInvalid boundaries.

Typed small-step rules. For typed execution, we use:

$$\frac{s_t^+ = O(s_t) \quad \Gamma_t^+ = \mathrm{update}(\Gamma_t, O)}{(\Gamma_t, s_t, (O, G)) \to (\Gamma_t^+, s_t^+, (O, G))} \ (\text{STEP-STATE})$$

$$\frac{\tilde{\ell}_t = G(s_t^+)\big(f_\theta(s_t^+)\big) \quad y_t \sim \mathrm{softmax}(\tilde{\ell}_t) \quad s_{t+1} = \mathsf{step}(s_t^+, y_t)}{(\Gamma_t^+, s_t^+, (O, G)) \to (\Gamma_{t+1}, s_{t+1}, (O, G))} \ (\text{STEP-LOGIT})$$

with $\Gamma_{t+1} = \mathrm{update}(\Gamma_t^+, G)$ for trace/materialization bookkeeping.

Proposition B.1 (Rule-level derivation for Proposition 4.1)**.** *Let $P = (O, G)$ with $\vdash P$ wf, and let $\mathsf{normalize}(P) = (O, G')$ where G' is obtained from G only by mask fusion, bias–mask commutation, and mask idempotence in the exact hard-mask fragment. Then $\vdash (O, G')$ wf is derivable in the rule system above.*

Derivation. From $\vdash P$ wf and PLAN-COMPOSE, obtain

$$\Gamma_0 \vdash O \ \mathsf{wf} \triangleright \Gamma_m, \qquad \Gamma_m \vdash G \ \mathsf{wf}.$$

By construction of the normalization fragment, every rewritten logit operator in G' is still annotated StateIndep (exact hard masks and finite biases), and rewrites do not introduce new reads. For each rewritten operator g_i':

$\mathsf{StateIndep} \in \mathrm{annot}(g_i') \Rightarrow \Gamma_m \vdash g_i' : \mathsf{StateIndep}$ (L-STATEINDEP) $\Rightarrow \Gamma_m \vdash g_i' \ \mathsf{wf}$ (L-WF-INDEP).

Applying G-SEQ yields $\Gamma_m \vdash G' \ \mathsf{wf}$. The state sequence is unchanged, so the original O-SEQ derivation for O is reused unchanged:

$$\Gamma_0 \vdash O \ \mathsf{wf} \triangleright \Gamma_m.$$

Finally, by PLAN-COMPOSE:

$$\frac{\Gamma_0 \vdash O \ \mathsf{wf} \triangleright \Gamma_m \quad \Gamma_m \vdash G' \ \mathsf{wf}}{\vdash (O, G') \ \mathsf{wf}} \ (\textsc{Plan-Compose}).$$

Hence $\vdash \mathsf{normalize}(P) \ \mathsf{wf}$.

Proposition B.2 (Side-condition invariance under typed execution)**.** *Assume* $\Gamma_t \vdash P$ wf *and* $(\Gamma_t, s_t, P) \to (\Gamma_{t+1}, s_{t+1}, P)$. *Then side conditions required to type-check* P *are preserved at* $t+1$. *In particular: (a) for every* g *in* P *with* StateDep $\in \mathrm{annot}(g)$, $\mathrm{reads}(g) \subseteq \Gamma_{t+1}(\mathsf{mat})$*; and (b) restart-discipline constraints for* KVInvalid*-annotated operators remain valid at their checking points in the next-step derivation.*

Proof sketch. The non-trivial part is context evolution. By definition of update, $\Gamma(\mathsf{mat})$ is monotone (fields are only added, not removed), so any previously valid inclusion $\mathrm{reads}(g) \subseteq \Gamma_t(\mathsf{mat})$ remains valid in $\Gamma_{t+1}(\mathsf{mat})$. Restart status changes only at explicit KVInvalid boundaries and according to the same discipline used by S-KVINVALID/O-SEQ; therefore subsequent checks are evaluated in a compatible restart context. Since operator annotations and plan structure are unchanged, all local premises used in the wf derivation remain derivable under Γ_{t+1}.

B.2 PROOF OF PROPOSITION 4.2

Let G be a finite composition of exact hard masks and finite state-independent biases. By associativity of vector addition, parenthesization is immaterial. By exact bias–mask commutation, every finite bias may be moved left across every exact hard mask. Thus G can be rewritten into a form where all finite biases appear first and all exact masks appear last.

Let $\{b_i\}_{i=1}^m$ denote the biases and $\{m_j\}_{j=1}^n$ the masks after reordering. Since finite biases compose by vector addition,

$$b^\star = \sum_{i=1}^{m} b_i.$$

Since exact hard masks are commutative and idempotent under $-\infty$ semantics,

$$m^\star = \bigwedge_{j=1}^{n} m_j.$$

Hence

$$G(\ell) = (\ell + b^\star) + m^\star.$$

Each rewrite step is exact under the stated assumptions, so G and G_{nf} are S1-equivalent.

B.3 PROOF OF THEOREM 4.3

We prove the theorem by induction on decoding steps.

Base case. At $t = 0$, the decoder samples from

$$\mathrm{softmax}\big(f_\theta(s_0) + m_{q_0}\big).$$

By assumption, the effective valid token set is non-empty at nonterminal steps, so this distribution is well-defined. Any token whose byte emission would produce an undefined intermediate DFA transition (or would emit EOS outside an accepting state) receives zero probability. Therefore the sampled token yields a well-defined next DFA state almost surely.

Inductive step. Assume the generated prefix after t steps corresponds to a well-defined DFA state q_t. At step $t+1$, the mask again excludes every token whose byte emission would lead to $\perp$. Therefore the sampled token yields a valid next state q_{t+1} almost surely.

Termination. By mask construction, EOS is admissible only when the current state belongs to F. By the almost-sure termination assumption, decoding emits EOS in finite time with probability 1. Therefore the final emitted byte string is finite and accepted by the DFA almost surely.

B.4 FINITE-PENALTY LEAKAGE BOUND

Practical systems often approximate $-\infty$ with a large negative penalty $-C$ for some $C > 0$. In that regime, invalid tokens are no longer impossible, but their probability mass remains quantitatively bounded.

Let $V_t = \mathsf{Valid}(q_t)$ denote the valid next-token set and $I_t = \mathsf{Invalid}(q_t) = \mathcal{V} \setminus V_t$ the invalid set. Define the finite-penalty mask

$$m_{q_t}^{(C)}(v) = \begin{cases} 0 & v \in V_t, \\ -C & v \in I_t. \end{cases}$$

Proposition B.3 (Finite-penalty leakage bound). *Assume $V_t \neq \emptyset$ and let $\ell_t \in (\mathbb{R} \cup \{-\infty\})^{|\mathcal{V}|}$ denote the pre-mask logits at step t. Define*

$$B_t := |I_t| \exp\left(\max_{i \in I_t} (\ell_t)_i - \max_{v \in V_t} (\ell_t)_v - C \right).$$

Under the finite mask $m_{q_t}^{(C)}$, the one-step leakage probability conditioned on the step-t state satisfies

$$\Pr(y_t \in I_t \mid q_t, \ell_t) \leq B_t.$$

Consequently, over a horizon T,

$$\Pr(\exists\, t \leq T : y_t \in I_t) \leq \sum_{t=1}^{T} \mathbb{E}[B_t].$$

In particular, if $B_t \leq \bar{B}_t$ almost surely for deterministic $\bar{B}_t$, then

$$\Pr(\exists\, t \leq T : y_t \in I_t) \leq \sum_{t=1}^{T} \bar{B}_t.$$

Proof. Condition on (q_t, ℓ_t) and let

$$Z_{\text{valid}} = \sum_{v \in V_t} \exp((\ell_t)_v), \qquad Z_{\text{invalid}} = \sum_{i \in I_t} \exp((\ell_t)_i - C).$$

The invalid sampling probability is

$$\Pr(y_t \in I_t \mid q_t, \ell_t) = \frac{Z_{\text{invalid}}}{Z_{\text{valid}} + Z_{\text{invalid}}} \leq \frac{Z_{\text{invalid}}}{Z_{\text{valid}}}.$$

Now

$$Z_{\text{invalid}} \leq |I_t| \exp\left(\max_{i \in I_t} (\ell_t)_i - C \right),$$

and because $V_t \neq \emptyset$,

$$Z_{\text{valid}} \geq \exp\left(\max_{v \in V_t} (\ell_t)_v \right).$$

Combining the inequalities yields $\Pr(y_t \in I_t \mid q_t, \ell_t) \leq B_t$. Taking expectation gives

$$\Pr(y_t \in I_t) = \mathbb{E}\big[\Pr(y_t \in I_t \mid q_t, \ell_t)\big] \leq \mathbb{E}[B_t].$$

The horizon bound then follows by a union bound over $t = 1, \ldots, T$:

$$\Pr(\exists\, t \leq T : y_t \in I_t) \leq \sum_{t=1}^{T} \Pr(y_t \in I_t) \leq \sum_{t=1}^{T} \mathbb{E}[B_t].$$

The deterministic-envelope version follows from $B_t \leq \bar{B}_t$ almost surely. □

C REPRODUCIBILITY MATERIALS

Datasets and splits. We use only public benchmarks: AGNews (test split), SST-2 from GLUE (validation split), SQuAD (validation split, for RAG-JSON evaluation), and HotpotQA distractor (validation split). Sample counts are fixed per task/model protocol and reported in the corresponding result tables.

Experimental protocol and hyperparameters. All main results use deterministic decoding with three random seeds per model–task–plan condition. Grounded QA uses retrieval depth $k = 3$. Model-scale evaluations use $n = 200$ for Qwen2.5-0.5B/Qwen2.5-1.5B/Qwen3-1.7B/Qwen3-4B and $n = 100$ for Qwen2.5-7B/Qwen3-8B/Qwen2.5-14B. The additional systems diagnostics use mask-fusion batches $\{1, 4, 8, 16\}$ with 5000 iterations and guarded-threshold sweep values $\{0.70, 0.75, 0.80, 0.85\}$. Task-level generation caps are 24 (AGNews), 8 (SST-2), 96 (SQuAD-RAG-JSON), and 120 (Hotpot-RAG-JSON) tokens; under strict JSON scoring, outputs truncated at this cap are counted invalid. Guarded-threshold settings are evaluated in ascending order; therefore small latency differences at fixed EM/join-rate (e.g., Qwen2.5-1.5B) are interpreted as measurement-order system variance rather than policy-induced changes.

Prompts. All plans use plain-text instruction prompts (no chat template and no stop sequences) that specify the required output format; the prompt is identical across plans for a given example, so plans differ only in their control operators. Exact prompt templates are in the released code.

Retriever and grounding setup. Grounded-QA runs use a dense retriever (`all-MiniLM-L6-v2` Sentence-BERT embeddings with an exact inner-product FAISS index over L2-normalized vectors) at retrieval depth $k = 3$. For SQuAD, the corpus consists of the contexts of the evaluation split. Join-Early serializes the retrieved documents (up to 6,000 characters) into the prompt before the output marker. The logit-additive join adds a constant bias of $+2.0$ to the 512 most frequent token ids of the retrieved documents. Hotpot context documents are constructed from provided supporting paragraphs. For reproducibility of **Supported%**, each prediction contains an answer string and cited document indices; indices are interpreted as 1-indexed ranks in the retrieved list, out-of-range indices are ignored, and support is counted iff the normalized predicted answer appears as a substring of at least one normalized cited document, using the first 500 characters of each document as stored in the trace (the model itself sees the full documents), which makes Supported% a conservative lower bound. The denominator is all evaluated examples N (not only valid outputs). This is intentionally a lexical citation-support metric, not semantic entailment or judge-model scoring. For Guarded Join analysis, we report gate entropy, gate decision, and retrieval-necessity signals; entropy is computed once from the initial next-token distribution under the same constraint context as decoding.