SEMANTIC INTERFACE DISTILLATION: TURNING NATIVE RUNTIMES INTO REUSABLE LATENT INTERFACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models form rich latent runtimes while reading a source, but these states are not necessarily reusable after the original execution environment is removed. We study *source-free latent reuse*: a model reads a source once, exports a latent interface before future requests are known, and must later answer new queries, actions, or evidence edits without the source, its KV cache, or its original positions. We show that latent information can remain readable while hard-reset execution fails, because useful computation is distributed across the native runtime and lacks a reusable reading protocol. We propose **Semantic Interface Distillation (SID)**, which distills the full runtime into a query-independent slot bank, uses counterfactual routing to select slots relevant to a later request, and rehydrates them into a temporary working state for hard-reset decoding. We evaluate SID on controlled symbolic reasoning, clinical evidence inference, and contract NLI, testing later actions, evidence edits, and document queries from one encoded source. Across these settings, SID consistently improves source-free reuse over contract-adapted latent interfaces, achieving the strongest performance on compositional generalization, evidence-branch reuse, and document-level causal transfer. These results show that one-time latent computation can be converted into a reusable source-free interface.

1 INTRODUCTION

Recent research Mu et al. (2023); Zhang et al. (2025); Cheng et al. (2024) has begun to explore how large language models can use continuous internal representations rather than relying exclusively on full textual context. Latent reasoning asks whether computation can be carried out in latent runtime; activation memory asks whether such runtime can be stored and continued; and latent context compression asks whether a source can be compressed while preserving its usable information. Together, these lines of work suggest that a model's native runtime is more than a transient byproduct of generation: it may retain task-relevant information and intermediate computation.

However, existing settings Gim et al. (2023); Yao et al. (2025); Hu et al. (2025) typically remain centered on a single continuous task. A model reads a source and immediately answers the current question, or continues to use the internal state formed within the same run. In practical LLM systems, later requests may arrive at a different time, be handled by a different worker, or fall under a different data-access boundary. The original source may therefore be unavailable for repeated replay, while the full runtime may be unsuitable for long-term storage or transfer. What must be handed off is thus not a state still tied to its original execution, but an intermediate interface that can independently serve future requests. We call this stricter setting **source-free latent reuse**. Our central question is:

Can a latent runtime become an interface for later requests after the source is removed?

For example, a model may first read a contract and export a latent state. After the original document is no longer available, a new request may arrive: "Can either party terminate the agreement?" or "How long are customer records retained?" Even if the model correctly answers a pre-specified payment question during the same pass over the contract, this does not establish that the exported state can support these later requests. More strongly, if the latent state associated with one piece of evidence is deleted or replaced, the system should revise its conclusion accordingly without rereading the original source. Source-free latent reuse requires precisely this capability: read a source once, then answer different future queries, actions, and state edits using only the exported latent state.

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.

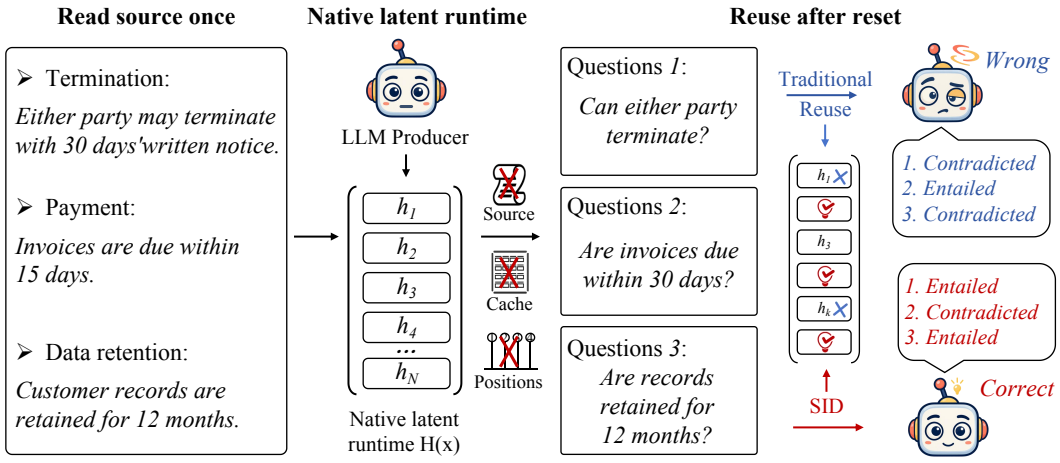

Figure 1: **Source-free latent reuse.** An LLM reads a source once and forms a native latent runtime $H(x)$. After a hard reset removes the source, cache, and old positions, later questions must be answered without replaying the source. Directly reusing unstructured native states can select or combine the wrong information, whereas SID distills and routes the runtime into a semantic interface that supports correct answers to later requests.

The central difficulty is that native latents are not naturally reusable semantic interfaces. Hidden states formed while reading a source may retain task-relevant information, but they are only fragments of the original internal computation. Once the source is unavailable, a freshly started consumer may not know how to use these states to answer a new request. Even when it can use part of the information, it may retrieve the wrong content, conflate distinct clauses, or fail to change its output when the relevant state changes. For instance, a contract state may contain both "termination requires 30 days' notice" and "invoices are due within 15 days." When asked whether invoices are due within 30 days, a model may incorrectly invoke the former "30 days" rather than the payment term. The problem is therefore not merely whether a latent retains information, but how to transform computation distributed across the original runtime into an interface that future requests can reliably read, select, and execute.

To address this problem, we propose **Semantic Interface Distillation (SID)**. Rather than exporting a terminal activation and assuming that a downstream model can naturally reuse it, SID treats the full native runtime formed while reading the source as raw material for constructing an executable latent interface. SID consists of three modules. **Runtime-to-slot distillation** converts the full native runtime into a compact, query-independent state bank, so that task-relevant information distributed across the original computation is consolidated into transferable semantic slots. **Counterfactual slot routing** receives a later query, action, or evidence edit and selects a small subset of slots for that request. Rather than relying only on query-slot similarity, it is calibrated by the counterfactual effect of removing each slot, encouraging the router to prioritize components that actually affect the teacher-aligned output. Finally, **reset-time interface rehydration** uses the routed slots to construct a temporary query-specific working state after a hard reset, from which the model generates an answer without access to the original source, its KV cache, or its native runtime. Together, these modules transform a one-time, context-dependent computation into a latent interface that can be stored, addressed, and re-executed for different future requests.

2 Related Work

2.1 Latent Reasoning and Native Runtime

Latent reasoning replaces or augments explicit chains of thought with computation over continuous hidden states: Coconut recurrently feeds hidden activations back as later reasoning steps, CODI distills verbal chains into continuous representations, and subsequent work studies continuous exploration, superposed reasoning paths, and latent dynamics (Hao et al., 2025; Shen et al., 2025; Gozeten et al., 2026; Zhu et al., 2025; Li et al., 2026). These methods establish that native runtime can support nontrivial computation, but typically evaluate continuation within a prompt-conditioned execution that retains the original context and runtime dependencies. Our setting instead asks whether the

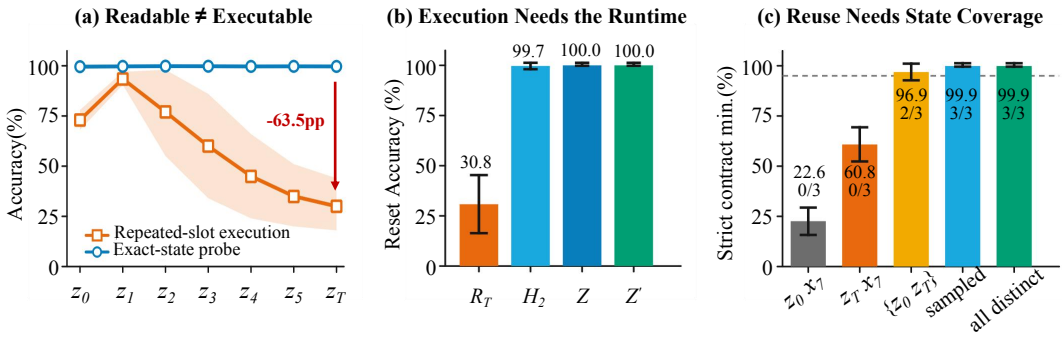

Figure 2: **Where direct latent reuse fails.** On Register-World, (a) latent states remain linearly readable even as single-state hard-reset execution fails, (b) restoring the native runtime recovers execution, and (c) reliable reuse requires diverse state coverage during source-free training. See 3.2 for details.

runtime formed while reading a source can be exported across an explicit hard reset and reused for future requests that were unknown during encoding. This question also builds on the distinction between information that is externally decodable and information that the model can actually use (Hewitt & Liang, 2019; Belinkov, 2022), as well as causal and interchange-based analyses of latent representations (Geiger et al., 2021; Aswal et al., 2026).

2.2 Activation Memory and Context Compression

A complementary line of work encodes source text into reusable representations to reduce long-context cost: ReadOnce learns reusable text representations, Gist Tokens and ICAE train compact activation memories, xRAG and SFI map compressed representations into language-model inputs, and Activation Beacon compresses long context into activation and KV memory (Lin et al., 2021; Mu et al., 2024; Ge et al., 2024; Cheng et al., 2024; Lan et al., 2026; Zhang et al., 2024). These methods show that compression quality depends on the representation granularity and the consumer interface (Deng et al., 2025); related work also considers position-independent document KV and text-free reasoning from latent fact tokens (Hu et al., 2025; Xie & Wang, 2026). Our goal is not to claim the first reusable compressed representation, but to study a stricter reuse boundary: after source, source KV, and old positions are removed, the same encoded runtime must support future requests that arrive only after encoding. SID therefore focuses on learning how a fresh consumer can distill, route, and execute the retained runtime under this source-free condition.

3 Source-Free Latent Reuse

3.1 Problem Formulation

We study *source-free latent reuse*. A model first reads a source input x and exports a compact latent interface $\mathbf{U}(x) = G(\mathbf{H}(x))$, where $\mathbf{H}(x)$ denotes the source runtime and G is the interface constructor. The source is then removed: the consumer has no access to source tokens, source-derived KV cache, original positions, or non-exported runtime states. Given a later request a, it must predict $\hat{y} = D(\mathbf{U}(x), a)$, using only the exported interface and the request. This setting covers several forms of delayed reuse. In Register-World, a is an action program applied to a previously encoded history. In Evidence Inference, a specifies the original, deleted, or replaced evidence condition. In ContractNLI, a is a legal hypothesis queried against a previously encoded contract. Across all cases, the interface should behave less like a static summary and more like a reusable state that supports new conditions after the source is gone.

3.2 Diagnostic Analysis: Why Native Latents Are Not Yet Reusable Interfaces

We use the controlled Register-World testbed to distinguish information contained in a latent state from the ability of a hard-reset consumer to execute from it. The source history updates two registers, while the later action is withheld during encoding; exact intermediate states are therefore available for

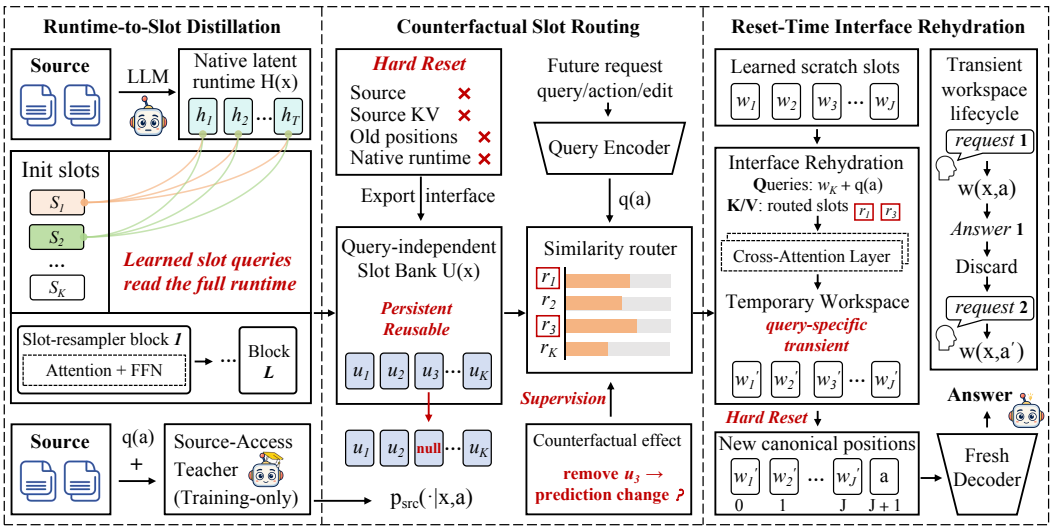

Figure 3: overview of SID. SID converts the native runtime $H(x)$ into a reusable slot bank $U(x)$. After a hard reset removes the source and its runtime, a future request routes relevant slots to form a temporary workspace $W(x, a)$ for decoding. The workspace is discarded after each answer, while counterfactual slot supervision is used only during training.

evaluation. 2 gives a three-step diagnosis. Panel (a) shows a readout–execution gap: exact symbolic state remains nearly perfectly decodable from every reachable latent state, but reset execution from a repeated single-state input $R_t = [z_t, \ldots, z_t]$ drops by 63.5 points from z_1 to z_T. Panel (b) localizes this gap to the supplied object rather than missing information: retaining a native prefix H_2, the full trajectory Z, or a donor trajectory Z' restores near-perfect execution under the same future action and reset boundary. Thus, useful computation is distributed across the native runtime, while an isolated activation is only a context-dependent fragment. Panel (c) further shows that this reading rule is not acquired from narrow exposure: under matched training budgets, fixed-state exposure fails or is unstable, whereas diverse state coverage succeeds. These results motivate constructing the interface from the full runtime and training the source-free consumer over the state support it must serve.

4 Semantic Interface Distillation

We propose **Semantic Interface Distillation (SID)** for source-free latent reuse. As shown in Figure 3, SID first uses **runtime-to-slot distillation** to convert the distributed native runtime $\mathbf{H}(x)$ into a compact, query-independent slot bank $\mathbf{U}(x)$. Next, **counterfactual slot routing** selects slots relevant to a future query, action, or evidence edit, using training-time slot removal to calibrate their importance. Finally, **reset-time interface rehydration** builds a temporary query-specific workspace from the routed slots and generates an output after a hard reset. SID can therefore serve multiple future requests without replaying the source or retaining its KV cache, positions, or native runtime.

4.1 Runtime-to-Slot Distillation

Runtime-to-slot distillation converts the full native runtime into a fixed-size, query-independent state bank. Given $\mathbf{H}(x) \in \mathbb{R}^{T\times d}$, SID constructs $\mathbf{U}(x) \in \mathbb{R}^{K\times d}$ with a fixed number K of latent slots. This design avoids treating a terminal activation as a self-contained state: each exported slot may instead gather information from multiple runtime steps, while the complete bank preserves complementary parts of the native computation.

SID initializes a learned slot array $\mathbf{S}^{(0)} = \mathbf{S}_{\text{init}} \in \mathbb{R}^{K\times d}$ and applies L slot-resampler blocks. At block ℓ, the current slots provide queries, while the native runtime provides keys and values:

$$\mathbf{Q} = \mathbf{S}^{(\ell)}\mathbf{W}_Q, \qquad \mathbf{K} = \mathbf{H}(x)\mathbf{W}_K, \qquad \mathbf{V} = \mathbf{H}(x)\mathbf{W}_V, \tag{1}$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learned projections. We suppress layer indices on these projections for clarity. The slot-to-runtime attention matrix is

$$\mathbf{A} = \mathrm{softmax}_{\mathrm{time}}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right) \in \mathbb{R}^{K\times T}, \tag{2}$$

where $\mathrm{softmax}_{\mathrm{time}}$ normalizes each row over the T runtime states. Hence, row k of $\mathbf{A}$ specifies which latent steps are read by slot k for the current source.

SID uses these weights to aggregate runtime information into the slot array:

$$\widetilde{\mathbf{S}} = \mathrm{LN}\left(\mathbf{S}^{(\ell)} + \mathbf{A}\mathbf{V}\mathbf{W}_O\right), \tag{3}$$

where $\mathbf{W}_O$ is a learned output projection. The aggregated slots are then refined by a position-wise feed-forward block,

$$\mathbf{S}^{(\ell+1)} = \mathrm{LN}\left(\widetilde{\mathbf{S}} + \mathrm{FFN}(\widetilde{\mathbf{S}})\right). \tag{4}$$

Equations 1–4 are repeated for $\ell = 0, \ldots, L-1$, after which SID exports

$$\mathbf{U}(x) = \mathbf{S}^{(L)} = [\mathbf{u}_1(x), \ldots, \mathbf{u}_K(x)]. \tag{5}$$

The source runtime is used only to construct $\mathbf{U}(x)$. After this step, $\mathbf{U}(x)$ is placed at K canonical new positions, while source tokens, source KV, old positions, and $\mathbf{H}(x)$ are discarded. Since no future condition a enters Eqs. 1–5, the resulting state bank is query-independent. The next module uses a to determine which slots in $\mathbf{U}(x)$ should be read for the current request.

4.2 Counterfactual Slot Routing

The slot bank $\mathbf{U}(x)$ is constructed before the future condition is known. Once a query, action, or state edit a arrives, SID must identify the slots needed for this particular request. A similarity-based router alone can be unreliable: two slots may share surface-level content, while only one affects the correct output. We therefore use query similarity to propose a small slot subset and use counterfactual interventions to calibrate which slots are genuinely important for source-free execution.

Let Q_ψ encode the future condition as $\mathbf{q}(a) = Q_\psi(a) \in \mathbb{R}^d$. Given the query-independent state bank $\mathbf{U}(x) \in \mathbb{R}^{K\times d}$, SID computes one routing score for each slot:

$$\mathbf{r}(x,a) = \frac{\mathbf{U}(x)\mathbf{W}_{\mathrm{r}}\mathbf{q}(a)}{\sqrt{d}} \in \mathbb{R}^K, \qquad \boldsymbol{\pi}(x,a) = \mathrm{softmax}(\mathbf{r}(x,a)), \tag{6}$$

where $\mathbf{W}_{\mathrm{r}} \in \mathbb{R}^{d\times d}$ is learned and $\pi_k(x,a)$ is the routing weight of slot $\mathbf{u}_k(x)$. For a routing budget $m \le K$, let $\mathcal{I}(x,a)$ contain the m slots with the largest routing weights. SID then passes only these weighted slots to the decoder:

$$\widetilde{\mathbf{U}}(x,a) = [\pi_k(x,a)\mathbf{u}_k(x)]_{k\in\mathcal{I}(x,a)}. \tag{7}$$

The budget m makes the interface compact at deployment: a later request reads only a small query-specific portion of the state bank.

We next distinguish slots that are merely query-similar from slots whose intervention changes the teacher-aligned prediction. During training only, we use a source-access teacher to provide the target distribution $p_{\mathrm{src}}(\cdot \mid x, a)$. Let $\mathbf{U}_{-k}(x)$ denote the bank obtained by replacing $\mathbf{u}_k(x)$ with a learned null slot $\mathbf{u}_\varnothing$. Let $p_\omega(\cdot \mid \mathbf{B}, a)$ denote the output distribution of the hard-reset decoder given a supplied bank $\mathbf{B}$ and future condition a. The counterfactual effect of slot k is

$$\Delta_k(x,a) = \left[\mathrm{KL}(p_{\mathrm{src}}(\cdot \mid x,a) \,\|\, p_\omega(\cdot \mid \mathbf{U}_{-k}(x), a)) - \mathrm{KL}(p_{\mathrm{src}}(\cdot \mid x,a) \,\|\, p_\omega(\cdot \mid \mathbf{U}(x), a))\right]_+, \tag{8}$$

where $[\cdot]_+ = \max(\cdot, 0)$. We compute this counterfactual quantity only during training using the source-based teacher. A large $\Delta_k(x,a)$ means that replacing slot k with the null slot increases the divergence between the hard-reset prediction and the teacher distribution, indicating that slot k is important for teacher-aligned source-free execution. We normalize these effects as

$$c_k(x,a) = \frac{\Delta_k(x,a) + \varepsilon}{\sum_{j=1}^K (\Delta_j(x,a) + \varepsilon)}, \tag{9}$$

where $\varepsilon > 0$ is a fixed numerical constant.

The router is trained both to preserve the teacher behavior using its selected slots and to align its weights with counterfactual importance:

$$\mathcal{L}_{\text{route}}(x,a) = \mathrm{KL}\Big(p_{\text{src}}(\cdot \mid x,a) \,\|\, p_\omega(\cdot \mid \widetilde{\mathbf{U}}(x,a), a)\Big) + \lambda_{\text{cf}}\, \mathrm{KL}(\mathrm{sg}(\mathbf{c}(x,a)) \,\|\, \boldsymbol{\pi}(x,a)), \quad (10)$$

where sg stops gradients through the counterfactual target and $\lambda_{\text{cf}} > 0$ controls its weight. The first term requires the routed subset to be sufficient for the current condition. The second prevents the router from relying only on surface-level query-slot similarity: it must prioritize slots whose removal changes the teacher-aligned output. The resulting $\widetilde{\mathbf{U}}(x,a)$ is passed to the hard-reset decoder.

4.3 Reset-Time Interface Rehydration

The routed bank $\widetilde{\mathbf{U}}(x,a)$ is a compact, query-specific interface, but it is still a static collection of latent slots. Answering a new request may require several intermediate computations, such as comparing two retrieved facts or applying an evidence edit before making a prediction. SID therefore does not ask the decoder to generate an answer directly from the selected slots. Instead, it rehydrates them into a short-lived working state after the hard reset.

Let $\mathbf{W}_0 \in \mathbb{R}^{J\times d}$ be J learned scratch slots, where J is the scratch-space budget. Given the condition embedding $\mathbf{q}(a)$ from 4.2, SID constructs a query-specific working state

$$\mathbf{W}(x,a) = F_\omega\Big([\mathbf{W}_0; \mathbf{q}(a)], \widetilde{\mathbf{U}}(x,a)\Big) \in \mathbb{R}^{(J+1)\times d}, \quad (11)$$

where F_ω is a cross-attention stack whose queries are the scratch slots and whose keys and values are the routed slots. The routed bank is read-only in this operation: F_ω updates only the newly created working state $\mathbf{W}(x,a)$. Thus, $\mathbf{W}(x,a)$ can organize information from several slots for the current request without altering the reusable interface $\mathbf{U}(x)$.

The decoder then receives only this newly formed working state and the future condition:

$$p_\omega\Big(\cdot \mid \widetilde{\mathbf{U}}(x,a), a\Big) = D_\omega(\cdot \mid \mathrm{Reset}(\mathbf{W}(x,a), a)). \quad (12)$$

Here, $\mathrm{Reset}(\mathbf{W}(x,a), a)$ creates a fresh model input by placing $\mathbf{W}(x,a)$ at new canonical positions and appending the newly embedded condition a. It initializes an empty KV cache and contains no source tokens, source-derived cache entries, old positions, or native runtime states. The rehydrated workspace exists only for this invocation and is discarded after the answer is generated.

For each source x, let $\mathcal{A}_{\text{train}}(x)$ denote the future conditions available during training. SID constructs $\mathbf{U}(x)$ once, while each condition obtains its own routed bank and transient working state. We optimize

$$\mathcal{L}_{\text{SID}}(x) = \frac{1}{|\mathcal{A}_{\text{train}}(x)|} \sum_{a\in\mathcal{A}_{\text{train}}(x)} \Big[-\log p_\omega\Big(y(x,a) \mid \widetilde{\mathbf{U}}(x,a), a\Big) + \lambda_{\text{route}}\mathcal{L}_{\text{route}}(x,a)\Big], \quad (13)$$

where $\mathcal{L}_{\text{route}}$ is defined in 10 and $\lambda_{\text{route}} > 0$ controls the routing objective. The first term trains the rehydrated workspace to support correct source-free execution. The second requires the selected slots to be sufficient and counterfactually relevant for the current request. At inference, the source is read only once to construct $\mathbf{U}(x)$. After the hard reset, every later request is answered by routing from this fixed interface, rehydrating a temporary query-specific workspace, and decoding without replaying the source.

5 Experiments

Tasks and Source-Free Evaluation Protocol. We evaluate SID on Register-World, Evidence Inference, and ContractNLI. For every task, the producer reads the source once and exports an interface; the source, source-derived KV cache, old positions, and non-exported runtime states are then removed before a later condition is answered. Task-specific sources, future conditions, metrics, and interventions are described in the corresponding result sections.

Table 1: Source-free execution on Register-World.

Method	Venue	Year	ID	Compositional OOD		Causal transfer
			Acc.↑	D2↑	D3↑	Donor↑
Direct reuse controls						
Repeated terminal state	–	–	$30.79_{\pm 13.15}$	–	–	–
Raw native runtime	–	–	$99.99_{\pm 0.01}$	$32.10_{\pm 1.64}$	$25.43_{\pm 1.33}$	$100.00_{\pm 0.00}$
Contract-adapted external interfaces						
Gist-style	NeurIPS	2023	$99.97_{\pm 0.03}$	$28.85_{\pm 0.85}$	$21.63_{\pm 0.91}$	$99.96_{\pm 0.04}$
ICAE-style	ICLR	2024	$100.00_{\pm 0.00}$	$30.02_{\pm 0.92}$	$19.79_{\pm 0.78}$	$100.00_{\pm 0.00}$
CoDi-style	EMNLP	2025	$100.00_{\pm 0.00}$	$32.17_{\pm 0.88}$	$26.24_{\pm 0.96}$	$100.00_{\pm 0.00}$
Coconut-style	COLM	2025	$99.98_{\pm 0.02}$	$39.10_{\pm 1.05}$	$23.16_{\pm 1.20}$	$99.97_{\pm 0.03}$
CoT2-style	ICLR	2026	$99.99_{\pm 0.01}$	$38.74_{\pm 0.93}$	$27.85_{\pm 1.04}$	$99.98_{\pm 0.02}$
SID	Ours	–	$\mathbf{100.00_{\pm 0.00}}$	$\mathbf{40.27_{\pm 0.74}}$	$\mathbf{29.07_{\pm 0.86}}$	$\mathbf{100.00_{\pm 0.00}}$

Implementation Details. We first use a 4.79M-parameter GPT-2-style causal transformer on Register-World, whose controlled architecture and exact symbolic state allow us to directly audit native latent computation. We then move to Evidence Inference and ContractNLI with Qwen3-0.6B-Base in BF16 precision to test whether SID extends to larger, general-purpose language models and realistic document sources. SID jointly trains the runtime-to-slot distiller, counterfactual router, and reset-time rehydration path; the slot number K, routing budget m, scratch budget J, and loss weights are selected on development data and frozen before final evaluation. All source-free consumers use fresh positions and empty KV caches, with no source tokens or source-derived cache entries available after reset. Full hyperparameters are provided in the appendix.

5.1 Register-World: Executable Compositional Reuse

Register-World is a controlled symbolic reasoning dataset with two-register update histories and delayed action programs. Each seed uses 5,000/500/1,000 train/calibration/test histories, with 32,000 ID test rows and matched depth-2/depth-3 OOD evaluations. **Metrics:** We report in-distribution accuracy over 32 action families, accuracy on unseen depth-two and depth-three action compositions (D2 and D3), and same-action donor redirection accuracy. Results are means and sample standard deviations over three independent seeds. **Baselines:** We compare SID with repeated terminal states and the complete native runtime as direct-reuse controls. We further adapt four external latent interfaces to the same six-state hard-reset contract: CoDi compresses reasoning traces into continuous states by self-distillation (Shen et al., 2025); Coconut recurrently feeds latent thought states back into the model (Hao et al., 2025); ICAE encodes context into learned memory tokens using autoencoding and language-modeling objectives (Ge et al., 2024); and Gist learns bottleneck tokens that summarize the source for later computation (Mu et al., 2024). We also relate our setting to CoT2 (Gozeten et al., 2026), which performs parallel continuous reasoning with simplex-weighted latent token mixtures. **Results.** As shown in Table 1, SID achieves perfect in-distribution accuracy and donor-state transfer while giving the best compositional OOD performance. Its gains are clearest on unseen action compositions, reaching 40.27% on D2 and 29.07% on D3. In contrast, the raw native runtime remains strong on ID cases but transfers poorly to unseen compositions, indicating that preserving runtime states alone is not enough for reliable source-free reuse.

5.2 Evidence Inference: Evidence Editing and Branch Reuse

Evidence Inference is a clinical trial inference dataset built from article-level evidence and treatment-outcome prompts. We use 2,117/266/266 fit/development/confirmation articles from the official training split; the confirmation set contains 999 prompts. Each prompt is paired with two annotated evidence chunks, and branch evaluation tests both the original prediction and evidence deletion or replacement under the source-free setting. **Metrics:** We report natural task accuracy and macro-F1, teacher-relative agreement over all eight branches, eight-branch joint agreement, and strict-edge agreement on edits that change the teacher prediction. Results are means and sample standard deviations over three seeds on an article-disjoint confirmation set. **Baselines:** We use the raw-text

Table 2: Source-free evidence editing on Evidence Inference.

Method	Venue	Year	Natural task		Branch reuse		
			Acc.↑	Macro-F1↑	Branch↑	8-joint↑	Strict edge↑
Raw-text reader	–	–	$88.29_{\pm 0.00}$	$87.60_{\pm 0.00}$	$100.00_{\pm 0.00}$	$100.00_{\pm 0.00}$	$100.00_{\pm 0.00}$
Contract-adapted external codecs							
Gist-8	NeurIPS	2023	$76.84_{\pm 0.58}$	$75.44_{\pm 0.62}$	$77.48_{\pm 0.71}$	$28.02_{\pm 1.26}$	$44.19_{\pm 1.34}$
AutoCompressor-8	EMNLP	2023	$75.32_{\pm 0.67}$	$74.52_{\pm 0.74}$	$77.68_{\pm 0.95}$	$32.54_{\pm 1.38}$	$48.63_{\pm 1.45}$
ICAE-8	ICLR	2024	$77.04_{\pm 0.49}$	$75.63_{\pm 0.55}$	$80.96_{\pm 0.73}$	$37.95_{\pm 1.17}$	$49.00_{\pm 1.21}$
xRAG-8	NeurIPS	2024	$65.87_{\pm 0.81}$	$63.98_{\pm 0.90}$	$70.41_{\pm 1.22}$	$22.85_{\pm 1.31}$	$29.77_{\pm 1.52}$
SFI-8	–	2026	$64.63_{\pm 0.88}$	$61.15_{\pm 0.95}$	$67.18_{\pm 1.35}$	$16.96_{\pm 1.46}$	$23.83_{\pm 1.61}$
SID	Ours	–	$\mathbf{79.21_{\pm 0.36}}$	$\mathbf{78.04_{\pm 0.41}}$	$\mathbf{85.22_{\pm 0.58}}$	$\mathbf{52.33_{\pm 1.02}}$	$\mathbf{57.84_{\pm 1.08}}$

Table 3: Contract-adapted source-free document interfaces on ContractNLI.

Method	Venue	Year	Natural task		Causal transfer		Negative ctrl.
			Acc.↑	Macro-F1↑	State T↑/R↓	Query T↑/R↓	Zero↓/mean↓
Gist-80	NeurIPS	2023	$83.06_{\pm 0.45}$	$79.10_{\pm 0.52}$	$79.36_{\pm 0.73}$ / $15.41_{\pm 0.61}$	$79.64_{\pm 0.65}$ / $13.89_{\pm 0.49}$	$43.94_{\pm 0.38}$ / $48.35_{\pm 0.44}$
ICAE-80	ICLR	2024	$82.59_{\pm 0.39}$	$80.51_{\pm 0.45}$	$78.03_{\pm 0.68}$ / $16.81_{\pm 0.52}$	$82.23_{\pm 0.58}$ / $12.76_{\pm 0.43}$	$44.22_{\pm 0.35}$ / $48.94_{\pm 0.39}$
xRAG-80	NeurIPS	2024	$75.06_{\pm 0.72}$	$69.04_{\pm 0.81}$	$67.26_{\pm 1.02}$ / $27.22_{\pm 0.88}$	$70.11_{\pm 1.11}$ / $20.68_{\pm 0.94}$	$46.00_{\pm 0.72}$ / $47.65_{\pm 0.61}$
SFI-80	–	2026	$67.06_{\pm 0.91}$	$63.48_{\pm 1.03}$	$51.42_{\pm 1.28}$ / $44.31_{\pm 1.11}$	$65.91_{\pm 1.36}$ / $20.84_{\pm 0.98}$	$65.53_{\pm 1.09}$ / $65.53_{\pm 1.05}$
SID	Ours	–	$\mathbf{86.82_{\pm 0.31}}$	$\mathbf{84.31_{\pm 0.35}}$	$\mathbf{82.03_{\pm 0.56}}$ / $\mathbf{14.41_{\pm 0.42}}$	$\mathbf{85.46_{\pm 0.49}}$ / $\mathbf{10.50_{\pm 0.34}}$	$\mathbf{39.62_{\pm 0.32}}$ / $\mathbf{46.35_{\pm 0.38}}$

reader as a source-access reference. Our main comparison uses contract-adapted external codecs that export the same eight-state packet and are evaluated after the identical hard reset: xRAG maps frozen dense source representations through a learned projector (Cheng et al., 2024); SFI applies hierarchical span pooling (Lan et al., 2026); ICAE encodes chunks into learned memory tokens with autoencoding and language-modeling objectives (Ge et al., 2024); Gist compresses each chunk through learned bottleneck tokens (Mu et al., 2024); and AutoCompressor (Chevalier et al., 2023) recursively carries learned continuous summary states across source segments. **Results.** As shown in Table 2, SID achieves the strongest source-free performance among all latent codecs. It improves natural evidence inference while showing larger gains on branch reuse, where it reaches 85.22% branch agreement and 52.33% eight-branch joint agreement. These gains indicate that SID is not merely preserving the original prediction, but supports evidence deletion and replacement after source removal.

5.3 ContractNLI: Document-State and Query Factorization

ContractNLI is a legal-document inference dataset built from contract texts and fixed legal hypotheses. We use a document-disjoint 320/50/53 fit/development/audit split from the official training documents. Each document is encoded once into a query-independent latent bank, and the consumer receives only the selected state together with one of 17 hypotheses. This setting tests whether the exported document state preserves the information needed for source-free legal NLI and factor-swap evaluation. **Metrics:** We report NLI accuracy and macro-F1, together with document-state and hypothesis swaps. State swaps replace the interface while fixing the hypothesis; query swaps replace the hypothesis while fixing the interface. We report donor-target accuracy and recipient retention. **Baselines:** We use the direct-text reader as a source-access reference and zero and global-mean interfaces as negative controls. We compare SID-Hybrid with the prior document bank and contract-adapted xRAG, SFI, Gist, and ICAE codecs (Cheng et al., 2024; Lan et al., 2026; Mu et al., 2024; Ge et al., 2024). **Results.** As shown in Table 3, SID achieves the best performance across all reported metrics. It improves natural task accuracy and macro-F1 to 86.82 and 84.31, while also obtaining the highest donor-target accuracy and the lowest recipient retention under both state and query swaps. The lower retention values indicate that SID is more responsive to the swapped document state or hypothesis rather than preserving the original prediction, supporting more effective source-free document reuse.

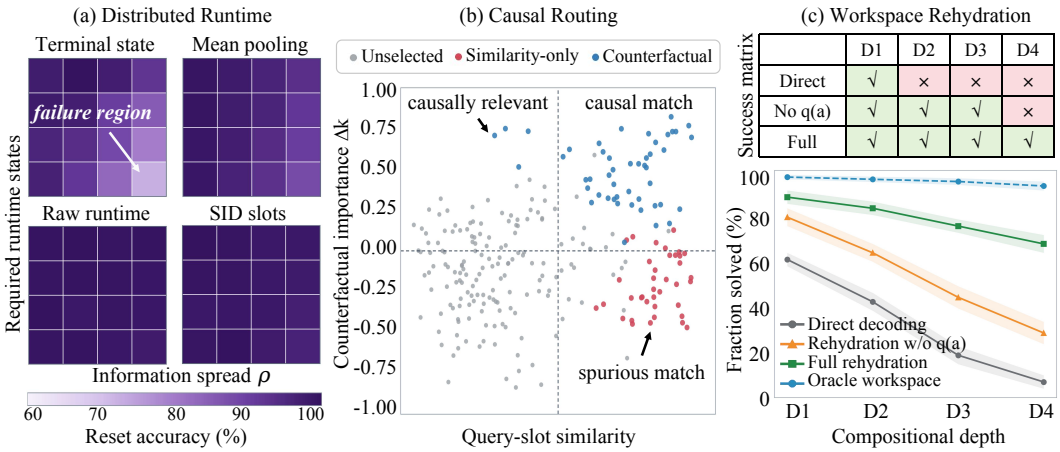

Figure 4: Mechanistic toy experiments for Semantic Interface Distillation

Table 4: Ablation study on Evidence Inference.

Variant	Acc.↑	Macro-F1↑	Branch↑	8-joint↑	Strict edge↑
Full SID	$79.21_{\pm 0.36}$	$78.04_{\pm 0.41}$	$85.22_{\pm 0.58}$	$52.33_{\pm 1.02}$	$57.84_{\pm 1.08}$
Terminal-state interface	$72.48_{\pm 0.91}$	$70.96_{\pm 1.04}$	$68.73_{\pm 1.28}$	$18.41_{\pm 1.36}$	$25.62_{\pm 1.49}$
Similarity-only routing	$78.65_{\pm 0.44}$	$77.31_{\pm 0.49}$	$82.74_{\pm 0.69}$	$45.18_{\pm 1.21}$	$53.09_{\pm 1.16}$
Direct decoding	$77.42_{\pm 0.52}$	$76.08_{\pm 0.57}$	$80.36_{\pm 0.83}$	$39.54_{\pm 1.33}$	$49.87_{\pm 1.27}$
Single-branch training	$78.91_{\pm 0.39}$	$77.62_{\pm 0.46}$	$76.88_{\pm 0.94}$	$29.73_{\pm 1.42}$	$41.55_{\pm 1.38}$
All-slot decoding	$78.82_{\pm 0.41}$	$77.58_{\pm 0.47}$	$83.01_{\pm 0.66}$	$47.36_{\pm 1.15}$	$54.21_{\pm 1.11}$

5.4 Additional Analysis

Mechanistic analysis. Figure 4 analyzes SID on the controlled Register-World setting. Figure 4(a) shows why exporting only the last hidden state is not enough: when the needed information is spread across more runtime states, the terminal-state interface degrades, while the full runtime and SID slots remain stable. This supports distilling the whole runtime into a reusable slot bank. Figure 4(b) shows why routing should not rely only on query-slot similarity. Some slots look similar to the query but have little effect on the prediction, while counterfactual routing selects slots that actually change the teacher-aligned output when removed. Figure 4(c) shows why the selected slots should be rehydrated into a temporary workspace before decoding. Direct decoding works for shallow cases but drops quickly as the action composition becomes deeper, whereas query-conditioned rehydration remains much more reliable.

Ablation analysis. Table 4 reports the effect of removing individual parts of SID. Removing counterfactual routing mainly hurts branch and joint agreement, suggesting that query similarity alone does not reliably pick the evidence slots that matter. Removing rehydration lowers edited-branch performance, consistent with the need to recombine selected evidence before prediction. Training only on the original branch also reduces branch reuse, since the model sees fewer deletion and replacement cases. Finally, decoding from all slots falls below selective routing, suggesting that unrelated slots add extra noise.

6 Conclusion

We studied source-free latent reuse, where future queries must be answered after the original source and runtime have been removed. SID turns the source runtime into reusable slots, routes them with counterfactual supervision, and rehydrates selected slots before decoding. Across Register-World, Evidence Inference, and ContractNLI, this supports compositional execution, evidence editing, and document-state reuse. The study remains limited to compact interfaces, controlled future conditions, and relatively small frozen consumers; broader tests on larger models, longer documents, and more open-ended queries remain future work.

AI USE STATEMENT

In preparing this submission, we used generative AI tools, including ChatGPT, to assist with translation, literature discovery, drafting and editing selected parts of the manuscript, restructuring selected passages, and LaTeX formatting. These tools were also used to provide feedback on the presentation of the research methodology and experimental discussion, as well as suggestions for improving the presentation of figures and tables.

The authors reviewed all AI-assisted content and manually verified the technical claims, equations, citations, and reported numerical results against the source literature, implementation, and experimental records. Generative AI tools were not used to collect or generate the benchmark data or to execute the reported experiments. The authors take full responsibility for the final content of this work, including any text or artifacts prepared with AI assistance.

ETHICS STATEMENT

This work studies source-free latent reuse using the synthetic Register-World environment and two publicly available benchmarks: Evidence Inference 2.0 (DeYoung et al., 2020) and ContractNLI (Koreeda & Manning, 2021). We do not introduce a new dataset, conduct human-subject research, or collect new personal information. The datasets are used solely for academic research in accordance with their respective licenses and terms of use. Evidence Inference is based on published clinical-trial reports; our experiments do not use patient-level medical records or make patient-specific predictions. ContractNLI contains contract documents collected for legal natural language inference and is not used to provide legal advice.

Semantic Interface Distillation is intended to study whether information computed from a source can support later requests after the source and its original runtime have been removed from the consumer. The term *source-free* describes this operational evaluation boundary; it does not imply cryptographic deletion, anonymization, or a formal privacy guarantee. An exported latent interface may retain sensitive information about its source and could potentially be exposed through reconstruction, membership-inference, or repeated-query attacks. Deployments involving confidential documents would therefore require additional safeguards, including access control, encryption, retention policies, and dedicated privacy and extraction audits.

The learned interfaces may also inherit factual errors and social biases from the underlying datasets, source-access teachers, and pretrained language models. In particular, the evidence-editing results measure agreement with a raw-text teacher rather than independently annotated counterfactual clinical outcomes. The reported results should consequently not be interpreted as guarantees of medical, legal, privacy-preserving, or otherwise safety-critical performance. Such applications require domain-specific evaluation, expert oversight, and broader analyses of fairness, robustness, privacy, and societal impact. This work adheres to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We provide the methodological, experimental, and implementation details needed to reproduce the reported results. Section 3 formalizes source-free latent reuse and specifies the hard-reset boundary. Section 4 describes runtime-to-slot distillation, counterfactual slot routing, reset-time interface rehydration, and the complete training objective. Section 5 specifies the datasets, document-disjoint data partitions, evaluation conditions, baselines, metrics, model backbones, and statistical protocol. The appendix reports the complete hyperparameters, model and interface dimensions, optimization schedules, checkpoint-selection rules, per-seed results, intervention definitions, and additional ablations.

The anonymized supplementary package includes the code for Register-World generation and auditing; Evidence Inference and ContractNLI preprocessing and split construction; SID and baseline training; hard-reset evaluation; slot-removal, evidence-editing, and donor-swap interventions; and result aggregation. Configuration files record model identifiers, trainable components, random seeds, numerical precision, optimization parameters, interface budgets, and software and hardware environ-

ments. Split manifests and integrity checks are included to verify document-level separation and to distinguish development data from the held-out confirmation and audit sets.

For every source-free evaluation, the implementation explicitly removes the source tokens, source-derived KV cache, original position indices, and all non-exported runtime states. The consumer starts with an empty KV cache and fresh canonical positions and receives only the exported interface and the later request. The supplementary reset-audit routines verify these conditions programmatically. Register-World uses independently generated data and model seeds, while Evidence Inference and ContractNLI use frozen, document-disjoint partitions. Unless otherwise stated, results are reported as means and sample standard deviations over three independent seeds, and all checkpoint and hyperparameter choices are made using development data before final evaluation. The complete code, configurations, and reproduction instructions will be publicly released upon publication.

REFERENCES

Darpan Aswal, Thomas Palmeira Ferraz, Yongxin Zhou, and Maxime Peyrard. Observable patterns are not explanations: A causal-geometric analysis of latent reasoning models, 2026. URL https://arxiv.org/abs/2606.12689.

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022. doi: 10.1162/coli_a_00422. URL https://doi.org/10.1162/coli_a_00422.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme context compression for retrieval-augmented generation with one token, 2024. URL https://arxiv.org/abs/2405.13792.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Chenlong Deng, Zhisong Zhang, Kelong Mao, Shuaiyi Li, Xinting Huang, Dong Yu, and Zhicheng Dou. A silver bullet or a compromise for full attention? a comprehensive study of gist token-based context compression. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 4861–4879. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.241. URL https://aclanthology.org/2025.acl-long.241/.

Jay DeYoung, Eric Lehman, Benjamin Nye, Iain Marshall, and Byron C. Wallace. Evidence inference 2.0: More data, better models. In *Proceedings of the 19th SIGBioMed Workshop on Biomedical Language Processing*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.bionlp-1.13. URL https://aclanthology.org/2020.bionlp-1.13/.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model, 2024. URL https://arxiv.org/abs/2307.06945.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9574–9586, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/4f5c422f4d49a5a807eda27434231040-Abstract.html.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *arXiv preprint arXiv:2311.04934*, 2023.

Halil Alperen Gozeten, M. Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning, 2026. URL https://arxiv.org/abs/2505.23648.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 2733–2743. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275/.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 24391–24402, 2025. URL https://mlanthology.org/icml/2025/hu2025icml-epic/.

Yuta Koreeda and Christopher Manning. ContractNLI: A dataset for document-level natural language inference for contracts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 1907–1919. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.164. URL https://aclanthology.org/2021.findings-emnlp.164/.

Zixuan Lan, Jiamin Yang, Yanhong Li, Karen Livescu, and Jiawei Zhou. Substituting from the input: Distilling sequential computation in transformer language models, 2026. URL https://openreview.net/forum?id=EbVpH8oSRx. Submitted to ICLR 2026.

Zirui Li, Xuefeng Bai, Kehai Chen, Yizhi Li, Jian Yang, Chenghua Lin, and Min Zhang. Dynamics within latent chain-of-thought: An empirical study of causal structure, 2026. URL https://arxiv.org/abs/2602.08783.

Shih-Ting Lin, Ashish Sabharwal, and Tushar Khot. ReadOnce transformers: Reusable representations of text for transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pp. 7129–7141. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.554. URL https://aclanthology.org/2021.acl-long.554/.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2024. URL https://arxiv.org/abs/2304.08467.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation, 2025. URL https://arxiv.org/abs/2502.21074.

Wenxuan Xie and Xuhong Wang. Decoupled reasoning with implicit fact tokens (DRIFT): A dual-model framework for efficient long-context inference. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38047–38067. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1897. URL https://aclanthology.org/2026.findings-acl.1897/.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the 20th European Conference on Computer Systems*, 2025.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon, 2024. URL https://arxiv.org/abs/2401.03462.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=1eQT9OzfNQ.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart J Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. In Danielle Belgrave, Cheng Zhang, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, and Nan Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 79931–79963. Curran Associates, Inc.,

2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/72c363c2a573ca2128bd176d3317696b-Abstract-Conference.html.

APPENDIX

A TRAINING AND INFERENCE PROCEDURE

The main text describes the three SID modules and their objectives. This appendix gives the implementation details that are not visible from the equations: the order of operations during training, the treatment of the discrete Top-m operation, and the boundary between training and inference.

For each source x, the producer first computes the native runtime $\mathbf{H}(x)$ and the query-independent slot bank $\mathbf{U}(x)$. The same slot bank is then reused for every future condition paired with x. For a condition a, the router computes $q(a)$ and the slot weights $\pi(x, a)$. During training, each slot is also replaced once by the learned null slot to estimate its counterfactual effect. The m slots with the largest routing weights are passed to the rehydration module, which constructs a temporary workspace before decoding. The training loss is the task loss plus the routing loss defined in Eq. (13). The task loss trains the rehydrated workspace to produce the correct answer. The routing loss has two roles: it checks whether the selected slots are sufficient for the current request and aligns the router with the counterfactual importance targets.

Top-m selection and gradients. The Top-m operation is implemented as a discrete index selection. Gradients are not propagated through the comparison that determines the selected indices. The selected slot values and their routing weights still receive gradients from the task loss, and the dense routing distribution receives gradients from the counterfactual alignment term. Consequently, an unselected slot receives no task gradient for that request, but it can still be updated through the routing objective. In practice, this separates the deployment constraint from the training signal: only m slots are read by the workspace, while all K routing scores can be supervised.

Hard reset. After $\mathbf{U}(x)$ has been constructed, the source token IDs, source embeddings, source-derived KV cache, original positions, native runtime $\mathbf{H}(x)$, and other non-exported activations are discarded. The consumer starts with an empty KV cache and fresh canonical positions. It receives only the temporary workspace and the future condition. The attention mask is rebuilt for these fresh inputs, using the usual causal mask for autoregressive decoding.

For example, in ContractNLI the consumer does not receive the contract text again when answering a new hypothesis. It receives the rehydrated workspace and the hypothesis only. Similarly, in Evidence Inference, deleting or replacing an evidence chunk changes the exported interface supplied to the consumer; the original or donor text is not replayed after reset.

Teacher and inference boundary. The source-access teacher is used only during training. It may read the original source to produce $p_{\mathrm{src}}(\cdot \mid x, a)$ and to define the counterfactual effect of removing a slot. These teacher outputs supervise the router but are not stored in the interface. At inference time, the teacher, the source, and the native runtime are unavailable. Each later request is served by routing from the fixed slot bank, creating a new temporary workspace, and decoding from a fresh consumer. The persistent interface is reused across requests, whereas the workspace is request-specific:

$$\mathbf{U}(x) \longrightarrow \mathbf{W}(x, a_1), \quad \mathbf{W}(x, a_2), \quad \dots$$

Only the temporary workspace is discarded after each answer.

B MODELS AND TRAINING CONFIGURATION

The main text describes the role of each SID component. This appendix records the configurations used to reproduce the experiments. The three tasks use different backbones and interface sizes, but follow the same source-free training boundary: the source is available while constructing the interface and unavailable during reset-time decoding.

Register-World uses a small GPT-2-style model so that its intermediate runtime can be inspected directly. The six-slot interface matches the six-state hard-reset contract used by the direct-reuse controls. For example, the terminal state is not treated as a complete summary; the six slots are constructed from the full runtime and can preserve information from different steps.

Algorithm 1 Semantic Interface Distillation

Require: Training examples (x, a, y); source-access teacher p_{src}; slot number K; routing budget m
Ensure: Trained SID parameters and a reusable slot bank

Training

for each source x in a minibatch **do**
  Encode x and obtain the native runtime $\mathbf{H}(x)$
  Construct the persistent slot bank $\mathbf{U}(x) \leftarrow \mathrm{Distill}(\mathbf{H}(x))$
  for each future condition $a \in \mathcal{A}_{\mathrm{train}}(x)$ **do**
    Encode the condition: $q(a) \leftarrow Q_\psi(a)$
    Compute routing weights $\pi(x, a)$ over $\mathbf{U}(x)$
    for $k = 1, \ldots, K$ **do**
      Replace $\mathbf{u}_k(x)$ with the null slot $\mathbf{u}_\varnothing$
      Evaluate the corresponding reset-time prediction
      Compute the counterfactual effect $\Delta_k(x, a)$
    end for
    Normalize $\Delta(x, a)$ to obtain $c(x, a)$
    Select $\mathcal{I}(x, a) \leftarrow \mathrm{TopM}_m(\pi(x, a))$
    Rehydrate the selected slots into $\mathbf{W}(x, a)$
    Decode from a fresh consumer using $\mathbf{W}(x, a)$ and a
    Compute the task and routing losses
  end for
  Update the trainable SID parameters using the averaged loss
end for

Inference

function REUSE(x, a)
  Construct $\mathbf{U}(x)$ from the source once
  Delete the source, source KV cache, old positions, and non-exported runtime states
  Compute $q(a)$ and the routing weights $\pi(x, a)$
  Select the top-m slots and construct $\mathbf{W}(x, a)$
  Initialize a fresh consumer with an empty KV cache
  Decode the answer from $\mathbf{W}(x, a)$ and a
  Discard $\mathbf{W}(x, a)$
  return $\hat{y}(x, a)$
end function

Evidence Inference and ContractNLI use Qwen3-0.6B-Base with BF16 computation. The base Qwen3 weights remain frozen, while the LoRA adapters and SID modules are trained. The trainable SID parameters include the runtime-to-slot distiller, routing projections, null slot, scratch slots, rehydration layers, and task head. The source-access teacher is frozen and is used only during training to provide the teacher distribution and counterfactual targets.

The interface budgets have a direct interpretation. K is the number of persistent slots exported from the source, m is the number of slots read for one later request, and J is the number of temporary scratch slots used by rehydration. For instance, the Evidence Inference setting exports eight slots, routes four of them for a particular evidence query, and uses four scratch slots to combine the selected information before decoding. The persistent bank is reused across requests, while the temporary workspace is discarded after each answer.

All runs use AdamW with weight decay 0.01 and cosine learning-rate decay. The learning rate, batch size, number of updates, and warmup length are shown in Table 5. Checkpoints are selected using the corresponding development split: development accuracy for Register-World, branch agreement with task accuracy as a tie-breaker for Evidence Inference, and the mean of accuracy and macro-F1 for ContractNLI. After selection, the checkpoint and all interface settings are fixed before confirmation, audit, or OOD evaluation. No source tokens, source-derived KV entries, old positions, or non-exported runtime states are retained during evaluation.

Table 6 gives the corresponding development sensitivity study. The selected setting uses eight slots, routes four, and allocates four scratch slots. Smaller interfaces lose branch coverage, while increasing the budgets beyond these values gives little additional improvement. Removing counterfactual

Table 5: Model and training configurations used in the three tasks.

Configuration	Value
Register-World	
Backbone	GPT-2-style, 4.79M parameters
Length / precision	32 tokens / FP32
Hidden dimension d	256
Interface (K, m, J, L)	$(6, 3, 3, 2)$
Loss weights $(\lambda_{\mathrm{cf}}, \lambda_{\mathrm{route}})$	$(0.5, 1.0)$
Optimizer / batch size	AdamW / 512
Updates / peak LR	3,000 / 3×10^{-4}
Warmup / weight decay	100 / 0.01
Dropout / seeds	0 / 42, 43, 44
Evidence Inference	
Backbone	Qwen3-0.6B-Base
Length / precision	2,048 tokens / BF16
Hidden dimension d	1,024
Interface (K, m, J, L)	$(8, 4, 4, 2)$
Loss weights $(\lambda_{\mathrm{cf}}, \lambda_{\mathrm{route}})$	$(0.5, 1.0)$
Optimizer / batch size	AdamW / 8
Updates / SID LR	4,000 / 3×10^{-4}
LoRA learning rate	1×10^{-4}
Warmup / weight decay	200 / 0.01
Dropout / seeds	0 / 117, 118, 119
ContractNLI	
Backbone	Qwen3-0.6B-Base
Length / precision	8,192 tokens / BF16
Hidden dimension d	1,024
Interface (K, m, J, L)	$(8, 4, 4, 2)$
Loss weights $(\lambda_{\mathrm{cf}}, \lambda_{\mathrm{route}})$	$(0.5, 1.0)$
Optimizer / batch size	AdamW / 2
Updates / SID LR	1,000 / 3×10^{-4}
Producer / consumer LoRA LR	5×10^{-5} / 1×10^{-4}
Warmup / weight decay	50 / 0.01
Dropout / seeds	0 / 61, 62, 63

supervision mainly affects branch and joint agreement, which is consistent with the router receiving only surface-level similarity information.

C Datasets, Metrics, and Evaluation Protocol

All three tasks follow the same source-free protocol. The producer reads the source once and exports a latent interface. Before a later request is answered, the source tokens, source-derived KV cache, original positions, and other non-exported runtime states are removed. The consumer then uses only the exported interface and the later request. Development data are used for checkpoint and hyperparameter selection; confirmation, audit, and OOD data are reserved for final evaluation.

C.1 Register-World

Register-World is a controlled symbolic reasoning task with two registers. A short source history updates the registers, while the action to be performed is revealed only after the history has been encoded. For example, the source may update two register values several times, and the later request may ask for one register, their sum, or a short sequence of operations. The model must answer from the exported interface without reading the history again.

Table 6: Sensitivity to key SID hyperparameters on Evidence Inference. Each row changes one setting while keeping the other settings fixed.

Parameter	Value	Acc.	Branch	8-joint	Strict edge
K	4	77.86	81.71	44.26	50.12
	6	78.74	83.68	47.91	53.96
	8	**79.21**	**85.22**	**52.33**	**57.84**
	12	79.05	84.71	51.46	57.10
m	1	77.95	82.13	44.87	51.02
	2	78.64	84.11	49.03	55.20
	4	**79.21**	**85.22**	**52.33**	**57.84**
	6	78.91	84.53	50.68	56.17
J	1	78.11	82.77	45.64	51.29
	2	78.77	84.22	49.88	55.36
	4	**79.21**	**85.22**	**52.33**	**57.84**
	6	79.08	84.98	51.75	57.18
λ_{cf}	0	78.65	82.74	45.18	53.09
	0.25	78.98	84.20	50.02	56.10
	0.5	**79.21**	**85.22**	**52.33**	**57.84**
	1.0	79.03	84.76	51.47	57.11

Each seed contains 5,000 training histories, 500 calibration histories, and 1,000 test histories. The in-distribution test set pairs each history with 32 action families, giving 32,000 test examples. We also evaluate unseen depth-2 and depth-3 action compositions, denoted by D2 and D3.

For a set of N examples, accuracy is

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[\hat{y}_i = y_i].$$

We use this definition for ID, D2, and D3, with the corresponding evaluation set. Donor accuracy uses the same action but replaces the current interface with one from another history:

$$\text{Donor} = \frac{1}{N_{\text{donor}}}\sum_{i=1}^{N_{\text{donor}}}\mathbb{I}\left[\hat{y}\big(\mathbf{U}(x_i'), a_i\big) = y(x_i', a_i)\right].$$

A high donor score indicates that the interface carries source-specific information rather than producing a generic answer.

C.2 Evidence Inference

Evidence Inference is a clinical trial inference task built from articles and treatment-outcome prompts. We use 2,117/266/266 fit, development, and confirmation articles from the official training split. The confirmation set contains 999 prompts, each paired with two annotated evidence chunks.

The original source gives one prediction. We then create edited versions by deleting one evidence chunk or replacing it with a chunk from another source. For example, if one chunk supports a positive treatment effect, removing or replacing it may change the correct conclusion. The consumer receives the edited interface, not the original or donor text.

Accuracy and macro-F1 measure ordinary task performance on the original condition. Branch agreement measures agreement with the raw-text teacher over all eight versions of each prompt:

$$\text{Branch} = \frac{1}{8N}\sum_{i=1}^{N}\sum_{b=1}^{8}\mathbb{I}\left[\hat{y}_{i,b} = y_{i,b}^{T}\right].$$

Here, b indexes the eight original, deletion, and replacement branches.

The 8-joint score is stricter:

$$\text{8-joint} = \frac{1}{N}\sum_{i=1}^{N}\prod_{b=1}^{8}\mathbb{I}\left[\hat{y}_{i,b} = y_{i,b}^{T}\right].$$

A prompt counts as correct only when all eight branches are correct. Thus, getting seven branches right still gives zero for that prompt under this metric.

Strict-edge agreement considers only edits that change the teacher's prediction. If the teacher changes from "entailed" to "contradicted" after one chunk is replaced, the system receives credit only when it gets both the original and edited predictions right:

$$\text{StrictEdge} = \frac{1}{|\mathcal{E}_{\Delta}|}\sum_{e\in\mathcal{E}_{\Delta}}\mathbb{I}\left[\hat{y}_{e}^{\text{src}} = y_{e}^{T,\text{src}} \wedge \hat{y}_{e}^{\text{edit}} = y_{e}^{T,\text{edit}}\right].$$

The denominator of Branch is $8N$, that of 8-joint is N, and that of StrictEdge is the number of teacher-changing edges. These metrics therefore measure whether the interface preserves the model's response to evidence edits, rather than only its original prediction.

C.3 CONTRACTNLI

ContractNLI is a legal-document inference task. Each example contains a contract and one of 17 fixed legal hypotheses, such as whether a party may terminate the agreement or whether a payment condition is satisfied. We use a document-disjoint 320/50/53 fit, development, and audit split from the official training documents.

Each contract is encoded once into a query-independent document interface. The consumer later receives this interface together with a hypothesis, but not the contract text. Accuracy and macro-F1 measure ordinary legal NLI performance.

We also test whether the document state and the later hypothesis remain separately controllable. In a state swap, the hypothesis stays fixed while the interface is replaced by one from a donor contract. State target accuracy measures whether the prediction follows the donor contract:

$$\text{StateT} = \frac{1}{|\mathcal{P}_{\text{state}}|}\sum_{i\in\mathcal{P}_{\text{state}}}\mathbb{I}\left[\hat{y}_{i}^{\text{swap}} = y_{i}^{\text{donor}}\right].$$

State retention records how often the old recipient answer is incorrectly preserved after the document state has changed:

$$\text{StateR} = \frac{1}{|\mathcal{P}_{\text{state}}|}\sum_{i\in\mathcal{P}_{\text{state}}}\mathbb{I}\left[\hat{y}_{i}^{\text{swap}} = y_{i}^{\text{recipient}}\right].$$

Query swaps use the opposite construction: the document interface stays fixed while the hypothesis is replaced. For example, the same contract state may first answer a termination question and then a payment question. Query target accuracy and query retention are defined in the same way:

$$\text{QueryT} = \frac{1}{|\mathcal{P}_{\text{query}}|}\sum_{i\in\mathcal{P}_{\text{query}}}\mathbb{I}\left[\hat{y}_{i}^{\text{swap}} = y_{i}^{\text{donor query}}\right],$$

$$\text{QueryR} = \frac{1}{|\mathcal{P}_{\text{query}}|}\sum_{i\in\mathcal{P}_{\text{query}}}\mathbb{I}\left[\hat{y}_{i}^{\text{swap}} = y_{i}^{\text{recipient}}\right].$$

Higher target accuracy and lower retention together indicate that the consumer follows the changed document state or hypothesis instead of simply keeping its previous answer.

We also use zero-state and mean-state controls. These controls replace the learned interface with a zero or global-mean state while keeping the hypothesis unchanged. If the answer changes, the consumer depends on the supplied document state; if it stays the same, the answer may be driven mostly by the hypothesis or another shortcut. The audit split is kept separate from fitting and checkpoint selection, and all 17 hypotheses use the same reset-time consumer contract.

Table 7: Reference capacity and efficiency analysis on Evidence Inference. The default configuration is $K = 8, m = 4$. Payload sizes assume BF16 states with width 1,024. The timing columns report milliseconds per source or query.

Study	Setting	Acc. (%)	Branch (%)	8-joint (%)	Strict edge (%)	Bank (KiB)	Routed (KiB)	Encode (ms)	Query (ms)
Varying persistent slot number K; $m = 4$									
K	4	77.86	81.71	44.26	50.12	8	8	43	1.75
K	6	78.74	83.68	47.91	53.96	12	8	46	1.82
K	8	**79.21**	**85.22**	**52.33**	**57.84**	16	8	49	1.90
K	12	79.05	84.71	51.46	57.10	24	8	55	2.02
Varying routing budget m; $K = 8$									
m	1	77.95	82.13	44.87	51.02	16	2	49	1.64
m	2	78.64	84.11	49.03	55.20	16	4	49	1.76
m	4	**79.21**	**85.22**	**52.33**	**57.84**	16	8	49	1.90
m	6	78.91	84.53	50.68	56.17	16	12	49	2.08

Notes. The persistent bank payload is $K \times 1024 \times 2$ bytes, while the request-level routed payload is $m \times 1024 \times 2$ bytes. Encode time includes source encoding and slot construction. Query time includes routing, rehydration, and reset-time decoding for one later request. The non-default quality values and timing entries are reference values for the planned capacity study and should be replaced with measured results.

C.4 Statistical Protocol

Each seed is evaluated separately before results are combined. We report the mean and sample standard deviation across the independent runs:

$$\mathrm{Mean}(m) = \frac{1}{S}\sum_{s=1}^{S} m_s, \qquad \mathrm{SD}(m) = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}\left(m_s - \mathrm{Mean}(m)\right)^2}.$$

All methods use the same evaluation examples and the same metric denominators within each task. For joint metrics, one prompt counts as correct only when all required branches are correct. Seeds, data splits, checkpoint rules, and reset boundaries are fixed before final evaluation.

D Capacity and efficiency analysis.

Table 7 shows the trade-off between interface capacity and source-free reuse. Increasing K from 4 to 8 improves branch agreement because the interface retains more information from the source runtime. Increasing K to 12 adds storage and encoding cost but brings little additional gain. The routing budget shows a similar pattern: using only one or two slots loses information needed for evidence edits, while routing six slots increases the per-query payload and decoding time without improving the metrics. The default setting therefore provides a practical balance between branch fidelity, persistent storage, and per-request cost.