BRANCHFENCE: CLOSING RECOVERABLE QUERY-PLANNING HAZARDS UNDER FIXED NATIVE EXPOSURE

Anonymous authors
Paper under double-blind review

ABSTRACT

On JOB query 17a, PostgreSQL 16.2 selects a plan that spills 248 MB and runs for 4.2 s even though its current candidate set already contains a nonspilling continuation that finishes in 0.9 s. This missed escape is common in held-out planner traces: 215/264 JOB, 274/356 TPC-H SF-100, and 467/628 TPC-DS SF-100 failure-bearing decisions expose a demonstrated nonviolating continuation for another native candidate. The interval before that alternative disappears is a *recoverable hazard window*. BranchFence acts before it closes: the controller preserves PostgreSQL's enumerator, rolls up to five exposed actions through eight deterministic planner transitions, averages continuation-cost predictions, penalizes peak spill/timeout/OOM/invalidity risk, and defers when the selected action's peak-risk score has high binary entropy. Across five training seeds and three split seeds, BranchFence reaches normalized mean latency 0.548/0.587/0.617 on JOB/TPC-H/TPC-DS, versus 0.600/0.628/0.650 for an architecture-matched $H = 0$ controller; workload-specific failure events fall from 382 to 263, 41 to 23, and 238 to 149. The paired latency and failure intervals exclude zero on every workload, the ordering persists under five-fold template holdout, and spill-prone JOB queries gain a median $3.8\times$ speedup with 12.2 ms total planning time per query. Spill and timeout risk can therefore be intercepted at the planning decision where PostgreSQL still exposes a safer branch, producing reliability gains through targeted reranking.

1 INTRODUCTION

PostgreSQL's dangerous planning error can be a pruning decision rather than a missing plan. At one dynamic-programming frontier, local total cost favors a join that consumes memory headroom; returning that path removes a safer alternative, and the later spill only reveals a commitment already made. Figure 3 traces this event for JOB query 17a. The useful control point is the final state at which the exposed escape can still replace the hazardous branch.

We name this state interval a *recoverable hazard window*. A diagnostic decision belongs to such a window when its logged native continuation ends in a violation and the trace corpus demonstrates a nonviolating continuation from another action in the same exposed set. The definition separates two optimization targets: extending search when no suitable action is exposed, and reranking when exposure is adequate but local cost misses downstream resource consequences. It also supplies decision-level observables—opportunity, warning lead, risk separation, and capture—that connect a learned scorer to the failures it prevents.

Existing learned query optimizers improve cardinality inputs (Kipf et al., 2019; Hilprecht et al., 2020; Yang et al., 2020), construct plans (Marcus et al., 2019; Yang et al., 2022), choose completed plans or hint configurations (Marcus et al., 2021; Zhu et al., 2023; Anneser et al., 2023), or spend a larger offline search budget (Tao et al., 2025). Because these interfaces can alter both exposure and ranking, their endpoint gains do not decompose the two mechanisms; learned-optimizer conclusions are sensitive to precisely such protocol choices (Lehmann et al., 2024). Our fixed-exposure question is more surgical: when PostgreSQL has already generated an adequate action, can bounded future-state reasoning recognize its value before that action disappears?

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.

BranchFence implements this intervention at PostgreSQL's planner hooks. A typed graph records the partial plan, legal bindings, and resource headroom. For up to five native candidates, deterministic symbolic transitions and a selective state-space model propagate eight planning steps; shared heads estimate completed-plan cost and the maximum probability of spill, timeout, OOM, or invalidity. Averaging per-step cost estimates keeps horizons on a common score range, while binary entropy of the selected peak-risk score gates learned selection.

Across workloads, three quarters or more of audited failure-bearing decisions expose a demonstrated recovery action four or five transitions before commitment, and BranchFence captures 72.8–82.3% of those windows. The matched $H = 0$ path holds representation, targets, score range, fallback, and serving fixed while removing future-state propagation and horizon aggregation. In the 5×3 seed–split study, full rollout lowers normalized mean latency by .033–.052 and workload-specific failure incidence by 1.1–1.4 points; five-fold template holdout, distribution shifts, and shared-resource contention preserve the ordering. The gain lands where the decision mechanism predicts: spill-prone JOB queries improve by a median $3.8\times$, compared with $1.4\times$ for already-safe queries.

Together, the audit and controller reveal where learned optimization can help a mature dynamic program: after useful paths have been generated but before local cost commits to delayed resource damage. Recoverability and capture measure that interval directly, while the matched no-rollout path identifies the effect of the complete future-state scoring pathway under fixed exposure. The PostgreSQL realization closes most observed windows with 12.2 ms/query total planning time and moves mean latency, p95 latency, and execution reliability together. Pre-commitment hazard capture thereby changes the reliability–latency frontier at the decision that creates the failure.

2 Selective Control of Recoverable Hazard Windows

Planning hazard window. At planning step t, PostgreSQL exposes partial plan G_t, candidate set $\Gamma(G_t, K)$, and native choice a_t^{nat}. Let $\rho_t(a)$ be the maximum predicted probability of spill, timeout, OOM, or invalidity along the H-step native-transition rollout that begins with a. The recoverable-risk gap is

$$\Delta_t = \rho_t(a_t^{\mathrm{nat}}) - \min_{a \in \Gamma(G_t, K)} \rho_t(a).$$

A positive gap signals that the same exposed set contains a lower-risk continuation. In the empirical audit, we call an observed decision recoverable when the native continuation carries a realized violation label and at least one logged same-set continuation is nonviolating. The warning lead is the number of planning transitions remaining at the earliest such state. Recoverability is therefore a decision-level, trace-policy-conditioned property of demonstrated continuations.

Fixed native exposure. Let $\mathcal{A}(G_t)$ be the distinct legal paths presented at a hook invocation after PostgreSQL's native feasibility checks. We retain a_t^{nat} and fill the remaining $K - 1$ slots in ascending native total-cost order, breaking ties by a stable relation/operator key; if fewer than K paths are legal, $\Gamma(G_t, K) = \mathcal{A}(G_t)$. Every fixed-exposure comparator receives this ordered set and may change only its scorer. The same construction is applied to a symbolic successor during rollout. A commitment occurs when the hook returns a selected path to the native dynamic-programming table and alternatives removed from the active frontier are no longer available at the next state.

Selective-control objective. The controller either returns an intervention action $a_t \in \Gamma(G_t, K)$ or defers to PostgreSQL's native argmin over that set. Its objective combines predicted continuation cost and $\rho_t(a)$; intervention is accepted only when peak-risk entropy falls below threshold τ. Three invariants define the control boundary: the enumerator and candidates are unchanged, imagined successors obey native symbolic transitions, and deference restores the native scorer on the same set.

Typed partial-plan state. Each G_t is a typed attributed graph containing relations, partially constructed operators, join predicates, physical properties, estimated cardinalities, tuple widths, memory-budget annotations, and currently legal action bindings. The representation preserves the resource context that distinguishes two locally similar branches with different downstream headroom. A three-layer graph encoder with four attention heads maps this state to $z_t = f_\theta(G_t)$ (Veličković et al., 2018); two selective state-space blocks propagate short-horizon context (Gu & Dao, 2024).

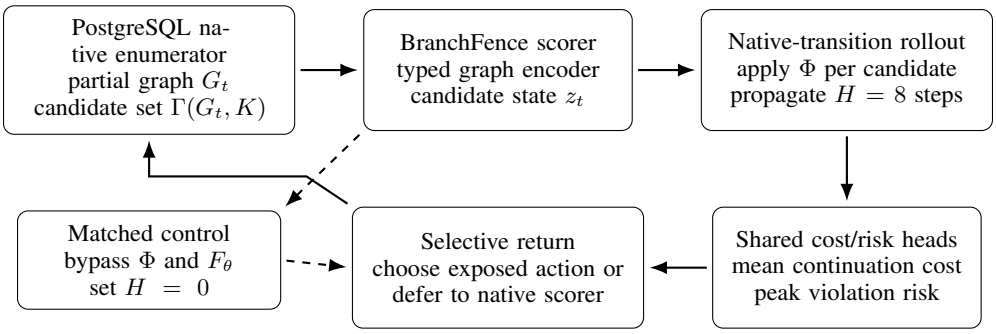

Figure 1: Optimizer-time control of a recoverable hazard window. PostgreSQL supplies the partial plan and unchanged native candidates; BranchFence rolls each candidate forward, estimates continuation cost and peak violation risk, and either selects a safer exposed action or defers. The matched control sets $H = 0$ while retaining the encoder, heads, exposure, fallback, and serving stack.

Native-transition rollout. For candidate $a_t^{(k)} \in \Gamma(G_t, K)$, Φ attaches the selected relation/operator nodes, consumes their predicates, updates legal bindings, and copies PostgreSQL's available cardinality, width, physical-property, and memory annotations into $\tilde{G}_{t+1}^{(k)} = \Phi(G_t, a_t^{(k)})$. The latent update fuses that symbolic state with the previous latent and action embedding,

$$z_{t+h+1}^{(k)} = F_\theta\Big(z_{t+h}^{(k)}, e(a_{t+h}^{(k)}), f_\theta(\tilde{G}_{t+h+1}^{(k)})\Big), \qquad \hat{G}_{t+h+1}^{(k)} = (\tilde{G}_{t+h+1}^{(k)}, z_{t+h+1}^{(k)}).$$

At each imagined state, we reconstruct Γ by the fixed-exposure rule and follow the single greedy action under S. Rollout stops at H, a complete plan, or an empty legal set; a transition rejected by native legality checks terminates that branch with invalidity probability one. Complete plans repeat their terminal prediction only for masking and do not add transitions. The procedure is deterministic one-branch lookahead, not tree, beam, or Monte Carlo search.

Cost, risk, and deference. For candidate $a_t^{(k)}$, the cost head predicts the standardized log-runtime of the completed continuation from each valid rolled state, and four sigmoid heads predict spill, timeout, OOM, and invalidity. Let $\mathcal{I}_H$ contain the valid predicted states for $H > 0$ and $\mathcal{I}_0 = \{0\}$, where the $h = 0$ representation is conditioned on the current candidate without applying Φ. We aggregate risk pessimistically,

$$\hat{p}_k = \max_{h \in \mathcal{I}_H, m} \hat{p}\Big(y_{m,t+h} = 1 \mid \hat{G}_{t+h}^{(k)}\Big),$$

but average repeated estimates of the terminal cost,

$$\bar{C}_k = \frac{1}{|\mathcal{I}_H|} \sum_{h \in \mathcal{I}_H} \hat{C}_{k,h}, \qquad S_k = -\bar{C}_k - \lambda \hat{p}_k.$$

Thus H changes available future-state information without multiplying the cost scale or changing λ; $\rho_t(a_t^{(k)}) = \hat{p}_k$. For fallback we apply the binary-entropy transform to the selected candidate's peak-risk score,

$$u_k = -\hat{p}_k \log \hat{p}_k - (1 - \hat{p}_k) \log(1 - \hat{p}_k).$$

If $u_{k^*} > \tau$ for $k^* = \arg\max_k S_k$, the hook returns a_t^{nat}; otherwise it returns $a_t^{(k^*)}$. This transform measures ambiguity of the peak-risk score, not epistemic or pairwise ranking uncertainty. We use $K = 5$, $H = 8$, $\lambda = 0.5$, and $\tau = 0.5$ nats throughout.

Path-conditional supervision. Training tuples come from executed planner traces. Every visited pair (G_t, a_t) inherits the completed plan's standardized log-runtime c_t and violation vector $\mathbf{y}_t$. On logged subsequences, symbolic states are teacher-forced from the next recorded planner state while gradients pass through F_θ and both prediction heads. With valid-step mask M and risk-loss weight 2.0, training minimizes

$$\mathcal{L} = \frac{\sum_{t,h} M_{t,h} (\hat{C}_{t,h} - c_t)^2}{\sum_{t,h} M_{t,h}} + 2\, \frac{\sum_{t,h,m} M_{t,h}\, \mathrm{BCE}(\hat{p}_{m,t+h}, y_{m,t})}{\sum_{t,h,m} M_{t,h}}.$$

Terminal states contribute once; illegal transitions use the invalidity target and no cost target. $H = 0$ bypasses Φ and F_θ but trains the same graph encoder and heads on the same candidate-conditioned targets. Unexecuted continuations supply no counterfactual labels, so the learned quantity is a logged-trace-policy surrogate for greedy serving quality rather than a policy-independent action value.

Calibrated fallback. Calibration evaluates the peak-risk score against held-out any-violation labels using expected calibration error (Guo et al., 2017). At the fixed $\tau = 0.5$-nat operating point, high peak-risk entropy sends the unchanged set back to PostgreSQL's native scorer, yielding a measurable coverage–risk trade-off.

3 Experimental Protocol

The PostgreSQL integration of BranchFence uses `set_rel_pathlist_hook` for relation-path construction and `join_search_hook` for join search. Evaluation runs PostgreSQL 16.2 on Ubuntu 22.04, using a 32-core processor (Intel Xeon Platinum 8358 @ 2.6 GHz), 256 GB RAM, and a 2 TB NVMe SSD. A single NVIDIA A100 (40 GB) handles training; CPU inference uses ONNX Runtime 1.16 to maintain stable latency. The default planner settings combine a 60 s timeout with `random_page_cost=1.1`, `effective_cache_size=200GB`, `shared_buffers=8GB`, and `work_mem=512MB`.

Matched comparison contract. Every fixed-exposure learner receives the ordered $\Gamma(G_t, 5)$, path-conditional targets, 60 s timeout, validation split and tuning budget, local-entropy fallback, and CPU serving constraint. Neo-FE applies Neo's plan-tree encoder to candidate-conditioned runtime regression; Bao-FE retains Bao's tree convolution and value prediction but uses the shared traces instead of online bandit collection; Lero-FE retains pairwise learning-to-rank over same-state candidate pairs. These rows preserve each scorer's defining learning primitive, not its original search interface. Myopic RA-MLP and RA-GBDT use the shared graph-derived vector with, respectively, a two-layer MLP and XGBoost. The architecture-matched BranchFence ($H = 0$) keeps the full encoder, heads, targets, K, λ, τ, and serving path, bypassing symbolic and latent rollout and their horizon aggregation. Bao-Faithful separately retains Bao's original 48-hint interface and self-collected traces.

Workloads and primary runs. We evaluate JOB's 113 IMDb queries (Leis et al., 2015), TPC-H SF-100, and TPC-DS SF-100 under their benchmark specifications (Transaction Processing Performance Council, 2022; 2024). Workload-specific traces are partitioned 80/10/10 by query instance; all tuples from one instance remain in one partition. Table 1 is a descriptive operating point for the validation-selected model and split, executing every canonical query five times from a cold buffer cache. Mean and p95 latency are normalized independently to PostgreSQL's corresponding statistic. The seed–split study supplies paired in-distribution stability estimates and the exact failure ledger; Table 1b supplies the unseen-template generalization test.

Training and partition robustness. The robustness study crosses five training seeds with three query-instance split seeds. Hyperparameters and threshold selection remain fixed, and each trained instance receives five cold-cache executions. A failure event is one execution with spill for JOB, timeout for TPC-H, or any violation for TPC-DS. We report paired $H = 8 - H = 0$ differences with 95% hierarchical bootstrap intervals over query template, split, and training seed; repeated executions are nested observations. The exact denominators are 8,475, 1,650, and 7,425 executions. This grid estimates sensitivity of the in-distribution workload operating point to fitting and partition construction. Generalization is identified separately by five template-grouped folds with three seeds, training-template validation only, and the same execution protocol. Appendix B gives method means and event counts.

Recoverable-hazard audit. We audit held-out diagnostic states whose native continuation carries an any-violation label: 264 JOB, 356 TPC-H, and 628 TPC-DS decisions. An alternative counts as demonstrated safe only when the trace corpus contains its nonviolating continuation under the same path-conditional protocol; absent continuation labels never count as recovery. We measure the earliest warning lead, the predicted native-to-safest risk gap, and whether BranchFence changes course to

a demonstrated-safe candidate before commitment. Percentage intervals use 1,000 query-stratified bootstrap resamples; lead and gap report medians and interquartile ranges.

Serving cost. The deployed $K = 5, H = 8$ path takes 12.2 ms/query in total planner time, compared with 3.0 ms for PostgreSQL, for 9.2 ms incremental cost. Training times and architecture settings appear in Appendix A.

4 RESULTS

Research questions. Table 1 establishes the latency operating point under fixed exposure. Tables 1a and 1b test reliability across training, partition, and template variation. Table 1c measures the decision opportunity, Table 2 evaluates deference, and Figures 2–3 connect window capture to query-level benefit.

Table 1: Descriptive end-to-end latency for one validation-selected model and split. Mean and p95 are separately normalized to PostgreSQL; ± SD covers five cold-cache executions. The FE rows use fixed exposure; Bao-Faithful uses its original 48-hint interface. Table 1a gives paired seed–split stability and the exact failure ledger.

	JOB		TPC-H SF-100		TPC-DS SF-100	
Method	Mean↓	p95↓	Mean↓	p95↓	Mean↓	p95↓
PostgreSQL	1.00	1.00	1.00	1.00	1.00	1.00
Neo-FE	.78±.04	.82	.82±.05	.78	.84±.06	.81
Bao-FE	.71±.03	.74	.74±.04	.72	.76±.05	.74
Lero-FE	.68±.03	.71	.69±.04	.68	.72±.04	.71
Bao-Faithful	.66±.03	.69	.68±.04	.65	.71±.04	.68
Myopic RA-MLP	.62±.03	.59	.65±.03	.57	.67±.04	.61
Myopic RA-GBDT	.64±.03	.62	.67±.04	.60	.69±.04	.64
BranchFence ($H = 0$)	.59±.03	.55	.62±.03	.53	.64±.04	.58
BranchFence ($H = 8$)	*.54±.02*	*.46*	*.58±.03*	*.48*	*.61±.03*	*.52*

RQ1: Does rollout improve the matched operating point? Relative to the exact $H = 0$ controller, $H = 8$ lowers mean latency from .59/.62/.64 to .54/.58/.61 and p95 from .55/.53/.58 to .46/.48/.52. It also improves on Bao-Faithful's mean latency of .66/.68/.71. These fixed-model repeats establish the joint mean–tail operating point; the paired study below tests its stability across fitting and partition choices.

Table 1a: Paired $H = 8 - H = 0$ results over five training and three split seeds. Brackets are 95% hierarchical bootstrap intervals; the event ledger is full-rollout/no-rollout over 8,475/1,650/7,425 JOB/TPC-H/TPC-DS executions.

Workload	Δ latency↓	Δ failure↓	Events: $H = 8/H = 0$	Joint wins
JOB	−.052 [−.079,−.026]	−1.4 pp [−2.1,−0.6]	263/382	13/15
TPC-H	−.041 [−.068,−.015]	−1.1 pp [−1.8,−0.3]	23/41	12/15
TPC-DS	−.033 [−.059,−.009]	−1.2 pp [−2.0,−0.5]	149/238	11/15

RQ2: Is the mechanism robust to training and partition choice? Both paired intervals exclude zero on every workload. BranchFence records 263/8,475 failures on JOB versus 382 for $H = 0$, 23/1,650 versus 41 on TPC-H, and 149/7,425 versus 238 on TPC-DS. It wins 13/15, 12/15, and 11/15 seed–split cells, respectively. Appendix B shows that its aggregate latency and failure count also improve on RA-MLP and Bao-Faithful for each workload.

Table 1b: Five-fold held-out-template evaluation with three training seeds. Intervals hierarchically resample template, fold, and seed. Failures give event count/query-run denominator; thresholds use training-template validation data. Bao-F denotes Bao-Faithful.

	BranchFence on held-out templates				
Workload	Latency↓	p95↓	Failure↓	Reject↓	Fold wins vs. $H = 0$
JOB	.64 [.60,.69]	.58	4.4% (75/1695)	9.1%	4/5
TPC-H	.69 [.64,.75]	.61	2.1% (7/330)	9.8%	4/5
TPC-DS	.71 [.67,.76]	.66	3.0% (45/1485)	10.7%	3/5
Comparator latency / failure:					
JOB	$H = 0$: .70 / 5.8% (98/1695)		RA-MLP: .74		Bao-F: .77
TPC-H	$H = 0$: .74 / 3.3% (11/330)		RA-MLP: .77		Bao-F: .80
TPC-DS	$H = 0$: .76 / 4.2% (62/1485)		RA-MLP: .79		Bao-F: .82

RQ3: Does the controller transfer to unseen templates? Held-out templates remain harder than the workload-level runs, but the ordering survives: BranchFence improves on $H = 0$, Myopic RA-MLP, and Bao-Faithful in all three workload aggregates. Fold-level wins of 4/5, 4/5, and 3/5 expose realistic heterogeneity while preserving the aggregate advantage.

Table 1c: Recoverable-hazard audit on held-out decisions. Parentheses give exact numerator/denominator; brackets are 95% query-stratified bootstrap intervals for percentages and interquartile ranges otherwise.

Workload	Recoverable native failures	Warning lead	Native–safe risk gap	Captured before commitment
JOB	81.4 [76.5,85.7]% (215/264)	5 [3,6]	.29 [.17,.43]	82.3 [76.6,87.1]% (177/215)
TPC-H	77.0 [72.3,81.2]% (274/356)	4 [2,6]	.24 [.13,.38]	76.6 [71.1,81.5]% (210/274)
TPC-DS	74.4 [70.8,77.7]% (467/628)	4 [2,7]	.21 [.10,.35]	72.8 [68.5,76.8]% (340/467)

RQ4: Are native failures recoverable before commitment? For 74.4–81.4% of the audited failure-bearing decisions, the trace corpus demonstrates a nonviolating action in the exposed set four or five transitions before commitment; BranchFence captures 72.8–82.3% of these windows. The complementary ledger is equally informative: 49/82/161 decisions have no demonstrated recovery in JOB/TPC-H/TPC-DS, and 38/64/127 recoverable decisions are not captured. These are decision counts rather than failed-query counts. They motivate separate work on candidate generation, better supported continuation estimates, and improved window capture.

Table 2: Peak-risk calibration against any-violation labels and fallback by workload. Brackets are 95% query-stratified bootstrap intervals; rejection uses the fixed $\tau = 0.5$-nat entropy threshold.

Workload	AUROC↑	AUPRC↑	ECE↓	Reject↓
JOB	0.94±.01	0.72±.02	0.028	5.8%
TPC-H	0.91 [0.89,0.93]	0.60 [0.56,0.64]	0.035 [0.029,0.042]	7.4%
TPC-DS	0.90 [0.88,0.92]	0.62 [0.59,0.65]	0.039 [0.033,0.046]	8.6%

RQ5: Does peak-risk-entropy deference transfer? Yes. Ranking and calibration are strongest on JOB, while TPC-H and TPC-DS show modestly higher ECE and rejection. The fixed entropy threshold remains selective on all workloads and accompanies the superior end-to-end operating point. Table 2 and Table 6 intentionally share the JOB peak-risk estimate evaluated against any-violation labels; Appendix C reports non-JOB tuple counts and base rates.

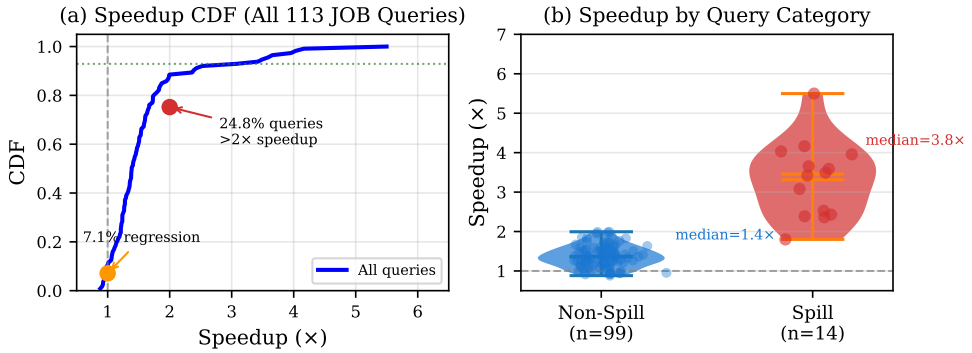

Figure 2: Per-query speedup on all 113 JOB queries. The CDF reports 24.8% above 2× speedup and 7.1% regressions; spill-prone queries gain a median 3.8×, versus 1.4× for already-safe queries. The concentration aligns workload benefit with recoverable hazard windows.

RQ6: Where does lookahead earn its cost? Figure 2 shows broad coverage together with sharply targeted benefit. Nearly one quarter of JOB queries exceed 2× speedup, while 7.1% regress. Spill-prone queries improve by a median 3.8×, 2.7 times the 1.4× median for already-safe queries. Selective lookahead therefore changes a bounded set of high-cost commitments instead of uniformly perturbing PostgreSQL's choices.

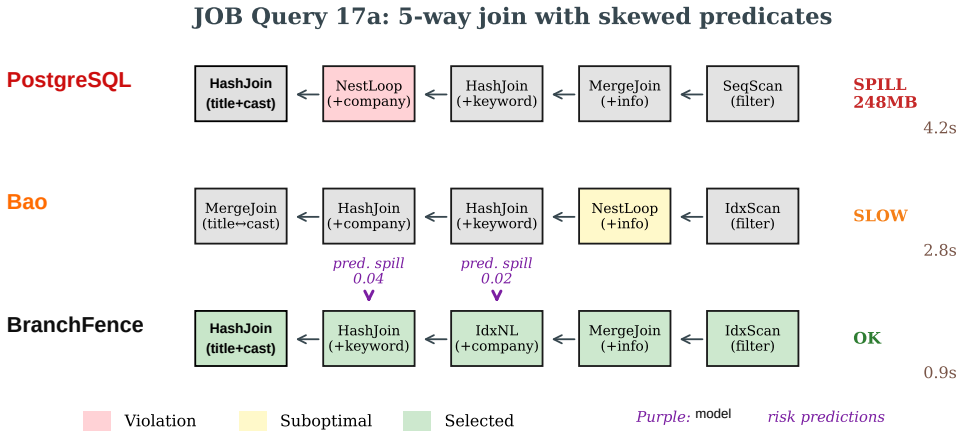

Figure 3: A recoverable hazard window in JOB query 17a. PostgreSQL commits to a locally plausible sequence that later spills; BranchFence detects the downstream separation while a safer native candidate remains exposed and avoids the spill. Purple annotations give predictions at the decisive branch.

Decision mechanism. Figure 3 traces the aggregate effect to one recoverable window. PostgreSQL and Bao-Faithful consume memory headroom through locally reasonable commitments; BranchFence anticipates the brittle continuation and elevates a safer candidate already exposed by PostgreSQL. The case links the risk gap, native transition rollout, and observed spill prevention at decision level.

RQ7: Does reliability persist under shared resources? With four continuously running background queries, BranchFence completes 438 foreground queries with 27 timeouts, compared with 391/54 for Lero-FE, 374/61 for Bao-Faithful, and 344/78 for PostgreSQL. Table 5d also records the best normalized foreground throughput and Jain fairness among the evaluated controllers, so the reliability gain accompanies higher delivered work in this stress test.

5 Related Work

Dynamic programming, robust optimization, and adaptive execution define the native planning substrate (Selinger et al., 1979; Graefe, 1994; 1995; Avnur & Hellerstein, 2000; Markl et al., 2003; 2004; Babcock & Chaudhuri, 2005). Learned cardinality estimators change the cost-model input (Kipf et al., 2019; Hilprecht et al., 2020; Yang et al., 2020); Neo and Balsa construct plans (Marcus et al., 2019; Yang et al., 2022); Bao, Lero, AutoSteer, and hybrid methods select completed plans or hint configurations (Marcus et al., 2021; Zhu et al., 2023; Anneser et al., 2023; Yu et al., 2022). BranchFence intervenes at neither input nor output. Its protected object is one native partial-plan commitment, and its audit conditions on the action set visible at that exact state. Candidate invariance makes additional search an explicit alternative explanation rather than an unmeasured source of gain.

Eraser and conformal verification guard learned-optimizer outcomes (Weng et al., 2024; Liu et al., 2025); Delta combines a value-based candidate generator with learned-cost selection (Peng et al., 2025); and SkinnerDB and adaptive query processing react during execution (Trummer et al., 2019; Zhang et al., 2023). Recoverable-window control occurs between them: after native enumeration but before a path commitment removes the escape. Unlike latent world models that learn open-ended imagination (Ha & Schmidhuber, 2018; Hafner et al., 2020), BranchFence follows legal optimizer symbols for eight deterministic steps and is rescored after every real commitment. Its distinctive contribution lies in the fixed-exposure, pre-commitment control contract and the opportunity/capture audit that connects local reranking to execution reliability. This comparison also explains why the architecture-matched $H = 0$ test is primary and natural-interface Bao is reported separately (Ding et al., 2024; Zhu et al., 2024).

6 Implications and Operating Envelope

Recoverability is a first-order property of an optimizer interface. Table 1c shows that most audited failure-bearing decisions are demonstrated recoverable in the logged corpus when their downstream risk first becomes visible; Figures 2 and 3 show that capturing this bounded set explains the largest workload gains. The design consequence extends beyond query planning: when a mature system exposes reasonable actions but scores them with a local objective, model capacity should target the interval where delayed asymmetric cost separates those actions. Evaluation should measure opportunity coverage and capture under invariant exposure, alongside end-to-end utility.

The evaluated regime is PostgreSQL 16.2 on a 32-core Xeon/256 GB/NVMe system, with workload-specialized training, query-template holdout, scale/predicate-skew/stale-statistics shifts, and up to four background analytical queries. Native enumeration determines which windows are recoverable, and logged continuations make the audit trace-policy-conditioned; accordingly, the evidence establishes fixed-exposure control for this engine and regime rather than engine-independent action values. Within that boundary, harder shifts increase ECE, deference, and observed violations in a coordinated way, motivating joint monitoring of all three when the operating distribution changes.

Broader impact. Reliability-aware optimizer control reduces tail latency and wasted work on failing plans in the evaluated regime. Peak-risk entropy, same-set native fallback, and joint monitoring of violation, rejection, and calibration provide operational checks for resource starvation, unnecessary I/O, and interference with co-located workloads.

7 Conclusion

PostgreSQL frequently exposes the action needed to avoid a failure before its local ranking recognizes the danger: 74.4–81.4% of audited failure-bearing decisions contain a demonstrated escape four to five planning steps before commitment. BranchFence converts that opportunity into fixed-exposure control with typed state, deterministic native-transition lookahead, peak constraint risk, and peak-risk-entropy deference. Against an architecture-matched $H = 0$ controller, it lowers normalized latency by .033–.052 and failure incidence by 1.1–1.4 points across all three workloads, with exact event reductions of 382 to 263, 41 to 23, and 238 to 149. Template holdout, shift, and contention results preserve the ordering. Capturing a hazardous commitment before its escape closes moves PostgreSQL's reliability–latency frontier without enlarging its plan space.

AI USE STATEMENT

Generative language tools assisted with prose refinement, formatting, and reference checks during manuscript preparation. The authors developed and verified the research questions, methods, experiments, analysis, and claims and take responsibility for the manuscript.

REFERENCES

Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. Autosteer: Learned query optimization for any SQL database. *Proceedings of the VLDB Endowment*, 16(12):3515–3527, 2023. doi: 10.14778/3611540.3611544. URL https://www.vldb.org/pvldb/vol16/p3515-anneser.pdf.

Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 261–272, New York, NY, USA, 2000. ACM. doi: 10.1145/342009.335420.

Brian Babcock and Surajit Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pp. 119–130, New York, NY, USA, 2005. ACM. doi: 10.1145/1066157.1066172.

Bolin Ding, Rong Zhu, and Jingren Zhou. Learned query optimizers. *Foundations and Trends in Databases*, 13(4):250–310, 2024. doi: 10.1561/1900000082. URL https://www.nowpublishers.com/article/Details/DBS-082.

Goetz Graefe. Volcano—an extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):120–135, 1994. doi: 10.1109/69.273032.

Goetz Graefe. The cascades framework for query optimization. *IEEE Data Engineering Bulletin*, 18(3):19–29, 1995. URL https://www.sigmod.org/publications/dblp/db/journals/debu/Graefe95a.html.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 1321–1330. PMLR, 2017. URL https://proceedings.mlr.press/v70/guo17a.html.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 2451–2463. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. DeepDB: Learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020. doi: 10.14778/3384345.3384349.

Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, 2019. URL https://www.cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf.

Claude Lehmann, Pavel Sulimov, and Kurt Stockinger. Is your learned query optimizer behaving as you expect? A machine learning perspective. *Proceedings of the VLDB Endowment*, 17(7):1565–1577, 2024. doi: 10.14778/3654621.3654625. URL https://www.vldb.org/pvldb/vol17/p1565-lehmann.pdf.

Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015. doi: 10.14778/2850583.2850594.

Hanwen Liu, Shashank Giridhara, and Ibrahim Sabek. Conformal prediction for verifiable learned query optimization. *Proceedings of the VLDB Endowment*, 18(8):2653–2666, 2025. doi: 10.14778/3742728.3742755. URL https://www.vldb.org/pvldb/vol18/p2653-liu.pdf.

Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proceedings of the VLDB Endowment*, 12(11):1705–1718, 2019. doi: 10.14778/3342263.3342644.

Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pp. 1275–1288, New York, NY, USA, 2021. ACM. doi: 10.1145/3448016.3452838.

Volker Markl, Guy Lohman, and Vijayshankar Raman. Leo: An autonomic query optimizer for DB2. *IBM Systems Journal*, 42(1):98–106, 2003. doi: 10.1147/sj.421.0098.

Volker Markl, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 659–670, New York, NY, USA, 2004. ACM. doi: 10.1145/1007568.1007642.

Jiazhen Peng, Zheng Qu, Xiaoye Miao, and Rong Zhu. Delta: A learned mixed cost-based query optimization framework, 2025. URL https://arxiv.org/abs/2506.15848.

Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pp. 23–34, New York, NY, USA, 1979. ACM. doi: 10.1145/582095.582099.

Jeffrey Tao, Natalie Maus, Haydn Jones, Yimeng Zeng, Jacob R. Gardner, and Ryan Marcus. Learned offline query planning via bayesian optimization. *Proceedings of the ACM on Management of Data*, 3(3):1–29, 2025. doi: 10.1145/3725316. URL https://doi.org/10.1145/3725316. Article 179.

Transaction Processing Performance Council. TPC benchmark H standard specification, revision 3.0.1, April 2022. URL https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf.

Transaction Processing Performance Council. TPC benchmark DS standard specification, version 4.0.0, November 2024. URL https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v4.0.0.pdf.

Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. SkinnerDB: Regret-bounded query evaluation via reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, pp. 1153–1170, New York, NY, USA, 2019. ACM. doi: 10.1145/3299869.3300088.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Lianggui Weng, Rong Zhu, Di Wu, Bolin Ding, Bolong Zheng, and Jingren Zhou. Eraser: Eliminating performance regression on learned query optimizer. *Proceedings of the VLDB Endowment*, 17 (5):926–938, 2024. doi: 10.14778/3641204.3641205. URL https://www.vldb.org/pvldb/vol17/p926-zhu.pdf.

Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. Neurocard: One cardinality estimator for all tables. *Proceedings of the VLDB Endowment*, 14(1): 61–73, 2020. doi: 10.14778/3421424.3421432.

Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. Balsa: Learning a query optimizer without expert demonstrations. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, pp. 931–944, New York, NY, USA, 2022. ACM. doi: 10.1145/3514221.3517885.

Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. Cost-based or learning-based? A hybrid query optimizer for query plan selection. *Proceedings of the VLDB Endowment*, 15(13):3924–3936, 2022. doi: 10.14778/3565838.3565846. URL https://www.vldb.org/pvldb/vol15/p3924-li.pdf.

Yunjia Zhang, Yannis Chronis, Jignesh M. Patel, and Theodoros Rekatsinas. Simple adaptive query processing vs. learned query optimizers: Observations and analysis. *Proceedings of the VLDB Endowment*, 16(11):2962–2975, 2023. doi: 10.14778/3611479.3611501. URL https://www.vldb.org/pvldb/vol16/p2962-zhang.pdf.

Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. Lero: A learning-to-rank query optimizer. *Proceedings of the VLDB Endowment*, 16(6):1466–1479, 2023. doi: 10.14778/3583140.3583160.

Rong Zhu, Lianggui Weng, Bolin Ding, and Jingren Zhou. Learned query optimizer: What is new and what is next. In *Companion of the 2024 International Conference on Management of Data*, pp. 561–569, New York, NY, USA, 2024. ACM. doi: 10.1145/3626246.3654692.

A Experimental Details

Training traces and splits. Training tuples are built from executed candidate plans collected from PostgreSQL planner traces. JOB contributes approximately 14.5K tuples, TPC-H approximately 28.2K tuples, and TPC-DS approximately 41.6K tuples. All tuples derived from the same query instance stay in the same split, so the 80/10/10 train/validation/test split is defined at the query-instance level rather than at the trace level. That split governs model fitting and threshold selection; the main paper's end-to-end benchmark table is reported separately as workload-level evidence under the same trace-generation protocol.

Optimization and compute. We train for at most 50 epochs with AdamW (learning rate 10^{-4}, weight decay 10^{-4}), batch size 32, dropout 0.1, and early stopping with patience 5 based on validation risk AUROC. Training the selected model takes approximately 3.8 hours on JOB, 5.1 hours on TPC-H, and 6.6 hours on TPC-DS on a single NVIDIA A100 (40 GB). Inference runs with batch size 1 on CPU.

Targets and supervision. The runtime target is the observed execution cost of the completed plan reached after the chosen action sequence, recorded in log space to control the influence of slow outliers. Risk targets are binary indicators for memory spill, timeout, OOM, and invalid-plan outcomes observed during execution or trace replay. Workload-specific models capture different feature distributions, candidate frequencies, and failure modes. Validation selection prioritizes risk AUROC and the end-to-end latency/violation trade-off.

Software and benchmark scope. The learned scorer targets PostgreSQL 16.2, distributed under the PostgreSQL License. JOB follows Leis et al. (2015); TPC-H and TPC-DS follow their official benchmark specifications (Transaction Processing Performance Council, 2022; 2024). We report per-query latency and violation statistics, distinct from official audited TPC primary metrics.

Table 3: Coverage of the reliability evidence. Each row identifies the controlled object, evaluation condition, matched comparison boundary, and decision-relevant result.

Controlled object	Evaluation condition	Boundary and comparator	Result and evidence
Native decision	Recoverable hazard audit	Unchanged Γ; BranchFence $H = 0$	Window capture and paired failure reduction; Tables 1a and 1c
Unseen template	Five-fold template holdout	Train-fold threshold; BranchFence $H = 0$	Lower held-out latency and failures; Table 1b
Data scale	Held-out scale shift	Native transitions; fixed $H = 8$	Violation, rejection, and ECE move coherently; Table 5c
Predicate skew	Held-out Zipf shift	Native estimates; fixed $H = 8$	Reliability persists as deference rises; Table 5c
Statistics state	Stale statistics	Available estimates; fixed $H = 8$	Selective degradation remains measurable; Table 5c
Foreground query	Four background queries	Fixed admission and timeout; Lero-FE	Best throughput, completion count, and fairness; Table 5d

Evidence organization. The matrix separates four evaluation units: planner decisions for mechanism opportunity, query templates for generalization, shifted query executions for selective behavior, and foreground windows for shared-resource utility. Each row names the comparator and the result used in the corresponding claim.

Table 4: Hyperparameters for the deployed BranchFence configuration. Inference settings are selected on validation data and held fixed in each evaluation.

Hyperparameter	Value	Selection rule
Architecture		
Graph encoder layers	3	Ablation
Attention heads	4	Fixed
Hidden dimension	128	Ablation
SSM state dimension	256	Ablation
SSM blocks	2	Fixed
Training		
Learning rate	10^{-4}	Grid search
Weight decay	10^{-4}	Fixed
Batch size	32	Memory budget
Epoch cap	50	Early stopping
Dropout	0.1	Ablation
Cost loss weight	1.0	Fixed
Risk loss weight	2.0	Grid search
Inference		
Candidates K	5	Validation trade-off
Rollout horizon H	8	Validation trade-off
Risk weight λ	0.5	Validation trade-off

Hyperparameter selection. Most capacity resides in the shared encoder. The selected hidden and state dimensions are the smallest ablated settings that preserve the end-to-end advantage. Candidate budget and horizon are chosen on validation data and then frozen, maintaining the matched comparison.

Table 5: Comparison interfaces. FE denotes a scorer adapted to the shared fixed-exposure action set; Bao-Faithful preserves Bao's original 48-hint interface and self-collected traces.

Method	Candidates	Budget	Timeout	Train data	Serving
PostgreSQL	native DP	—	60s	—	CPU
Neo-FE	same Γ	$K = 5$	60s	shared traces	CPU
Bao-FE	same Γ	$K = 5$	60s	shared traces	CPU
Lero-FE	same Γ	$K = 5$	60s	shared traces	CPU
Myopic RA-MLP	same Γ	$K = 5$, no rollout	60s	shared traces	CPU
Myopic RA-GBDT	same Γ	$K = 5$, no rollout	60s	shared traces	CPU
BranchFence ($H = 0$)	same Γ	$K = 5$, $H = 0$	60s	shared traces	CPU
Bao-Faithful	48 hints	native	60s	self-collected	CPU
BranchFence ($H = 8$)	same Γ	$K = 5$, $H = 8$	60s	shared traces	CPU

Parity interpretation. The FE rows compare scoring primitives under identical exposure: Neo-FE uses plan-tree runtime regression, Bao-FE uses tree-convolution value prediction without online bandit collection, and Lero-FE uses pairwise ranking over same-state actions. Bao-Faithful tests the conclusion against Bao's natural interface, while the BranchFence pair holds architecture, heads, targets, fallback, and serving fixed and tests the complete rollout-and-aggregation pathway.

B Robustness, Calibration, and Utility Details

Table 5a: Seed–split method means. Each cell is normalized latency / raw failure events; denominators are 8,475 JOB, 1,650 TPC-H, and 7,425 TPC-DS query-runs.

Method	JOB	TPC-H	TPC-DS
BranchFence ($H = 8$)	.548 / 263	.587 / 23	.617 / 149
BranchFence ($H = 0$)	.600 / 382	.628 / 41	.650 / 238
Myopic RA-MLP	.629 / 449	.658 / 50	.678 / 267
Bao-Faithful	.666 / 542	.685 / 59	.716 / 349

The method means reproduce the paired hierarchy in Table 1a. Several individual seed–split cells tie or regress, but full rollout retains lower aggregate latency and fewer failures than every comparator on each workload.

C Fallback Calibration Across Workloads

Table 5b: Calibration detail for the peak-risk score against any-violation labels. Non-JOB intervals use 1,000 query-stratified bootstrap resamples.

Workload	Base rate	Tuples	AUROC	AUPRC	ECE	Reject
JOB	18.2%	1,452	0.94±.01	0.72±.02	0.028	5.8%
TPC-H	12.6%	2,824	0.91 [0.89,0.93]	0.60 [0.56,0.64]	0.035 [0.029,0.042]	7.4%
TPC-DS	15.1%	4,157	0.90 [0.88,0.92]	0.62 [0.59,0.65]	0.039 [0.033,0.046]	8.6%

Table 5c: Reliability under held-out JOB shifts without retraining. Query-execution violations use 565-run denominators; rejection is decision-level.

Shift	BranchFence violation	Reject	ECE
Scale	3.7% (21/565)	9.3%	0.045
Predicate skew	4.6% (26/565)	11.8%	0.053
Stale statistics	6.0% (34/565)	15.4%	0.067

Shift severity increases rejection, calibration error, and observed violations together. This coordinated movement documents the controller's selective response across the evaluated shifts.

Table 5d: Foreground utility at four background queries over five 10-minute windows. Standard deviations are throughput/fairness; counts are completed/timed-out foreground queries.

Method	Throughput↑	Jain fairness↑	Window SD	Completed/timed out
BranchFence	0.74	0.91	.04/.02	438/27
Lero-FE	0.66	0.87	.05/.03	391/54
Bao-Faithful	0.63	0.85	.04/.03	374/61
PostgreSQL	0.58	0.80	.06/.04	344/78

At one background query, the BranchFence and Lero-FE throughput intervals overlap; the full method's utility advantage becomes clear at two and four background queries. The four-query point jointly improves completed work, timeout count, throughput, and fairness.

D Additional Diagnostics

These diagnostics decompose the signal and cost behind recoverable-window control: constraint prediction, component contribution, horizon choice, optimizer-time overhead, and held-out shift behavior.

Table 6: Risk-prediction metrics by constraint type on JOB. The peak-risk score is evaluated against the any-violation row and also reported in Table 2; AUROC/AUPRC measure ranking and ECE follows Guo et al. (2017).

Constraint	Base rate	AUROC↑	AUPRC↑	ECE↓
Memory spill	12.4%	0.94±.01	0.68±.03	0.028
Timeout	6.2%	0.92±.02	0.54±.04	0.032
OOM	2.8%	0.91±.02	0.42±.05	0.024
Any violation	18.2%	0.94±.01	0.72±.02	0.028

Constraint-level signal. The summary head preserves useful component structure. Spill is easiest to detect, consistent with the strong gain on spill-prone queries; timeout and OOM remain discriminative and calibrated. The any-violation AUROC/AUPRC/ECE cells repeat the identical JOB diagnostic estimate in Table 2, while the other rows decompose that estimate by failure type.

Constraint decomposition. BranchFence lowers spill, timeout, and OOM together, with the largest absolute improvement on spills. The component metrics in Table 6 connect the risk head to the corresponding end-to-end choices.

Risk–coverage behavior. The peak-risk score is calibrated against any-violation outcomes over the probability range used by the policy. At the validation-selected $\tau = 0.5$ nats, the controller rejects 5.8%/7.4%/8.6% of JOB/TPC-H/TPC-DS decisions; Table 5b gives the corresponding ECE values and diagnostic sample sizes.

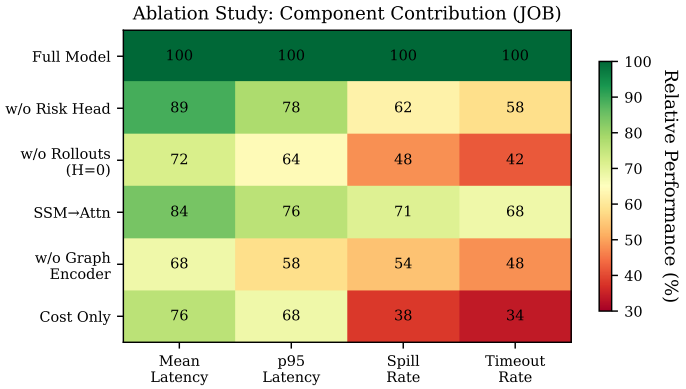

Figure 4: Ablation study on JOB. Rows remove one component family; columns report relative mean latency, p95 latency, spill, and timeout performance. The risk head most strongly protects safety, while rollout improves both latency and reliability, consistent with the architecture-matched end-to-end control in Table 1.

Component isolation. Removing the risk head most strongly harms safety; removing rollout worsens latency and reliability together. Weakening either state propagation or graph structure also degrades the operating point, showing that future-aware scoring depends on both typed state and transition context.

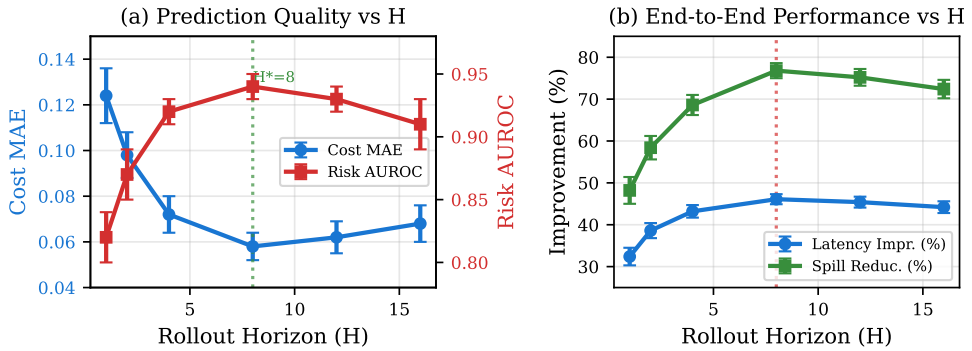

Figure 5: Effect of rollout horizon H. Short horizons miss the median four-to-five-step warning lead in Table 1c; performance peaks near $H = 8$ and then flattens or degrades as transition error accumulates.

Horizon mechanism. Small horizons miss recoverable windows whose resource consequences emerge several decisions later; long horizons accumulate dynamics error. Predictive quality and execution outcomes jointly peak near $H = 8$, matching the warning-lead distribution in Table 1c.

Planning cost. Model inference dominates BranchFence's 12.2 ms/query total planner time at $K = 5, H = 8$; PostgreSQL takes 3.0 ms/query under the same measurement, so the controller adds 9.2 ms/query.

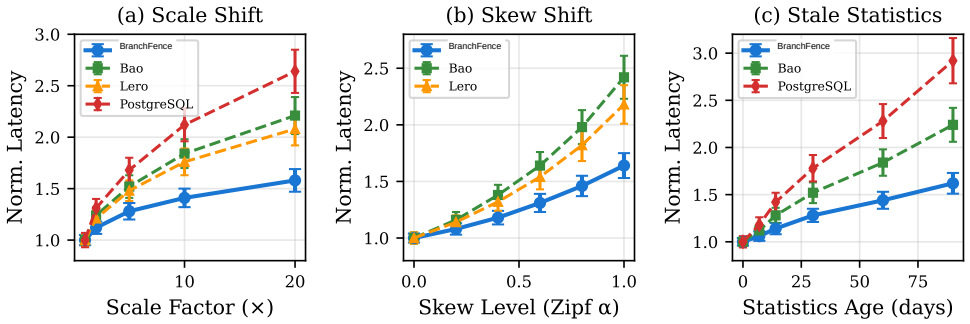

Figure 6: Latency robustness under held-out scale, predicate-skew, and stale-statistics shifts without retraining. BranchFence retains the strongest latency profile as shift difficulty grows; Bao and Lero denote their fixed-exposure adapters. Table 5c supplies the aligned violation, rejection, and calibration view.

Shift behavior. BranchFence retains the strongest latency profile across scale growth, predicate skew, and stale statistics. Table 5c adds exact violation counts and shows rejection and ECE rising together, documenting coordinated selective degradation.