Learning Reusable Partial-Order Skills from Agent Traces

Anonymous authors
Paper under double-blind review

Abstract

Agent traces are sequential, but the procedures they record may allow alternative action orders, repetition, and repair. We introduce Reusable Partial-Order Skills (RPOS), a probabilistic model that jointly infers skill boundaries, assignments, and shared precedence constraints from action-labelled traces. The action dictionary, number of skills, are specified in advance; skill decompositions and partial orders are inferred. Each skill instance tracks the validity of action outputs, modelling repetition and repair without adding cycles to its shared graph. Semi-Markov dynamic programming sums exactly over admissible segmentations and skill assignments to evaluate structural proposals. Experiments on synthetic data show improved recovery and prediction in the completed runs. On coverage-controlled TaskBench simulations, RPOS predicts unseen valid action orderings better than the tested total-order and sequence models. On τ^3-Retail, RPOS learns an explicit library of partial-order skills and transitions between them, making within-skill precedence relations and between-skill transitions transparent. Finally, on 120 paired synthetic Retail tasks, an executor combining the learned library before evaluation reduces online token use by 86.97% relative to a stepwise agent while meeting a prespecified non-inferiority margin for strict task success.

1 Introduction

Each agent trace records one action sequence, not every ordering a procedure allows. Independent actions may appear in different orders, and earlier actions may be repeated when their results need revision. Treating every observed ordering as a dependency therefore obscures this flexibility; representing each repetition as a separate step obscures what is shared across executions.

Explicit procedural structure can support parallel tool calls and reduce redundant model prompting (Kim et al., 2024; Xu et al., 2023). Learning that structure from traces requires identifying both the procedures and their boundaries. CompILE and BUDS discover reusable segments (Kipf et al., 2019; Zhu et al., 2022), whereas BPOP infers precedence over fixed action occurrences (Li et al., 2026a). We study the joint problem: discovering skill instances and the partial orders shared by them.

We introduce *Reusable Partial Order Skills* (RPOS), a probabilistic model of skill structure and execution. RPOS represents each trace as a sequence of contiguous skill instances. Each instance uses a reusable partial order whose nodes are canonical procedural action (CPA) labels. Its validity state records which action outputs remain usable. Repeating an upstream action can invalidate downstream outputs, allowing repair without introducing graph cycles. The action dictionary and model settings are specified in advance, but skill boundaries, assignments, ordering constraints, and transitions are learned. Figure 1 distinguishes skill reuse from within-instance repetition: two instances reuse H_1 with different orders of B and C, while the H_2 instance repeats actions.

Inference is challenging because segmentation and skill structure are coupled in the posterior: a proposed graph may fit a trace under a different skill path than the one currently sampled. We therefore use a semi-Markov recursion to sum over paths when scoring structural proposals (Sarawagi & Cohen, 2004; Yu, 2010). The same recursion supports exact conditional path sampling. Each path-marginal structural update is followed by a conditional path redraw, forming a partially collapsed transition. A differentiable relaxation provides an alternative point-estimation approach.

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.

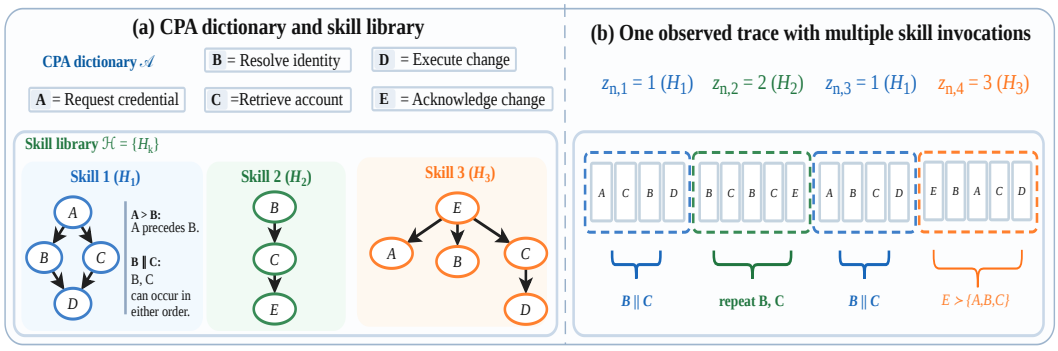


Figure 1: **From sequential traces to reusable partial-order skills. (a)** A fixed CPA dictionary and a library of reusable partial-order skills; for example, $B \parallel C$ in H_1. **(b)** One observed trace with four skill invocations, illustrating repeated skill use, alternative valid orders, and repeated actions.

Contributions. **(i) Representation:** a recurrent segmental model that jointly learns skill boundaries, assignments, transitions, and reusable partial orders with repetition and repair. **(ii) Inference:** exact conditional path inference and path-marginal structural updates using semi-Markov dynamic programming. **(iii) Evaluation:** Studies of joint recovery and prediction on unseen skill compositions and action orderings, plus a Retail execution study combining the learned library.

2 Related Work

Bayesian partial-order inference. Latent-coordinate Bayesian models infer partial orders from noisy rankings (Nicholls et al., 2025). BPOP models agent traces with a frontier-based likelihood and represents repeated actions as distinct occurrences; its differentiable extension relaxes precedence and frontier constraints (Li et al., 2026a;c). Hierarchical partial-order models share structure across groups and allow latent clustering (Li et al., 2026b). RPOS instead jointly infers a library of partial-order skills over CPA labels, instance boundaries and assignments within each trace, and transitions between instances. Repeated occurrences of an action reuse the same node within a skill, while instances of that skill share its partial-order template. The action dictionary, action supports, and number of skill types are specified in advance.

Process and task-graph discovery. Partially ordered macro-operator learning, workflow mining, and frequent-episode discovery learn procedural structure and recurring local patterns (Mooney, 1988; van der Aalst et al., 2004; Leemans & van der Aalst, 2014). Partial-order process mining also represents concurrency and uncertainty in observed event order (Leemans et al., 2023). POWL combines partial orders with choice and loops in hierarchical workflow models, with automated discovery methods (Kourani & van Zelst, 2023; Kourani et al., 2025). Neural Task Graphs support transfer through graph-based task representations (Huang et al., 2019), while differentiable task-graph learning estimates procedural dependencies from action sequences (Seminara et al., 2024). RPOS focuses on a joint probabilistic decomposition of traces: local orders form a shared skill library, while latent boundaries, assignments, and transitions explain how its instances compose.

Reusable skills and agent workflows. CompILE learns latent boundaries and reusable codes, while BUDS discovers sensorimotor skills from unsegmented demonstrations (Kipf et al., 2019; Zhu et al., 2022); DreamCoder learns program abstractions shared across tasks (Ellis et al., 2021). For agents, Agent Workflow Memory induces reusable workflows, whereas Agent Skill Induction (ASI) induces and tests executable skill programs (Wang et al., 2025b;a). TraceCompiler uses argument-provenance evidence to compile clusters of agent traces into executable workflows (Yadouni & Li, 2026). RPOS fits labelled sequences rather than inferring executable argument bindings.

Segmental and marginalized inference. Semi-Markov conditional random fields and hidden semi-Markov models provide dynamic programs over labelled, variable-length segments (Sarawagi & Cohen, 2004; Yu, 2010; Johnson & Willsky, 2013). Partially collapsed MCMC integrates out

selected latent variables, with validity depending on the ordering of marginal and conditional updates (van Dyk & Park, 2008; van Dyk & Jiao, 2015). RPOS specializes established segmental inference to its recurrent partial-order likelihood and uses the resulting path sum to evaluate structural proposals. In our sampler, each marginal MH attempt, accepted or rejected, is followed by an FFBS path redraw before subsequent path-conditioned updates.

3 A Segmental Model of Reusable Partial-Order Skills

Let $\mathcal{X} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$ contain action-labelled traces, where $\boldsymbol{x}^{(n)} = (x_1^{(n)}, \ldots, x_{J_n}^{(n)}) \in \mathcal{A}^{J_n}$ and J_n counts action occurrences. Labels may repeat. We model each trace as contiguous, variable-length skill instances, each assigned to one of K shared partial-order skills. Instance boundaries, assignments, templates, and transitions between instances are latent.

Canonical actions and skills For Retail, the CPA protocol of Li et al. (2026d) converts raw traces into labelled action occurrences. These annotations supply neither skill boundaries nor dependency labels. Skill k has a fixed action support $\mathcal{A}_k \subseteq \mathcal{A}$, with one template node per supported label; repeated occurrences reuse that node. In Retail, $\mathcal{A}_k = \mathcal{A}$ for every skill. An instance need not execute every supported action. The vocabulary, supports, and K are fixed; instance boundaries, assignments, and ordering constraints are inferred. Appendix B describes sequence preparation.

3.1 Reusable partial-order skills

Following the latent-variable partial-order construction of Nicholls et al. (2025), we associate each skill k with a utility matrix $U_k \in \mathbb{R}^{m_k \times d}$, where $m_k = |A_k|$ and rows are indexed by CPA labels in A_k. The rows induce a strict partial order $H_k = h(U_k)$ through coordinatewise dominance:

$$r \succ_{H_k} s \quad \Longleftrightarrow \quad U_{k,rj} > U_{k,sj} \quad \text{for all } j = 1, \ldots, d. \tag{1}$$

Here, $r \succ_{H_k} s$ means that action r precedes action s. Two distinct actions are incomparable when neither utility vector strictly dominates the other in every coordinate. Each utility row has prior

$$\boldsymbol{U}_{k,r} \mid \rho \sim \mathcal{N}_d(\mathbf{0}, \Sigma_\rho), \qquad \Sigma_\rho = (1-\rho) I_d + \rho \mathbf{1}\mathbf{1}^\top. \tag{2}$$

The partial order H_k specifies each action's predecessors; the current validity of their outputs determines its frontier compatibility. The partial-order skill remains fixed during an instance, while the validity state changes . When $\epsilon > 0$, uniform noise allows actions outside the current frontier. At fixed remaining parameters, utility matrices that induce the same partial order have identical likelihoods, so inference concerns shared orders rather than unique utility coordinates.

3.2 Recurrent execution within a skill instance

Repeated actions update an instance's execution state without adding nodes or cycles to its shared partial-order template (Figure 5). A contiguous block $x_{a:b}^{(n)} = (x_a^{(n)}, \ldots, x_b^{(n)})$ assigned to skill k forms an instance of length $T = b - a + 1$. For $u = 1, \ldots, T$, let $y_u = x_{a+u-1}^{(n)}$ denote the observed CPA label at position u within the instance.

Validity state. $\boldsymbol{q}_u = (q_u(r))_{r \in \mathcal{A}_k} \in [0,1]^{m_k}$, starting from $\boldsymbol{q}_0 = \mathbf{0}$. The value $q_u(r)$ represents the validity of action r's output under the model, from zero (no valid output) to one (fully valid). The state is computed from the action history, not observed from the environment or sampled separately.

Write $r \succ_{H_k} s$ when r precedes s, let $\mathbb{I}[\cdot]$ be the indicator function. After executing y_u, the state is

$$q_u(r) = \mathbb{I}[r = y_u] + \mathbb{I}[r \neq y_u]\, q_{u-1}(r)\, [1 - \mathbb{I}[y_u \succ_{H_k} r]\, \mathrm{sigmoid}(\omega)]\,. \tag{3}$$

, where $\mathrm{sigmoid}(\omega) = (1+\exp(-\omega))^{-1}$ and $\omega \in \mathbb{R}$ controls the strength of invalidation. Executing an action sets its own validity to one and multiplies each downstream action's validity by $1 - \mathrm{sigmoid}(\omega)$.

Next-action distribution. For each action $r \in \mathcal{A}_k$, let $\mathrm{Pred}_{H_k}(r) = \{s \in \mathcal{A}_k : s \succ_{H_k} r\}$ include all predecessors. Its frontier compatibility is $F_u(r) = \prod_{s \in \mathrm{Pred}_{H_k}(r)} q_{u-1}(s)$. The progress score $Q_u(r)$ measures how many actions r precedes, discounting those whose outputs are already valid. The disruption score $C_u^{\mathrm{back}}(r)$ measures how much downstream validity executing r removes:

$$Q_u(r) = \log\left[1 + \sum_{s:\, r \succ_{H_k} s} \big(1 - q_{u-1}(s)\big)\right], \qquad C_u^{\mathrm{back}}(r) = \mathrm{sigmoid}(\omega) \sum_{s:\, r \succ_{H_k} s} q_{u-1}(s).$$

For nonnegative β, λ_{rep}, and λ_{back}, the unnormalized choice weight is

$$W_u(r) = F_u(r) \exp\big\{\beta Q_u(r) - \lambda_{\mathrm{rep}} q_{u-1}(r) - \lambda_{\mathrm{back}} C_u^{\mathrm{back}}(r)\big\}. \tag{4}$$

The three terms respectively reward progress (Q_u), penalize repeating an action whose output remains valid (q_{u-1}), and penalize disruption to downstream outputs (C_u^{back}). We collect the execution parameters as $\vartheta = (\beta, \omega, \lambda_{\mathrm{rep}}, \lambda_{\mathrm{back}})$. Let $y_{<u} = (y_1, \ldots, y_{u-1})$ denote the preceding action occurrences. The next-action distribution combines the normalized weights with uniform noise where $\epsilon \in [0, 1]$ is the noise-mixture weight.

$$p_{\mathrm{RFS}}(y_u = r \mid y_{<u}, H_k, \vartheta, \epsilon) = (1 - \epsilon)\frac{W_u(r)}{\sum_{s \in \mathcal{A}_k} W_u(s)} + \frac{\epsilon}{m_k}, \tag{5}$$

When $\epsilon > 0$, every action has positive probability, including actions with zero frontier compatibility.

Block likelihood. Starting from $\boldsymbol{q}_0 = \mathbf{0}$ and updating the state after each occurrence gives

$$e_{n,k}(a, b) = \prod_{u=1}^{T} p_{\mathrm{RFS}}(y_u \mid y_{<u}, H_k, \vartheta, \epsilon). \tag{6}$$

We set $e_{n,k}(a, b) = 0$ if any label in the block lies outside $\mathcal{A}_k$. Appendix A.2 describes its connection to the static frontier model, and works through Figure 5.

3.3 Segmental skill dynamics

Let $S_n = (0 = s_{n0} < s_{n1} < \cdots < s_{nL_n} = J_n)$ denote a duration-constrained segmentation of trace n, where L_n is the number of skill instances (segments) and $z_{n\ell} \in \{1, \ldots, K\}$ is the skill label assigned to segment ℓ. Let $P = (P_{hk})_{h,k=1}^{K}$ be a row-stochastic skill transition matrix with the no-self-transition constraint $P_{kk} = 0$, which requires $K \geq 2$. This is a modelling restriction: each instance resets its validity state, so merging adjacent same-type instances generally changes the likelihood. The single-skill TaskBench experiments instead condition on one complete instance and do not use an inter-skill transition matrix. Let $\Theta = (U, \vartheta, \pi, P, \rho)$, and ϕ collect the model specification, including δ_B, ϵ, the action supports, and the duration constraints. The model factorizes as

$$p(\boldsymbol{x}^{(n)}, S_n, z_n \mid J_n, \Theta, \phi) = p(S_n \mid J_n, \delta_B)\, \pi_{z_{n1}} \times \prod_{\ell=2}^{L_n} P_{z_{n,\ell-1}, z_{n\ell}} \prod_{\ell=1}^{L_n} e_{n, z_{n\ell}}(s_{n,\ell-1} + 1, s_{n\ell}). \tag{7}$$

The initial distribution π and transition matrix P govern successive skill instances, not individual actions. Within-instance repetition and non-adjacent skill reuse remain allowed. The normalized segmentation prior is specified in Appendix A.1.

Joint model. For $\mathcal{X} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$, let $S = (S_1, \ldots, S_N)$, $z = (z_1, \ldots, z_N)$, and $\boldsymbol{J} = (J_1, \ldots, J_N)$. The joint density is

$$p(\mathcal{X}, S, z, \Theta \mid \boldsymbol{J}, \phi) = p(U \mid \rho)\, p(\vartheta)\, p(\pi)\, p(P) p(\rho) \times \prod_{n=1}^{N} p(\boldsymbol{x}^{(n)}, S_n, z_n \mid J_n, \Theta, \phi). \tag{8}$$

Fixed coordinates of ϑ are conditioned on and their prior factors omitted. The likelihood couples shared structures with labelled paths: a proposed structure may explain a trace better under an alternative segmentation and assignment than under the current one. This motivates the path-marginal structural updates in Section 4. Skill-label exchangeability is restricted to permutations preserving the fixed action supports and slot-specific priors.

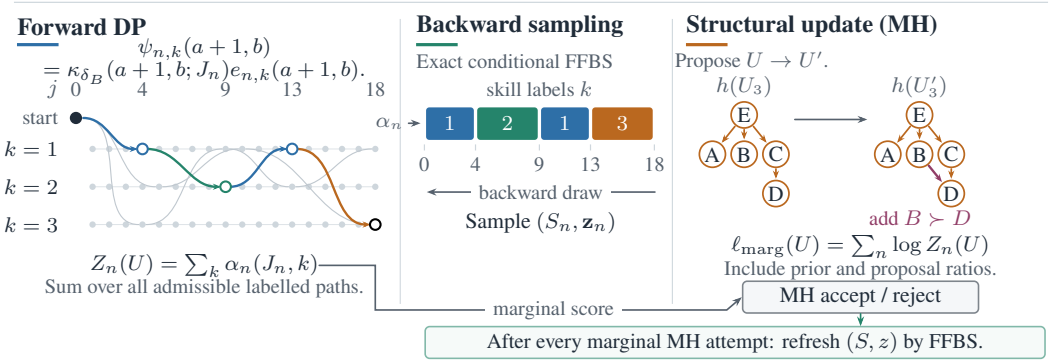


Figure 2: **One forward dynamic program, two uses.** The forward recursion (left) supports exact conditional path sampling (middle) and path-marginal structural scoring (right). Paths are redrawn by FFBS after every marginal MH attempt, whether accepted or rejected.

4 Posterior Inference

We jointly infer the trace-specific skill boundaries and assignments (S, z) and the shared skill structures U. A complete choice of boundaries and skill labels defines a *labelled path*. For fixed global parameters, a semi-Markov forward recursion sums the weights of all admissible paths, using recurrent partial-order likelihoods to score candidate blocks (Ostendorf et al., 1996; Yu, 2010).

Figure 2 illustrates two uses of this computation. The forward messages support exact conditional backward sampling of the labelled paths (Carter & Kohn, 1994; Frühwirth-Schnatter, 1994; Johnson & Willsky, 2013). The corresponding likelihood numerator also provides a path-marginal score for structural proposals $U \to U'$. A proposed library is therefore evaluated over alternative segmentations and assignments, rather than only against the currently sampled path. After every path-marginal Metropolis–Hastings (MH) attempt, whether accepted or rejected, the paths are redrawn by forward-filtering backward-sampling (FFBS) before subsequent path-conditioned updates.

Exact labelled-path inference. For fixed global parameters, each path in the left panel of Figure 2 represents a possible segmentation and skill assignment. An edge from boundary $a-1$ to j labelled by skill k assigns the block $\boldsymbol{x}_{a:j}^{(n)}$ to that skill. Its duration and emission factor is

$$\psi_{n,k}(a,j) = \kappa_{\delta_B}(a,j;J_n)\, e_{n,k}(a,j).$$

Here κ_{δ_B} is the unnormalized duration factor defined in Appendix A.3. Its product over segments gives the numerator of the segmentation prior; division by $C_{J_n}(\delta_B)$ provides the normalization.

Let $\mathcal{I}_j$ contain the admissible start positions for a block ending at j. The forward message $\alpha_n(j,k)$ sums the unnormalized weights of all paths covering positions $1{:}j$ and ending in skill k:

$$\alpha_n(j,k) = \sum_{a\in\mathcal{I}_j} \psi_{n,k}(a,j) \begin{cases} \pi_k, & a=1, \\ \sum_{h=1}^{K} \alpha_n(a-1,h) P_{hk}, & a>1, \end{cases} \qquad Z_n(U) = \sum_{k=1}^{K} \alpha_n(J_n,k). \quad (9)$$

Suppressing its dependence on $(\pi, P, \delta_B, \vartheta, \epsilon)$, the normalized trace likelihood is

$$p(\boldsymbol{x}^{(n)} \mid J_n, \Theta, \phi) = Z_n(U)/C_{J_n}(\delta_B).$$

The segmentation normalizer $C_{J_n}(\delta_B)$ cancels in structural acceptance ratios when only U changes, but is retained in predictive likelihoods. The middle panel illustrates backward sampling through the same forward messages. Starting at the trace end, we sample successive boundaries and skill labels to obtain an exact draw from $p\big(S_n, z_n \mid \boldsymbol{x}^{(n)}, U, \pi, P, \delta_B, \vartheta, \epsilon\big)$. Exactness concerns this conditional path distribution, not exploration of the global parameters. For a structural proposal, we recompute the forward sum under the proposed utilities

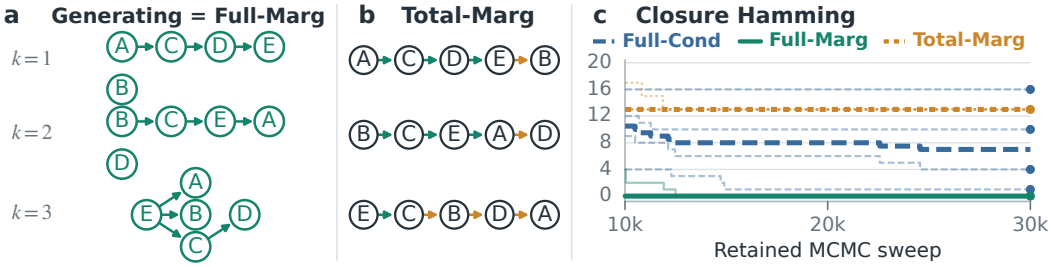


Figure 3: **Synthetic structure recovery.** (a) The generating library and the sampled FULL-MARG libraries. (b) Total-order fit, adding 13 closure relations. (c) Four chains per method and their median: 4,000 draws per chain from sweeps 10k–30k. Predictive sampling is specified in Appendix I.

Path-marginal structural inference. A proposed structure may fit the observations under a different segmentation but poorly under the current path. We therefore evaluate structural proposals using the path-marginal log-likelihood score $\ell_{\text{marg}}(U) = \sum_{n=1}^{N} \log Z_n(U)$ The MH acceptance ratio (Appendix A.3) combines its change with the prior and proposal ratios. After every attempt, accepted or rejected, FFBS redraws (S, z) before subsequent path-conditioned updates. This partially collapsed transition preserves the joint posterior under the stated update ordering (van Dyk & Park, 2008).

5 EXPERIMENTS AND RESULTS

We evaluate RPOS on model-matched synthetic traces, TaskBench-derived workflow simulations, and τ^3-Retail traces. Synthetic experiments test whether path marginalization improves joint recovery and prediction, and whether learned skills predict unseen combinations without retraining. TaskBench evaluates partial-order recovery, recurrent prediction, and generalization to unseen within-workflow orderings, treating each complete trace as one known skill instance. Retail assesses held-out prediction, source-skill reuse, and composition adaptation, alongside qualitative analysis of the learned structures.

Baselines and metrics. We compare RPOS with SEQ-HSMM (Johnson & Willsky, 2013), which models variable-length skill instances using within-skill CPA transitions shared across instances, and CompILE-CPA+SM, our adaptation of CompILE (Kipf et al., 2019) to CPA sequences. The latter learns a neural recognition model and code-specific first-order block decoders, then uses a separate semi-Markov model for prediction. Neither baseline explicitly represents partial-order dependencies. We evaluate recovery using F_1 scores when ground truth is available and examine learned skills qualitatively otherwise. Prediction and reuse are evaluated using held-out and query negative log-likelihood (NLL) per CPA occurrence, respectively. Definitions appear in Appendix D.

5.1 JOINT RECOVERY AND PREDICTIVE REUSE ON SYNTHETIC TRACES

Joint recovery. The model-matched corpora contain 100 training and 45 held-out traces from three skills over five labels, with $d = 2$, trace lengths $\{24, 32, 40, 48\}$, and segment widths 3–12. With execution parameters fixed, FULL-COND and FULL-MARG target the same posterior, but score structural proposals against the current path or after summing over paths, respectively (Section 4). In the four paired chains, every retained FULL-MARG library matches truth up to label permutation; no FULL-COND library does (Figure 3). Path marginalization lowers boundary Brier score from 0.098 to 0.064, co-skill Brier score from 0.1995 to 0.1047, and held-out NLL from 1.5444 to 1.4591. The total-order restriction ($d = 1$) changes both support and prior; its modal library adds 13 relations, attaining the enumerated minimum closure error $13/60$. With execution parameters jointly learned on a separate corpus, truth accounts for 99.46% of retained libraries (Appendices E.1–E.3).

Predictive reuse and alternative fitting. Using five additional independently generated skill libraries, we evaluate transfer from the six $i \to j \to i$ training compositions against six distinct-skill $i \to j \to k$ compositions. Each dataset has 120 training traces and 60 tests per familiar/unseen category. All adjacent transitions are observed in training, but the distinct-skill triples in the unseen

Table 1: **TaskBench with 24 training traces per workflow.** Length-conditioned NLL/action, averaged over 48 workflows (↓). *Exact*: recovered closures.

Model	Exact /48	Normal	Benign repeat	Repair
RPOS	48	0.6886	1.2856	1.2713
RPOS, $\kappa = 0$	48	0.7142	1.2386	1.3433
Intersection+RFS	48	0.6886	1.2855	1.2713
Intersection+RFS, $\kappa = 0$	48	0.7143	1.2386	1.3433
CompILE-CPA+SM	–	0.9534	1.6010	1.4398
SEQ-HSMM	–	0.9645	1.4094	1.3263

condition are not. Without training boundaries or labels, and with parameters frozen at test time, RPOS attains unseen NLL 1.3685, versus 1.4003 for SEQ-HSMM, and 1.5281 for total-order RPOS. Oracle training paths give 1.3549. RPOS has lower unseen-composition NLL than every tested non-oracle baseline on all five datasets (Appendix E.4).

A post-hoc differentiable relaxation retains exact path marginalization under its relaxed model. On the fixed-execution corpus, continuation recovers the generating library after strict projection in all four initializations. The soft model achieves mean NLL 1.4603 (SD 0.0006), compared with 1.4591 for the previously fitted MCMC predictive mixture. Settings and runtime comparisons with unmatched timing scopes appear in Appendix I.

5.2 Workflow Recovery and Compositional Generalization

An independent simulator generates normal executions, benign repetitions, and dependency repairs for 48 TaskBench Multimedia workflows with 4–7 actions (Shen et al., 2024). Each workflow is fitted as one skill with known whole-trace boundaries, using 6 or 24 training traces balanced across conditions; models receive action sequences but not the workflow graphs or simulator states. Training first-occurrence orders cover both directions of every incomparable pair with 3,456 held-out traces.

Recovery and recurrent prediction. RPOS recovers $47/48$ closures at $n = 6$ and $48/48$ at $n = 24$. First-occurrence intersection recovers all 48 at both training-set sizes, as implied by complete valid traces and the prescribed coverage. Intersection+RFS uses this graph with the same recurrent scorer and training-only exact integration over execution parameters; it tests the added value of Bayesian structure inference, not the validity mechanism. It predicts better at $n = 6$; at $n = 24$, all paired difference intervals include zero. Both methods outperform the sequence baselines. Relative to the no-invalidation ablation ($\kappa = 0$), RPOS improves repair prediction but worsens benign-repeat prediction (Table 1). Here $\kappa = \sigma(\omega) \in (0, 1)$ is the invalidation strength. In TaskBench, κ is sampled over $\{0.25, 0.75, 1\}$; the value $\kappa = 1$ is treated as the full-invalidation limit $\omega \to \infty$. The $\kappa = 0$ ablation directly fixes the invalidation strength to zero, rather than taking $\omega \to -\infty$, removing downstream invalidation and its associated backward-disruption cost, while allowing repeated actions.

Unseen ordering combinations. A separate repeat-free study on 30 workflows withholds one legal combination of ordering choices for two incomparable action pairs, while exposing both orientations of each pair during training. At $n = 24$, RPOS achieves unseen-combination NLL 0.6174, versus 0.6276 for Intersection+RFS, 2.3133 for total-order RPOS, 1.3660 for SEQ-HSMM, and 1.5143 for CompILE-CPA+SM. The paired *RPOS-minus-Intersection+RFS* difference is -0.01022 nats/action, despite identical extracted closures. When distinguishing unseen valid sequences from controls that violate exactly one true precedence relation, RPOS and Intersection+RFS achieve AUROCs of 0.9950 and 0.9949, respectively.

Some held-out executions also contain previously unseen orientations of unselected pairs. On the prespecified 22-workflow subset without this additional local novelty, Intersection+RFS has lower NLL at both budgets (0.4520 versus 0.4521 at $n = 24$), and the intersection violation score achieves AUROC 1.000. The advantage over intersection is therefore setting-dependent and concerns predictive distributions, not better point-graph recovery or uniformly superior compositional generalization. The no-invalidation ablation performs similarly on this repeat-free task. Full results and diagnostics appear in Appendix F; bootstrap intervals condition on the fixed data and fitted models.

5.3 Skill Discovery and Reuse in Retail Traces

The τ^3-Retail corpus, derived from τ-bench (Yao et al., 2024; Barres et al., 2025), contains 436 modelling traces and 7,085 occurrences over 24 CPA labels: 372 traces from 93 training tasks and 64 from 16 internal-development tasks. Appendix G.1 documents annotation provenance and a separate 20-trace final test. The evaluation focuses on prediction and structural analysis.

Learned skills and their composition. Figure 4 shows two skills and transitions among all five skills from a training-only fit (Full library: Figure 7). Within these skills, identity resolution precedes account retrieval, while order retrieval branches into query answering, variant retrieval, and other information-gathering actions. CPA labels may appear in multiple skills: "goal selection" appears in both displayed skills. Between-skill transitions compose these local partial orders into a recurrent procedure. Within-run mean transition probabilities are 0.98 from identity verification to investigation, 0.84 from investigation to preparation, and 0.97 from preparation to execution. The reverse transition from execution to preparation has probability 0.84, allowing recurrence across skill instances. The run and skills were selected post hoc for illustration; skill names were assigned after fitting (Appendix G.2).

Comparison with sequence models. The illustrated RPOS fit uses all five skills, whereas CompILE-CPA+SM assigns 95.8% of occurrences to two codes. The two most-used codes account for at least 92% in 11 of 12 fits (Figure 8). This describes code usage, not semantic correctness. With four 30,000-sweep runs per capacity, RPOS's mean development NLL falls from 1.821 at $K = 4$ to 1.737 at $K = 8$ (Table 14). It is lower than CompILE-CPA+SM at $K = 4, 8$ but slightly higher at $K = 5, 6$;

On the separate 20-trace final test, RPOS has NLL 1.600 and 1.590 at $K = 5, 8$, versus 1.305 for SEQ-HSMM and 1.303 for bigram. Thus, RPOS does not achieve the lowest sequence NLL on Retail. Unlike the sequence baselines, it explicitly represents within-skill precedence constraints, which can inform dependency-aware scheduling. Training budgets differ across methods; development results appear in Appendix G.1.

Source training versus adaptation. A post-hoc follow-up uses the previously evaluated five tasks, with one support trace and three queries each. Shared fixes the source-trained library and adapts (π, P); Support-only learns both the skill library and composition parameters from the same demonstration; Frozen leaves the source model unchanged. Across all four $K = 5$ source chains, Shared achieves query NLL 1.7798 after 20 updates, versus $2.4001/2.2785$ for Support-only after 2,000 and 10,000 sweeps, respectively. However, Frozen achieves lower query NLL (1.5920): adaptation worsens prediction on every task, and 200 updates do not change the ranking. All three sequence baselines have lower adapted query NLL (Table 15).

Separately, strict projection without refitting worsens the differentiable model's internal-development NLL in all 16 fits across $K = 4, 5, 6, 8$; at $K = 5$, the mean rises from 1.8191 to 2.2698. Two local relations recur in all fits. This result concerns projection without refitting and does not establish that strict partial-order models generally predict worse. (Appendix I.3).

5.4 Downstream Evaluation of the Execution System

We compare a stepwise tool agent (N) with an executor (G) on 120 paired synthetic Retail tasks, generated with new parameter draws from 23 task families used during development. Both use `qwen3.8-flash`, identical requests and tools, and separate copies of the same initial environment. Expected outcomes are available only to the evaluator. Whereas N calls the model to select each action from the interaction history, G combines model-extracted request fields with the five-skill library from run s102, predefined CPA-to-API mappings, state updates, and policy checks. The library and controller are fixed before evaluation. Resource totals include all online model calls, including those for request updates and fallback, and those made during unsuccessful episodes.

Strict success requires correct task outcomes and final responses, authentication, policy compliance, required supporting evidence, receipt of required user events, and no execution failure. G achieves strict success on $120/120$ tasks, compared with $103/120$ for N; business completion is $120/120$ and $119/120$, respectively. G reduces online token use by 86.97% and model requests by 83.77% (Table 2). The paired, family-stratified 95% bootstrap interval for the token ratio G/N is $[0.1111, 0.1513]$. No task succeeds under N but fails under G. Assuming independent task-parameter draws, the one-sided

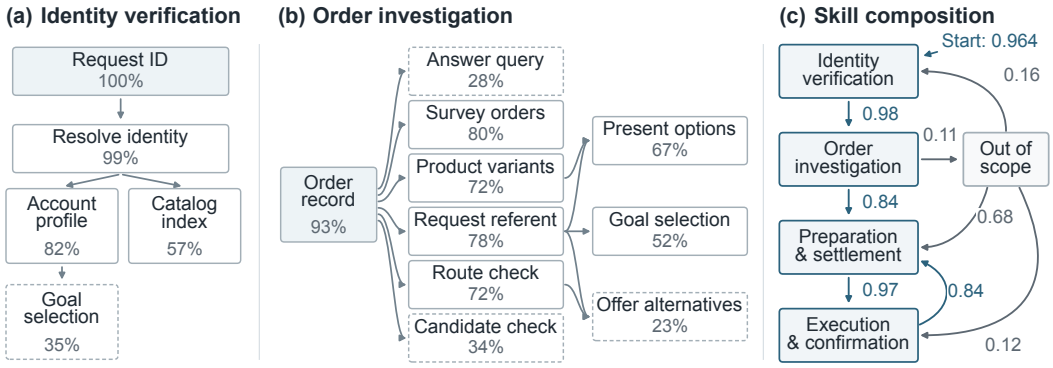


Figure 4: **Retail skills and composition.** Training-only s102 fit ($K = 5$); selection and naming are post hoc. (a,b) Node percentages show the fraction of each action's occurrence responsibility assigned to the displayed skill ($\geq 20\%$; dashed below 50%); Arrows are transitively reduced after thresholding within-run relation frequencies at 0.70. (c) Mean transition probabilities ≥ 0.10. Missing arrows imply neither independence nor zero transition probability.

Table 2: **Online execution on 120 paired synthetic Retail tasks.** N: stepwise agent; G: executor. Token totals include calls for request updates and fallback, as well as calls from unsuccessful episodes.

Metric	N**: Stepwise**	G**: Executor**
Strict success (%)	85.8	100.0
Model requests (per task)	8.22	1.33
Token use (thousands per task)	23.10	3.01

95% upper confidence bound on the probability of N succeeding and G failing is 2.47%. This is sufficient to meet the prespecified five-percentage-point non-inferiority margin for strict success over the evaluated mix of task families. The comparison evaluates the complete executor, not the learned precedence relations in isolation (Appendix J).

Illustrative parallel scheduling. A separate mock workload rather than live tool calls compares sequential and parallel scheduling on the displayed investigation graph. With ten required one-second jobs and sufficient concurrency, parallel scheduling reduces mean completion time from $10.011\,\mathrm{s}$ to $3.004\,\mathrm{s}$ across three paired runs: approximately 70% less time ($3.33\times$ speedup). Both schedules execute the same jobs and respect all displayed dependencies (Appendix J.3).

6 Conclusion and Limitations

RPOS learns inspectable, reusable skill libraries within a probabilistic framework that separates shared action-precedence structure from each skill instance's evolving execution state. Path marginalization supports joint inference of boundaries, assignments, and reusable structures. Controlled experiments demonstrate improved recovery with path marginalization and prediction on unseen skill compositions. A differentiable relaxation provides an alternative optimization-based fitting method. On synthetic Retail tasks, our execution system reduces online model calls and token use while meeting the prespecified non-inferiority margin for strict task success. A separate mock workload illustrates the completion-time benefit of parallel scheduling.

Limitations. Our annotated agent-trace evaluation covers only τ^3-Retail. Additional domains and annotation pipelines are needed to assess generalization and sensitivity to action labelling. The fully latent MCMC runs do not adequately explore all alternative skill decompositions, so sampled structure frequencies are treated as finite-run summaries rather than calibrated posterior probabilities.

AI Use Statement

Large language models assisted with manuscript editing and organization. CPA annotations were produced with LLM assistance following the protocol of Li et al. (2026d). Language models also serve as components of the evaluated execution systems; all online model calls are included in the reported resource accounting.

Ethics Statement

This work uses τ^3-Retail traces and simulated TaskBench workflows and introduces no new human-subject data. The experimental traces are synthetic or anonymized under the original benchmark protocols. Downstream evaluation takes place entirely in an isolated synthetic Retail environment. Although partial-order execution can reduce reliance on open-ended model decisions, deployment with real APIs would require further validation and appropriate safeguards.

Reproducibility Statement

The appendices document the model specifications, derivations, data-generation procedures, and experimental protocols. An anonymized supplementary package provides the inference, sampling, differentiable relaxation, and execution code. As detailed in Appendix J, we archive all 1,341 N/G execution records, together with prompts and environment snapshots. These materials support offline replay and recomputation of the reported downstream metrics without additional model calls.

References

Umut A. Acar, Matthias Blume, and Jacob Donham. A consistent semantics of self-adjusting computation. *Journal of Functional Programming*, 23(3):249–292, 2013. Conference version: ESOP 2007; preprint arXiv:1106.0478.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

C. K. Carter and R. Kohn. On Gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994. doi: 10.1093/biomet/81.3.541.

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850. Association for Computing Machinery, 2021. doi: 10.1145/3453483.3454080.

Sylvia Frühwirth-Schnatter. Data augmentation and dynamic linear models. *Journal of Time Series Analysis*, 15(2):183–202, 1994. doi: 10.1111/j.1467-9892.1994.tb00184.x.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. URL https://arxiv.org/abs/1807.03480.

Matthew J. Johnson and Alan S. Willsky. Bayesian nonparametric hidden semi-Markov models. *Journal of Machine Learning Research*, 14(20):673–701, 2013. URL https://www.jmlr.org/papers/v14/johnson13a.html.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. CompILE: Compositional imitation learning and execution. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3418–3428. PMLR, 2019. URL https://proceedings.mlr.press/v97/kipf19a.html.

Lingpeng Kong, Chris Dyer, and Noah A. Smith. Segmental recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Humam Kourani and Sebastiaan J. van Zelst. POWL: Partially ordered workflow language. In *Business Process Management*, volume 14159 of *Lecture Notes in Computer Science*, pp. 92–108. Springer, 2023. doi: 10.1007/978-3-031-41620-0_6.

Humam Kourani, Sebastiaan J van Zelst, Daniel Schuster, and Wil MP van der Aalst. Discovering partially ordered workflow models. *Information Systems*, 128:102493, 2025.

Maikel Leemans and Wil MP van der Aalst. Discovery of frequent episodes in event logs. In *International symposium on data-driven process discovery and analysis*, pp. 1–31. Springer, 2014.

Sander JJ Leemans, Sebastiaan J van Zelst, and Xixi Lu. Partial-order-based process mining: a survey and outlook: Sjj leemans et al. *Knowledge and Information Systems*, 65(1):1–29, 2023.

Dongqing Li, Zheqiao Cheng, Geoff K. Nicholls, and Quyu Kong. De-linearizing agent traces: Bayesian inference of latent partial orders for efficient execution, 2026a. URL https://arxiv.org/abs/2602.02806.

Dongqing Li, Geoff K. Nicholls, Jeong Eun Lee, and Chuxuan Jiang. Hierarchical partial-order models for ranking, 2026b. URL https://arxiv.org/abs/2606.25062.

Dongqing Li, Geoff K. Nicholls, Shiyi Sun, and You Luo. A differentiable bayesian relaxation for latent partial-order inference, 2026c. URL https://arxiv.org/abs/2605.06976.

Songqi Li, Dongqing Li, and Zheqiao Cheng. Canonical procedural actions: An auditable annotation protocol for tool-use agent traces, 2026d. URL https://arxiv.org/abs/2609.24264.

Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. Build systems à la carte: Theory and practice. *Journal of Functional Programming*, 30:e11, 2020.

Raymond J. Mooney. Generalizing the order of operators in macro-operators. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 270–283, 1988. URL https://www.cs.utexas.edu/~ai-lab/pub-view.php?PubID=126870.

Geoff K. Nicholls, Jeong Eun Lee, Nicholas Karn, David Johnson, Rukuang Huang, and Alexis Muir-Watt. Bayesian inference for partial orders from random linear extensions: Power relations from 12th century royal acta. *The Annals of Applied Statistics*, 19(2):1663–1690, 2025. doi: 10.1214/24-AOAS2002.

Mari Ostendorf, Vassilios Digalakis, and Owen A. Kimball. From HMM's to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 4(5):360–378, 1996. doi: 10.1109/89.536930.

Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems*, volume 17, pp. 1185–1192. MIT Press, 2004.

Luigi Seminara, Giovanni Maria Farinella, and Antonino Furnari. Differentiable task graph learning: Procedural activity representation and online mistake detection from egocentric videos. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6d19163eaec3b0f0accbe462a0139466-Abstract-Conference.html.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. TaskBench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0148.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. doi: 10.1109/TKDE.2004.47.

David A. van Dyk and Xiyun Jiao. Metropolis–hastings within partially collapsed Gibbs samplers. *Journal of Computational and Graphical Statistics*, 24(2):301–327, 2015. doi: 10.1080/10618600.2014.930041.

David A. van Dyk and Taeyoung Park. Partially collapsed Gibbs samplers: Theory and methods. *Journal of the American Statistical Association*, 103(482):790–796, 2008. doi: 10.1198/016214508000000409.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks, 2025a. URL https://arxiv.org/abs/2504.06821.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023. doi: 10.48550/arXiv.2305.18323. URL https://arxiv.org/abs/2305.18323.

Salma El Yadouni and Guanyi Li. Tracecompiler: Skill-guided mining and compilation of llm agent traces into mostly deterministic workflows. *arXiv preprint arXiv:2608.02680*, 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Shun-Zheng Yu. Hidden semi-Markov models. *Artificial Intelligence*, 174(2):215–243, 2010. doi: 10.1016/j.artint.2009.11.011.

Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 7(2):4126–4133, 2022. doi: 10.1109/LRA.2022.3146589.

A Model and Inference Details

This appendix specifies the model, recurrent block likelihood, and posterior updates. We use K for the number of reusable skill types, d for the product-order dimension, and J_n for the number of canonical procedural-action (CPA) occurrences in trace n. Block endpoints are one-indexed and inclusive. The validity state is computed deterministically from each block's action history; it is not an additional latent variable.

A.1 Model specification

For trace n, the observed sequence is

$$\boldsymbol{x}^{(n)} = (x_1^{(n)}, \ldots, x_{J_n}^{(n)}) \in \mathcal{A}^{J_n}.$$

A segmentation is an ordered sequence of boundaries

$$S_n = (0 = s_{n0} < s_{n1} < \cdots < s_{nL_n} = J_n),$$

with widths $w_{n\ell} = s_{n\ell} - s_{n,\ell-1}$ satisfying $1 \le D_{\min} \le w_{n\ell} \le D_{\max}$. Segment ℓ is assigned a skill label $z_{n\ell} \in \{1, \ldots, K\}$. We call (S_n, z_n) a *labelled path* and write $\mathcal{S}(J_n)$ for the set of legal segmentations. We condition on the observed trace lengths and assume $\mathcal{S}(J_n) \neq \varnothing$ for every trace.

For $0 < \delta_B < 1$, the normalized segmentation prior is

$$p(S_n \mid J_n, \delta_B) = \frac{\delta_B^{L_n-1}(1-\delta_B)^{J_n-L_n}}{C_{J_n}(\delta_B)}\, \mathbb{I}[S_n \in \mathcal{S}(J_n)], \tag{10}$$

where

$$C_J(\delta_B) = \sum_{S \in \mathcal{S}(J)} \delta_B^{L(S)-1}(1-\delta_B)^{J-L(S)}$$

and $L(S)$ is the number of segments in S. Skill labels follow

$$p(z_n \mid S_n, \pi, P) = \pi_{z_{n1}} \prod_{\ell=2}^{L_n} P_{z_{n,\ell-1}, z_{n\ell}},$$

where π is a probability vector and P is row-stochastic with $P_{kk} = 0$. This no-self-transition specification requires $K \geq 2$. Excluding adjacent instances of the same skill is a modelling restriction: each instance resets its validity state, so two adjacent same-type instances are not generally equivalent to a single longer instance.

Skill action supports and templates. Each skill k has a fixed finite nonempty action support $\mathcal{A}_k \subseteq \mathcal{A}$, with $m_k = |\mathcal{A}_k|$. We index template nodes directly by their CPA labels, so no separate role set or role-to-CPA map is needed. Repeated occurrences of a label execute the same node within a skill, while the same label may index distinct nodes in different skill templates.

The support $\mathcal{A}_k$ specifies which actions may occur in skill k, not which actions must occur in every invocation. Restricted skill-specific supports therefore supply prior knowledge about admissible action membership. In the Retail experiments, every skill instead uses the full 24-label vocabulary, $\mathcal{A}_k = \mathcal{A}$ for all k, so no skill-specific action membership restrictions are supplied. The model infers boundaries, assignments, precedence relations, and segment-level dynamics; it does not learn hard action-support subsets.

A utility matrix $U_k \in \mathbb{R}^{m_k \times d}$, with rows indexed by $c \in \mathcal{A}_k$, induces the strict partial order $H_k = h(U_k)$:

$$c \succ_{H_k} c' \quad \Longleftrightarrow \quad U_{k,c,j} > U_{k,c',j} \quad \text{for all } j = 1, \ldots, d. \tag{11}$$

The relation $c \succ_{H_k} c'$ denotes template precedence, not a requirement that every occurrence of c precede every occurrence of c'. Distinct labels are incomparable when neither utility row strictly dominates the other. At fixed remaining parameters, the likelihood depends on U_k only through H_k.

Define the compatibility indicator

$$\chi_{n,k}(a,b) = \mathbb{I}\Big[x_t^{(n)} \in \mathcal{A}_k \text{ for all } t = a, \ldots, b\Big]. \tag{12}$$

A block is compatible with skill k if and only if $\chi_{n,k}(a,b) = 1$; otherwise its likelihood under that skill is zero. When $\mathcal{A}_k = \mathcal{A}$ for every skill, compatibility imposes no skill-specific exclusion on blocks over the observed vocabulary.

Write $\mathcal{X} = \{\boldsymbol{x}^{(n)}\}_{n=1}^N$, $\boldsymbol{J} = (J_1, \ldots, J_N)$, $S = (S_1, \ldots, S_N)$, and $z = (z_1, \ldots, z_N)$. We specify the model with δ_B and ϵ fixed. The correlation ρ is inferred and included in Θ. Let

$$\Theta = (U, \vartheta, \pi, P, \rho), \qquad \vartheta = (\beta, \omega, \lambda_{\text{rep}}, \lambda_{\text{back}}),$$

and

$$\phi = (\delta_B, \epsilon, K, d, \{\mathcal{A}_k\}_{k=1}^K, D_{\min}, D_{\max}).$$

The joint density is

$$\begin{aligned} p(\mathcal{X}, S, z, \Theta \mid \boldsymbol{J}, \phi) = p(U \mid \rho)p(\vartheta)p(\pi)p(P)p(\rho) \\ \times \prod_{n=1}^N \Bigg[p(S_n \mid J_n, \delta_B)\, p(z_n \mid S_n, \pi, P) \\ \times \prod_{\ell=1}^{L_n} e_{n,z_{n\ell}}(s_{n,\ell-1}+1, s_{n\ell}) \Bigg], \end{aligned} \tag{13}$$

where $e_{n,k}$ is defined below. Fixed coordinates of ϑ are conditioned on, with their prior factors omitted. If an experiment fixes ρ rather than inferring it, then ρ is moved from Θ to ϕ and the factor $p(\rho)$ is omitted; we state this exception where it applies.

A.2 Recurrent execution within a skill instance

For a compatible block $x_{a:b}^{(n)}$ assigned to skill k, write its observed action sequence as

$$y_u = x_{a+u-1}^{(n)} \in \mathcal{A}_k, \qquad u = 1, \dots, T, \qquad T = b - a + 1.$$

Here, y_u is the observed CPA label itself, not an additional latent variable. Each candidate block starts from $\boldsymbol{q}_0 = \mathbf{0}$, with one state coordinate per $c \in \mathcal{A}_k$ and no carry-over from earlier blocks. For $\omega \in \mathbb{R}$, let $\mathrm{sigmoid}(\omega) = (1 + e^{-\omega})^{-1}$. After action y_u executes,

$$q_u(c) = \begin{cases} 1, & c = y_u, \\ q_{u-1}(c)\big[1 - \mathbb{I}[y_u \succ_{H_k} c]\,\mathrm{sigmoid}(\omega)\big], & c \neq y_u. \end{cases} \tag{14}$$

Executing an action sets its own validity to one and attenuates the validity of all its strict successors, including indirect successors. Thus $\boldsymbol{q}_u \in [0,1]^{m_k}$. This is a model-defined state, not an observed measure of environment validity.

Let $\mathrm{Pred}_{H_k}(c) = \{v \in \mathcal{A}_k : v \succ_{H_k} c\}$ include all strict predecessors. Before step u, frontier compatibility is

$$F_u(c) = \prod_{v \in \mathrm{Pred}_{H_k}(c)} q_{u-1}(v), \tag{15}$$

with an empty product equal to one. Progress and backward disruption are

$$Q_u(c) = \log\left[1 + \sum_{v \in \mathcal{A}_k \setminus \{c\}} \mathbb{I}[c \succ_{H_k} v]\big(1 - q_{u-1}(v)\big)\right], \tag{16}$$

$$C_u^{\mathrm{back}}(c) = \mathrm{sigmoid}(\omega) \sum_{v \in \mathcal{A}_k \setminus \{c\}} \mathbb{I}[c \succ_{H_k} v] q_{u-1}(v). \tag{17}$$

For finite nonnegative β, λ_{rep}, and λ_{back}, the unnormalized choice weight is

$$W_u(c) = F_u(c) \exp\big\{\beta Q_u(c) - \lambda_{\mathrm{rep}} q_{u-1}(c) - \lambda_{\mathrm{back}} C_u^{\mathrm{back}}(c)\big\}. \tag{18}$$

For $\epsilon \in [0,1]$, the next-action distribution is

$$p_{\mathrm{RFS}}(y_u = c \mid y_{<u}, H_k, \vartheta, \epsilon) = (1 - \epsilon)\frac{W_u(c)}{\sum_{v \in \mathcal{A}_k} W_u(v)} + \frac{\epsilon}{m_k}. \tag{19}$$

The block likelihood is defined piecewise:

$$e_{n,k}(a,b) = \begin{cases} \prod_{u=1}^{T} p_{\mathrm{RFS}}(y_u \mid y_{<u}, H_k, \vartheta, \epsilon), & \chi_{n,k}(a,b) = 1, \\ 0, & \chi_{n,k}(a,b) = 0. \end{cases} \tag{20}$$

For a compatible block, every state update and sum is indexed by labels in $\mathcal{A}_k$.

Every finite nonempty partial order has a node with no predecessors. Such a node has $F_u(c) = 1$ and a strictly positive exponential factor, so $\sum_{c \in \mathcal{A}_k} W_u(c) > 0$. The structured component of Equation (19) is therefore well defined. When $\epsilon > 0$, each action in $\mathcal{A}_k$ has probability at least ϵ/m_k; every compatible block consequently has positive likelihood, including blocks with repeated labels and blocks longer than m_k when permitted by the duration support. For any fixed block length, these likelihoods sum to one over all CPA strings, assigning zero probability outside $\mathcal{A}_k$.

Connection to repeat-free execution. Along a prefix that respects the partial order and contains no repeated actions, validity is binary. For an unexecuted action, F_u is its predecessor-eligibility indicator and $C_u^{\mathrm{back}} = 0$. Holding the other parameters fixed, taking $\lambda_{\mathrm{rep}} \to \infty$ suppresses previously executed actions in the structured component while an unexecuted frontier action remains. A repeat-free limit for the full mixture additionally requires $\epsilon = 0$ or a separately defined noise component that excludes executed actions. With the stated uniform noise and fixed $\epsilon > 0$, every executed action still has probability at least ϵ/m_k. This repeat-free interpretation applies only up to m_k executions; the length-conditioned model does not impose termination when all actions have executed.

(a) Shared template

$H_2 : B \succ C \succ E$



$F_u(B) = 1$

$F_u(C) = q_{u-1}(B)$

$F_u(E) = q_{u-1}(B)\, q_{u-1}(C)$

Full-invalidation limit

$\mathrm{sigmoid}(\omega) \to 1$

(b) Validity after execution

$x^{(n)}_{a:b}$, $b = a + 4$

Step u	0	1	2	3	4	5
Role y_u	–	B	C	B	C	E
$q_u(B)$	0	1	1	1	1	1
$q_u(C)$	0	0	1	0	1	1
$q_u(E)$	0	0	0	0	0	1

Repeat B: $q_3(C) = 0$. Re-execute C: $q_4(C) = 1$.

(c) Choice scores before execution

	Step u	1	2	3	4	5
Frontier compatibility (before step u)	$F_u(B)$	1	1	1	1	1
	$F_u(C)$	0	1	1	1	1
	$F_u(E)$	0	0	1	0	1
Progress	$Q_u(y_u)$	$\log 3$	$\log 2$	$\log 2$	$\log 2$	0
Repeat term	$q_{u-1}(y_u)$	0	0	1	0	0
Downstream disruption	$C^{\mathrm{back}}_u(y_u)$	0	0	1	0	0
Unnormalized weight	$W_u(y_u)$	3^β	2^β	$2^\beta \times e^{-\lambda_{\mathrm{rep}} - \lambda_{\mathrm{back}}}$	2^β	1

State values and unnormalized scores are not next-role probabilities.

Figure 5: **Revisitation and repair within one skill instance.** The template is $B \succ_{H_2} C \succ_{H_2} E$, and the block is (B, C, B, C, E). The figure uses the full-invalidation limit $\mathrm{sigmoid}(\omega) \to 1$. Re-executing B at step 3 sets $q_3(C) = 0$, reducing E's frontier compatibility to zero until C executes again. Step 3 is the only step with both a nonzero repeat term and a nonzero backward-disruption term. Uniform noise, when present, can still assign positive probability to an action with zero frontier compatibility.

Worked example. For the block (B, C, B, C, E) under $B \succ_{H_2} C \succ_{H_2} E$, the state before step 3 is

$$\big(q_2(B), q_2(C), q_2(E)\big) = (1, 1, 0),$$

so $F_3(E) = 1$. Revisiting B gives

$$Q_3(B) = \log 2, \qquad C^{\mathrm{back}}_3(B) = \mathrm{sigmoid}(\omega),$$

and

$$W_3(B) = 2^\beta \exp\{-\lambda_{\mathrm{rep}} - \lambda_{\mathrm{back}}\, \mathrm{sigmoid}(\omega)\}\,.$$

The resulting state is

$$\big(q_3(B), q_3(C), q_3(E)\big) = \big(1, 1 - \mathrm{sigmoid}(\omega), 0\big),$$

so $F_4(E) = 1 - \mathrm{sigmoid}(\omega)$. This compatibility vanishes only in the full-invalidation limit shown in Figure 5. Re-executing C restores $q_4(C) = 1$ and hence $F_5(E) = 1$. The model thus assigns probabilities to revisitation and repair without introducing cycles into the shared template.

A.3 Exact path inference and path-marginal structure updates

Throughout this subsection, let

$$\xi = (\rho, \vartheta, \pi, P, \delta_B, \epsilon)$$

collect the quantities held fixed during a structural update. The fixed action supports, duration bounds, and observed trace lengths are understood. For a legal block $[a, b]$, define

$$\kappa_{\delta_B}(a, b; J_n) = (1 - \delta_B)^{b-a} \delta_B^{\mathbb{I}[b < J_n]}, \tag{21}$$

$$\psi_{n,k}(a, b) = \kappa_{\delta_B}(a, b; J_n) e_{n,k}(a, b). \tag{22}$$

Their duration factors satisfy

$$\prod_{\ell=1}^{L_n} \kappa_{\delta_B}(s_{n,\ell-1}+1, s_{n\ell}; J_n) = \delta_B^{L_n-1}(1-\delta_B)^{J_n-L_n}.$$

For $j = 1, \ldots, J_n$, let

$$\mathcal{I}_j = \{a \in \{1, \ldots, j\} : D_{\min} \le j - a + 1 \le D_{\max}\}.$$

The forward recursion is

$$\alpha_n(j,k) = \sum_{a \in \mathcal{I}_j} \psi_{n,k}(a,j) \begin{cases} \pi_k, & a = 1, \\ \displaystyle\sum_{h=1}^{K} \alpha_n(a-1,h) P_{hk}, & a > 1. \end{cases} \tag{23}$$

An empty sum is zero. The terminal sum is

$$Z_n(U) = \sum_{k=1}^{K} \alpha_n(J_n, k), \tag{24}$$

with dependence on ξ suppressed. Since the block factors omit only the segmentation normalizer, the normalized trace likelihood is

$$p(\boldsymbol{x}^{(n)} \mid J_n, U, \xi) = \frac{Z_n(U)}{C_{J_n}(\delta_B)}. \tag{25}$$

Exact backward sampling. For a trace with $Z_n(U) > 0$, sample the final skill from

$$\Pr(z_{n,L_n} = k \mid \boldsymbol{x}^{(n)}, U, \xi) = \frac{\alpha_n(J_n, k)}{Z_n(U)}. \tag{26}$$

If the current backward state ends at j with skill k, sample its block start $a \in \mathcal{I}_j$ and, when $a > 1$, its preceding skill h:

$$\begin{aligned} &\Pr(a, h \mid j, k, \boldsymbol{x}^{(n)}, U, \xi) \\ &\quad = \frac{\psi_{n,k}(a,j)}{\alpha_n(j,k)} \begin{cases} \pi_k, & a = 1,\ h = \varnothing, \\ \alpha_n(a-1,h) P_{hk}, & a > 1,\ h \in \{1, \ldots, K\}. \end{cases} \end{aligned} \tag{27}$$

All other choices have probability zero. Backward sampling visits only states with positive forward weight. If $a > 1$, continue with $j \leftarrow a - 1$ and $k \leftarrow h$; otherwise stop. This forward-filtering backward-sampling (FFBS) procedure gives an exact draw from

$$p(S_n, z_n \mid \boldsymbol{x}^{(n)}, U, \xi).$$

Exactness concerns this conditional path distribution, not finite-run exploration of the global parameters.

Path-marginal structural target. Define

$$\Pi_\xi(U, S, z) = p(U, S, z \mid \mathcal{X}, \xi).$$

Its utility marginal is

$$\overline{\Pi}_\xi(U) = \sum_{S,z} \Pi_\xi(U, S, z) \propto p(U \mid \rho) \prod_{n=1}^{N} Z_n(U),$$

because $\prod_n C_{J_n}(\delta_B)$ is constant when only U changes. The path-marginal score is therefore

$$\ell_{\mathrm{marg}}(U) = \sum_{n=1}^{N} \log Z_n(U). \tag{28}$$

For $U' \sim q_U(\cdot \mid U, \rho)$, the log Metropolis–Hastings ratio is

$$\begin{aligned} \log R_U^{\mathrm{marg}} = {} & \ell_{\mathrm{marg}}(U') - \ell_{\mathrm{marg}}(U) + \log p(U' \mid \rho) - \log p(U \mid \rho) \\ & + \log q_U(U \mid U', \rho) - \log q_U(U' \mid U, \rho). \end{aligned} \tag{29}$$

The acceptance probability is $\min\{1, R_U^{\mathrm{marg}}\}$. For a symmetric Gaussian row proposal with a state-independent row-selection rule, the proposal terms cancel. Unlike a path-conditional update, this ratio evaluates the proposed structure after summing over all admissible labelled paths. The segmentation normalizer remains necessary for normalized prediction and for updates that change δ_B.

Partially collapsed ordering. Let $K_{\mathrm{marg},\xi}$ preserve $\overline{\Pi}_\xi$. Define

$$K_{\mathrm{pc},\xi}\big((U,S,z),\,dU'\,dS'\,dz'\big) = K_{\mathrm{marg},\xi}(U,dU')\,\Pi_\xi(dS'\,dz' \mid U'). \tag{30}$$

Proposition 1. *For fixed ξ, the composed kernel $K_{\mathrm{pc},\xi}$ preserves the joint posterior Π_ξ.*

Proof. Using invariance of $K_{\mathrm{marg},\xi}$ for the utility marginal,

$$\begin{aligned}
&\int \Pi_\xi(dU,dS,dz)\,K_{\mathrm{pc},\xi}\big((U,S,z),dU'\,dS'\,dz'\big)\\
&= \left[\int \overline{\Pi}_\xi(dU)\,K_{\mathrm{marg},\xi}(U,dU')\right]\Pi_\xi(dS'\,dz' \mid U')\\
&= \overline{\Pi}_\xi(dU')\Pi_\xi(dS'\,dz' \mid U')\\
&= \Pi_\xi(dU',dS',dz').
\end{aligned}$$

□

We implement Equation (30) by refreshing the paths immediately after every marginal MH attempt, including rejection:

$$U \xrightarrow{\text{path-marginal MH}} U_{\mathrm{new}} \xrightarrow{\text{FFBS}} (S_{\mathrm{new}}, z_{\mathrm{new}}) \xrightarrow{\text{path-conditioned updates}} \cdots .$$

On rejection, $U_{\mathrm{new}} = U$, but a fresh path is still drawn. In this implementation, no path-conditioned update intervenes between the marginal attempt and the refresh. This is the ordering covered by Proposition 1.

Appendix E.1 reports finite-state checks of the forward recursion, FFBS sampler, recurrent scorer, and composed transition, together with a stale-path negative control. These checks concern the tested implementations and do not establish global posterior convergence.

A.4 Priors and posterior updates

For the continuous recurrent-parameter specification, the priors are

$$\begin{aligned}
\boldsymbol{U}_{k,c} \mid \rho &\sim \mathcal{N}(\mathbf{0},\Sigma_\rho), & \Sigma_\rho &= (1-\rho)I_d + \rho\mathbf{1}_d\mathbf{1}_d^\top,\\
\beta,\lambda_{\mathrm{rep}},\lambda_{\mathrm{back}} &\overset{\mathrm{ind}}{\sim} \mathrm{Gamma}(2,2),\\
\omega &\sim \mathcal{N}(0,2^2),\\
\pi &\sim \mathrm{Dir}(\eta_\pi\mathbf{1}_K),\\
P_{h,-h} &\sim \mathrm{Dir}(\eta_P\mathbf{1}_{K-1}), & P_{hh} &= 0.
\end{aligned} \tag{31}$$

Utility rows, indexed by $c \in \mathcal{A}_k$, are independent conditional on ρ, and the remaining prior factors are independent. We use $\eta_\pi = \eta_P = 1$ unless an experiment specifies otherwise. Positive definiteness requires $-1/(d-1) < \rho < 1$ for $d > 1$; when $d = 1$, $\Sigma_\rho = [1]$ and this utility prior does not depend on ρ.

The continuous-ρ specification uses

$$p_{\mathrm{c}}(\rho) \propto f_{\mathrm{Beta}(1,1/6)}(\rho)\,\mathbb{I}[0 < \rho < 1 - 5\times10^{-3}].$$

A grid-based specification instead defines a discrete prior. For its stated grid $\{\rho_i\}_{i=1}^G$, density-proportional prior masses are

$$p_{\mathrm{g}}(\rho_i) = \frac{f_{\mathrm{Beta}(1,1/6)}(\rho_i)}{\sum_{j=1}^G f_{\mathrm{Beta}(1,1/6)}(\rho_j)}.$$

Its conditional update draws from masses proportional to $p(U \mid \rho_i)p_{\mathrm{g}}(\rho_i)$. Such a draw is exact for the discrete target, not for the continuous truncated prior. The applicable prior, support or grid, and fixed-versus-inferred status are specified with each experiment.

We use the shape–rate convention for $\mathrm{Gamma}(2, 2)$, with density $p(x) = 4x \exp(-2x)$ for $x > 0$.

In the headline controlled experiments, ρ, δ_B, and ϵ are fixed unless stated otherwise. Fixed execution parameters are conditioned on, and experiments using discrete execution-parameter priors use their separately stated specifications rather than the continuous priors above.

A path-conditional sweep applies

$$\mathrm{FFBS}(S, z) \longrightarrow (\pi, P) \longrightarrow U_{\mathrm{cond}} \longrightarrow (\rho, \vartheta), \tag{32}$$

with inactive coordinates omitted. At the stated cadence, the path-marginal transition and its immediate conditional refresh are inserted as one block:

$$U_{\mathrm{marg}} \longrightarrow \mathrm{FFBS}(S, z).$$

Proposal scales, update cadences, run lengths, and active coordinates are reported with the corresponding experiments. The auxiliary local relabel/split/merge/shift path kernel is not used in the reported fully latent runs; its implementation and tests are retained in the accompanying artifact.

Skill-label exchangeability holds only for permutations that preserve the fixed action supports and slot-specific priors. Under these symmetries, permutation-invariant summaries such as co-clustering, sorted skill usage, transition spectra, and relation counts avoid artificial disagreement due to relabelling. When action supports distinguish slots, their identities must also be respected in the diagnostics. Alignment to synthetic truth is used only for recovery evaluation, not to establish convergence.

B Preparing CPA Sequences for RPOS

Annotation source. The CPA annotation protocol and Retail codebook are documented by Li et al. (2026d). We retain here the interface required by RPOS rather than repeat the vocabulary-development and annotation-audit procedures. Each occurrence records a CPA label, the first agent event realizing it, and separate references to action realization and contextual evidence. Tool results provide context rather than additional agent actions. Several events may realize one occurrence, and one event may realize multiple occurrences.

Serialization. The stored annotations define the observed sequence

$$\mathbf{x}^{(n)} = \left(c_1^{(n)}, \ldots, c_{J_n}^{(n)}\right) \in \mathcal{A}^{J_n}, \tag{33}$$

where J_n counts CPA occurrences, not raw events. Occurrences are ordered by their action anchors, with repeated occurrences preserved. When several occurrences share an anchor, the implementation retains their recorded annotation order. This is a serialization convention, not observed precedence within the event. The likelihood conditions on this sequence; path marginalization sums over skill segmentations and assignments, not permutations of co-anchored occurrences.

Inputs and latent quantities. RPOS uses the serialized CPA labels. Event anchors, provenance, phases, and confidence metadata are retained for inspection but are not supplied as skill labels or dependency constraints. The annotation fixes action labels and occurrence boundaries, not skill-instance boundaries, skill assignments, or partial orders. Skill supports are specified separately by the model. Annotation uncertainty is not propagated into inference.

Compact Retail example. Training trace T46-3 contains 24 source events and 16 CPA occurrences. Credential requests at events 2 and 4 remain separate occurrences. The three order-retrieval calls at events 10, 12, and 14 form one annotated occurrence anchored at event 10. Event 16 contributes three occurrences: resolving the request referent, binding a parameter, and requesting mutation input. Their shared anchor does not establish a within-message ordering.

C Posterior Inference

Posterior inference for RPOS combines exact conditional path sampling with Gibbs and Metropolis–Hastings (MH) updates of the global parameters. Section C.1 describes the sampler; active parameters, proposal scales, and update schedules are specified separately for each experiment.

Table 3: **Components, literature connections, and scope.** CPA sequences, the skill count K, and skill-role interfaces are fixed inputs; boundaries, assignments, partial-order templates, and segment-level dynamics are inferred. The relation $r \succ_{H_k} s$ denotes template precedence, not a hard ordering constraint on every observed occurrence. No phase-guided seed segmentation, global instance-level partial order, or poset-substitution hierarchy is used.

Component	Mathematical object	Literature connection	Role and scope
Actions allowed in each skill	$\mathcal{A}_k \subseteq \mathcal{A}$, $m_k = \lvert\mathcal{A}_k\rvert$	Every skill has one graph node for each CPA label in the shared vocabulary.	Action supports and K are fixed before fitting. Each supported CPA label indexes one node in the skill template, reused by all occurrences of that label. In Retail, $\mathcal{A}_k = \mathcal{A}$ for every skill. An instance need not contain every supported action.
Reusable partial-order skill types	$U_k \in \mathbb{R}^{m_k \times d}$ $H_k = h(U_k)$ $r \succ_{H_k} s$ iff $U_{k,rj} > U_{k,sj}$ for all j $\mathcal{H} = (H_1, \ldots, H_K)$	Latent product-order representations for Bayesian precedence inference (Li et al., 2026a).	The shared order H_k is inferred through U_k, with d fixed and ρ fixed or inferred as specified. Roles are incomparable when neither utility row strictly dominates the other. Matrices inducing the same H_k have the same likelihood at fixed remaining parameters, but need not have equal prior density.
Recurrent execution within an instance	$\boldsymbol{q}_0 = \mathbf{0}$, $\boldsymbol{q}_u \in [0,1]^{m_k}$ $F_u(r)$, $Q_u(r)$, $C_u^{\text{back}}(r)$ $p_{\text{RFS}}(y_u \mid y_{<u}, H_k, \vartheta, \epsilon)$	Change propagation and build systems motivate tracking which outputs require updating after earlier inputs change (Acar et al., 2013; Mokhov et al., 2020).	The state is replayed deterministically from each candidate block's role history, not observed from the environment or sampled separately. Executing a role restores its own validity and attenuates downstream validity. This permits repetition and repair-like behaviour without adding nodes or cycles; the state is not a causal environment model.
Segmental skill dynamics	$S_n = (s_{n0}, \ldots, s_{nL_n})$ $w_{n\ell} = s_{n\ell} - s_{n,\ell-1}$ $D_{\min} \le w_{n\ell} \le D_{\max}$ z_n, π, P, with $P_{kk} = 0$	Semi-Markov models describe labelled variable-length segments (Sarawagi & Cohen, 2004; Kong et al., 2016). Options provide a related notion of temporal abstraction (Sutton et al., 1999).	Boundaries and skill labels are latent; π and P are learned under fixed duration support. Each trace concatenates contiguous instances, with state resets at boundaries. $P_{kk} = 0$ excludes adjacent same-type instances; contiguity excludes interleaved invocations.
Exact path inference and path-marginal structural updates	$e_{n,k}(a,b)$, $\alpha_n(j,k)$ $Z_n(U) = \sum_k \alpha_n(J_n, k)$ Length-conditioned trace likelihood: $Z_n(U)/C_{J_n}(\delta_B)$ FFBS draws of (S_n, z_n)	Semi-Markov dynamic programming (Sarawagi & Cohen, 2004; Kong et al., 2016). Here, path marginalization uses recurrent partial-order block likelihoods to score structural proposals.	At fixed global parameters, the recursion sums legal labelled paths and FFBS samples their conditional posterior. Structural MH includes the marginal score, prior, and proposal corrections. In our implementation, every marginal attempt, accepted or rejected, is immediately followed by FFBS before path-conditioned updates. This does not establish global posterior convergence.

Let

$$\Theta = (U, \theta, \pi, P, \rho), \qquad \theta = (\beta, \omega, \lambda_{\text{rep}}, \lambda_{\text{back}}),$$

and let the full sampled state, including trace-specific paths, be

$$\Omega = (S, z, \Theta) = (S, z, U, \theta, \pi, P, \rho).$$

Fixed coordinates are omitted from the sampled state and conditioned on throughout. Thus, when θ is fixed but ρ is inferred, the sampled state is

$$(S, z, U, \pi, P, \rho).$$

C.1 Posterior Updates

Define the complete-state *data* log likelihood

$$\ell_{\mathcal{X}}(\Omega) = \sum_{n=1}^{N} \sum_{\ell=1}^{L_n} \log e_{n,z_{n\ell}}(s_{n,\ell-1} + 1, s_{n\ell}), \tag{34}$$

where the block likelihoods are evaluated at the state's U and θ.

At fixed paths, $\ell_{\text{cond}}(\Theta; S, z) = \ell_{\mathcal{X}}(\Omega)$ is the likelihood term in the path-conditional updates. Path-marginal utility updates instead use $\ell_{\text{marg}}(U) = \sum_n \log Z_n(U)$, where $Z_n(U)$ is the semi-Markov forward sum and dependence on the remaining parameters is suppressed. Table 4 summarizes the updates, including their proposal corrections and recomputation requirements.

Discrete prior for the correlation update. The grid update in Table 4 uses a discrete prior, not a continuous Beta full conditional. On the fixed grid $\mathcal{G} = \{\rho_i\}_{i=1}^{399}$ in $[2.5 \times 10^{-3}, 1 - 2.5 \times 10^{-3}]$, define

$$p_{\text{g}}(\rho_i) = \frac{f_{\text{Beta}(1,1/6)}(\rho_i)}{\sum_{j=1}^{399} f_{\text{Beta}(1,1/6)}(\rho_j)}, \qquad w_i = \frac{p(U \mid \rho_i) p_{\text{g}}(\rho_i)}{\sum_{j=1}^{399} p(U \mid \rho_j) p_{\text{g}}(\rho_j)}. \tag{35}$$

The Gaussian prior is evaluated as

$$p(U \mid \rho) = \prod_{k=1}^{K} \prod_{c \in \mathcal{A}_k} \varphi_d(\boldsymbol{U}_{k,c}; \mathbf{0}, \Sigma_\rho), \qquad \Sigma_\rho = (1 - \rho) I_d + \rho \mathbf{1}_d \mathbf{1}_d^\top,$$

where φ_d denotes the multivariate normal density. Normalize the weights in the log domain and draw the grid index. This is exact for the stated discrete prior, not a continuous truncated Beta prior. Each experiment must identify its actual prior specification.

D Evaluation Metrics and Reporting Conventions

Reference correctness. We compare forward normalizers and marginals with exhaustive calculations, FFBS frequencies with the exact path posterior using total-variation distance, and composed transitions with finite-state stationarity checks. These assess implementation correctness, not recovery or convergence (Appendix E.1).

Structure recovery. We report exact-library retained-draw frequency, closure Hamming distance, and relation and incomparable-pair F_1 where applicable. Canonicalization and recovery-only alignment respect fixed action supports and priors. The alignment and aggregation rules are specified per experiment.

Segmentation and assignments. Boundary recovery uses AUROC and Brier score; occurrence partitions use ARI and recovery-aligned accuracy. Co-clustering Brier scores use the specified occurrence-pair set and distinguish same-instance from same-type membership. Brier improvement alone does not establish calibration.

Table 4: **Posterior update kernels for RPOS.** Only parameters inferred in a given experiment are updated; fixed parameters, priors, and proposal scales are specified with that experiment. FFBS and Dirichlet updates are exact conditional draws. MH updates include the proposal correction c_θ defined below. Partial orders and validity states are computed deterministically. Rows group updates by object rather than execution order.

Object	Conditional draw or proposal	Acceptance and recomputation
Labelled paths and segment-level dynamics		
S_n, z_n	**Exact FFBS.** Draw the final label with probability $\alpha_n(J_n, k)/Z_n$, then sample preceding boundaries and labels from the forward summands.	Joint conditional draw given $x^{(n)}, U, \pi, P, \delta_B, \vartheta, \epsilon$. No accept/reject step.
π	**Dirichlet Gibbs.** $\pi \sim \mathrm{Dir}(a^\pi + c^{\mathrm{init}})$.	c^{init} counts the first segment label of each current trace.
$P_{h,-h}$	**Row-wise Dirichlet Gibbs.** $P_{h,-h} \sim \mathrm{Dir}(a^P_{h,-h} + C_{h,-h})$; $P_{hh} = 0$.	$C_{h,-h}$ counts transitions between successive sampled skill instances.
Reusable structure: two distinct utility-update kernels		
$U_{k,r}$	**Path-conditional MH.** $\boldsymbol{U}'_{k,r} = \boldsymbol{U}_{k,r} + \sigma_U \boldsymbol{\xi}$, $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I_d)$; $\sigma_U = 0.5$ and $c_U = 0$.	Log ratio: $\Delta\ell_{\mathrm{cond}} + \Delta \log p(U_k \mid \rho)$. Recompute $H'_k = h(U'_k)$ and replay all current k-blocks, each from $\boldsymbol{q}_0 = \mathbf{0}$.
$U_{k,r}$	**Path-marginal MH.** Use the same symmetric row proposal; $c_U = 0$.	Log ratio: $\Delta\ell_{\mathrm{marg}} + \Delta \log p(U_k \mid \rho)$. Recompute affected block scores and forward sums without using stored paths. **Immediately redraw all paths by FFBS after acceptance or rejection.**
Recurrent parameters and utility correlation		
β, λ_{rep}, λ_{back}	**Separate log-scale MH updates.** For each positive scalar θ, $\theta' = \theta e^{s_\theta \xi}$, $\xi \sim \mathcal{N}(0,1)$; $c_\theta = \log\theta' - \log\theta$. Each has prior $\mathrm{Gamma}(2,2)$, shape rate.	Log ratio: $\Delta\ell_{\mathrm{cond}} + \Delta \log p(\theta) + c_\theta$. Re-score the corresponding progress, repetition or disruption weights in current blocks. The feature cache at fixed (U, ω) remains valid; no validity replay.
ω	**Random-walk MH on $\mathbb{R}$.** $\omega' = \omega + s_\omega \xi$, $\xi \sim \mathcal{N}(0,1)$; $c_\omega = 0$. Prior $\mathcal{N}(0, 2^2)$.	Log ratio: $\Delta\ell_{\mathrm{cond}} + \Delta \log p(\omega)$. Replay each current block from $\boldsymbol{q}_0 = \mathbf{0}$ and rebuild its validity-dependent features.
ρ (if active)	**Discrete Gibbs on a fixed grid.** Let $\{\rho_i\}_{i=1}^{399}$ be equally spaced on $[0.0025, 0.9975]$, with prior masses $p_i \propto f_{\mathrm{Beta}(1,1/6)}(\rho_i)$. Form $w_i \propto p(U \mid \rho_i) p_i$ and normalize. Draw $I \sim \mathrm{Categorical}(\boldsymbol{w})$, then set $\rho = \rho_I$.	Exact conditional draw under the discrete-grid prior; no MH correction or accept/reject step. Here $p(U \mid \rho) = \prod_{k,r} \mathcal{N}_d(\boldsymbol{U}_{k,r}; \mathbf{0}, \Sigma_\rho)$, where $\Sigma_\rho = (1-\rho) I_d + \rho \mathbf{1}\mathbf{1}^\top$. No block likelihoods or validity states need recomputation.
$H, \boldsymbol{q}$	**Deterministic.** $H = h(U)$; compute $\boldsymbol{q}_u$ by replaying the observed block from $\boldsymbol{q}_0 = \mathbf{0}$.	Derived from the utilities, ω, and the block's role sequence. No separate posterior draw.

Notation. $\Delta f = f_{\mathrm{proposed}} - f_{\mathrm{current}}$; $c_\theta = \log q(\theta \mid \theta') - \log q(\theta' \mid \theta)$ is added once to the log target ratio. MH accepts with probability $\min\{1, \exp(\log R)\}$. Here ℓ_{cond} is the complete-state data log likelihood and $\ell_{\mathrm{marg}}(U) = \sum_n \log Z_n(U)$. The three positive scalars are updated individually, not as one joint proposal.

Predictive performance. Held-out negative log likelihood per CPA occurrence is

$$\mathrm{NLL}_{\mathrm{occ}} = -\frac{\sum_{n \in \mathcal{D}_{\mathrm{test}}} \log \widehat{p}(\boldsymbol{x}^{(n)} \mid J_n, \mathcal{D}_{\mathrm{train}})}{\sum_{n \in \mathcal{D}_{\mathrm{test}}} J_n}.$$

MCMC-based predictors average complete-trace probabilities over the retained draws specified by each protocol before taking logarithms. Differentiable predictors condition on a fitted global state; across-start summaries average NLLs rather than probabilities. Aggregation across fitted runs, tasks, and workflows follows each experiment's protocol. Scoring marginalizes labelled paths and retains the length-dependent normalizer. Task/workflow aggregation and paired bootstrap 95% intervals follow each experiment's protocol. Across-run standard deviations are not confidence intervals; intervals conditional on fixed fits exclude refitting uncertainty.

Parameter recovery. We report posterior means, standardized errors, correlations, and inclusion of generating values in 50%, 80%, 90%, and 95% credible intervals. Single-dataset inclusion is not repeated-simulation coverage. Generating-parameter predictions are scale references, not bounds.

Convergence. We report available rank-normalized split $\widehat{R}$, bulk and tail ESS, and Monte Carlo standard errors using symmetry-respecting numerical summaries. Failed diagnostics restrict conclusions to finite-run evidence; stable or constant summaries do not establish global convergence.

External and real-data evaluation. TaskBench supports structure recovery, held-out prediction, and AUROC for legal versus dependency-violating executions. Retail lacks reference skill boundaries and libraries, so evaluation concerns prediction, reuse, uncertainty, and stability. Qualitative skill cards and external procedural-constraint audits do not establish latent-skill recovery.

D.1 CompILE-CPA Baseline

CompILE (Kipf et al., 2019) learns segment boundaries and reusable latent codes from unsegmented demonstrations. We adapt its official implementation to CPA sequences, using categorical codes without environment states or ground-truth segmentation. We distinguish the learned model, CompILE-CPA, from the separate semi-Markov model used for prediction, CompILE-CPA+SM.

Segmentation and recognition. For a trace of length J, an ordered segmentation with M available slots is written as

$$1 = b_0 \le b_1 \le \cdots \le b_M = J + 1, \qquad z_i \in \{1, \ldots, K_{\mathrm{C}}\}. \tag{36}$$

Segment i contains positions $b_{i-1}, \ldots, b_i - 1$; coincident boundaries represent an empty segment. The slot budget M limits the number of segment instances, whereas K_{C} counts reusable code types.

A shared 32-unit LSTM has separate boundary and code heads. Its hidden and cell states are masked using preceding boundary distributions. Raw boundary predictions need not be ordered: slot i uses $[\max_{0 \le j < i} b_j, b_i)$ and is empty when its end does not exceed its start. Deterministic decoding retains the resulting nonempty segments.

The per-trace slot budget is $M_J = \max\{1, \lfloor J/w_{\min} \rfloor\}$, with values given in Table 5. This budget does not impose the hard duration constraints used by the separate predictive scorer. Native recognition also allows adjacent instances with the same code.

CPA block decoder. Unlike the original state-conditioned imitation policy, CompILE-CPA predicts action labels using a first-order decoder. Each code k has an initial-action distribution B_k and transition matrix T_k, shared across segments and traces. For a nonempty block with exclusive end position b,

$$e_k(x_{a:b-1}) = B_k(x_a) \prod_{t=a+1}^{b-1} T_k(x_{t-1}, x_t). \tag{37}$$

The decoder resets to B_k at each block start. Categorical rows use smoothing $\epsilon = 0.02$.

Training and batching. We train with Adam at learning rate 10^{-3} and Gumbel–Softmax temperatures of 1 for boundaries and codes. The objective combines reconstruction with code and boundary KL penalties, each weighted by 0.1.

For Retail and the joint-recurrent synthetic study, each update samples 16 training traces without replacement. These fits optimize all encoder and decoder parameters without gradient clipping, using one CPU thread and float64 precision. For initialization seed s, minibatch sampling and Gumbel draws use seeds $s + 10{,}000$ and $s + 20{,}000$, respectively.

For TaskBench, each update samples $\min(16, n_{\mathrm{train}})$ traces uniformly without replacement from the workflow's training set. With six training traces, every update uses all six once, in randomly permuted order. With 24 traces, each update samples a subset of 16. Sampling is repeated at each update, so traces may be reused across updates.

Table 5: **CompILE-CPA settings.** The Retail column describes the $K_C = 5$ configuration. Duration constraints apply only to +SM scoring. TaskBench counts are per workflow in the 48-workflow experiment, which provides 3,456 held-out traces in total.

Setting	Retail	Joint synthetic	TaskBench
Action vocabulary size R	24	5	4–7
Codebook size K_C	5	3	3
Segment-slot budget M_J	$\max(1, \lfloor J/2 \rfloor)$	$\max(1, \lfloor J/3 \rfloor)$	J
+SM segment widths	2–12	3–12	1–12
Training traces	372	100	6 or 24
Training occurrences	5,981	3,600	Varies by workflow
Evaluation traces	64 (development)	45 (held-out)	72 (held-out)
Evaluation occurrences	1,104	1,608	Varies by workflow
Fitting seeds	$11, 29, 47$	$11, 29, 47, 71$	$11, 29, 47$
Batch size	16	16	$\min(16, n_{\text{train}})$
Per-fit update limit	20,000	6,000	5,000
Per-fit wall-clock limit	3,600 s	900 s	None

Stopping and checkpoint selection. Retail and joint-recurrent synthetic fits have both update and wall-clock limits (Table 5). TaskBench uses 5,000 updates without a wall-clock limit; all 288 fits in the 48-workflow experiment completed this budget. We retain the last completed update, without selecting checkpoints by training loss, development NLL, or recovery metrics. These stopping budgets do not establish convergence.

Timing. For the timed studies, the stopping clocks include preparation, optimization, and in-loop checkpoints. Time limits are checked between updates, so an update and its associated checkpointing may finish after the limit.

Retail terminal exports are timed separately from the optimization loop. Training-only composition estimation is included in the final-export time and reported as a component, not an additional cost. The joint-recurrent synthetic runner includes terminal exports and training-only composition estimation in its recorded total fitting time, including any overrun.

Normalized sequence prediction. CompILE-CPA+SM combines the learned block decoder with a separate semi-Markov composition model. We estimate the initial code probabilities π and transition probabilities P from deterministic training recognition paths after removing empty segments, using a pseudocount of 0.5 per supported category. For transition estimation, same-code transitions are excluded and each row is normalized over off-diagonal entries, imposing $P_{kk} = 0$. This restriction applies to the scorer, not to native recognition.

The scorer sums exactly over all labelled segmentations allowed by the study-specific duration constraints, using the block likelihood in Equation (37) and the normalized segmentation prior

$$p(S \mid J, \delta_B) = \frac{\delta_B^{L-1}(1-\delta_B)^{J-L}}{C_J(\delta_B)}, \qquad \delta_B = 0.15,$$

for a legal segmentation S with L nonempty blocks. Here, C_J sums the numerator over all legal segmentations. Decoder and composition parameters remain fixed during evaluation; the recognition network is not used for predictive scoring. NLL is the summed negative trace log probability divided by the total number of CPA occurrences, with aggregation across fits and workflows specified by each experiment. The duration constraints and composition prior are additional modelling choices, so this score is neither the original CompILE likelihood nor its training objective.

Recovery metrics and comparison scope. The primary synthetic comparison derives boundary F_1 and occurrence-marginal ARI from the exact +SM conditional path marginals. Metrics from native deterministic recognition are reported separately. CompILE-CPA represents skills through categorical codes and first-order decoders, not partial-order graphs, so relation F_1 is not applicable. Both methods learn representations shared across traces, but their model families, capacities, objectives, and inference procedures differ. This comparison therefore does not isolate the effect of partial versus total orders.

Table 6: **Implementation checks.** Maximum absolute differences, except FFBS total-variation distance from 10^5 draws.

Quantity	Reference	Difference
Forward log normalizer	Exhaustive path sum	8.88×10^{-16}
Boundary marginals	Exhaustive path sum	4.9×10^{-17}
Skill-label marginals	Exhaustive path sum	5.6×10^{-16}
FFBS path distribution	Exact path posterior	2.39×10^{-3} TV
Recurrent block likelihood	Independent replay	$< 10^{-9}$
Path-marginal MH + FFBS	Exact joint target	2.2×10^{-16}
Stale-path negative control	Exact joint target	2.9×10^{-3}

Table 7: **Oracle recovery.** The two conditions use the same oracle-study corpus. NLL is in nats per CPA occurrence.

Metric	Result
Known structures: infer (S, z)	
Boundary AUROC, all / held-out traces	0.956 / 0.966
Boundary ECE, pooled (oracle model)	0.008
Occurrence-level modal accuracy	0.898
Per-trace assignment ARI	0.747
Correct MAP segment count	70.3%
Exact MAP labelled path	20.7%
Median probability of the generating path	0.048
Known paths: infer H	
Closure / incomparable-pair F_1	1.000 / 1.000
Exact-library retained-draw frequency	1.000
Held-out oracle-path block-emission NLL	1.0772
Antichain / total-order NLL	1.7528 / 4.0398

E Synthetic Validation

These results complement the path-marginalization comparison in Section 5.1. Oracle recovery, joint recurrent recovery, and transfer use separate model-matched corpora.

E.1 Reference-Based Validation

Forward probabilities and marginals agree with exhaustive path sums to floating-point precision (Table 6). The composed MH–FFBS transition preserves the enumerated target; inserting a path-conditioned update before the refresh produces a detectable stationarity error.

E.2 Oracle Component Recovery

On a separate matched corpus, supplying the generating library, execution parameters, and (π, P) gives held-out boundary AUROC 0.966. Conversely, supplying the generating training paths, execution parameters, and ρ yields exact library recovery in every retained draw (Table 7). The latter NLL averages block log losses over retained structure draws using true held-out boundaries and skill labels, not marginalized paths.

E.3 Joint Recovery with Unknown Execution Parameters

On a new corpus of 100 training and 45 held-out traces, we infer $(S, z, U, \pi, P, \beta, \omega, \lambda_{\text{rep}}, \lambda_{\text{back}})$, keeping $K, \rho, \delta_B, \epsilon$, the vocabulary, and action supports fixed. Four chains each use 50,000 warm-up and 100,000 production sweeps, retaining 20,000 draws per chain.

The generating library is modal in every chain and occurs in $79{,}569/80{,}000$ retained draws (99.46%). Its aligned modal closure has relation and incomparable-pair $F_1 = 1.000$ and Hamming distance $0/60$. Composition errors are $\mathrm{TV}(\pi, \pi^\star) = 0.0495$ and $\|P - P^\star\|_F = 0.0921$. Normalized held-out

Table 8: **Execution-parameter recovery.** Retained-draw means and SDs on one corpus. $E_{\mathrm{std}} = (\mathrm{mean} - \mathrm{truth})/\mathrm{SD}$. Y/N records truth inclusion, not repeated-simulation coverage. Width ratios compare joint and oracle-conditioned 95% intervals.

Parameter	Truth	Mean	SD	E_{std}	Inclusion 50/80/90/95%	Width ratio
β	1.5000	1.5575	0.1135	$+0.51$	Y/Y/Y/Y	1.27
ω	1.7346	1.4906	0.2563	-0.95	N/Y/Y/Y	1.11
λ_{rep}	0.8000	0.9237	0.0722	$+1.71$	N/N/N/Y	1.20
λ_{back}	0.2500	0.2407	0.0568	-0.16	Y/Y/Y/Y	1.11

Table 9: **Structural diagnostics.** Production draws. The final row uses arbitrary numerical category codes and is not a coding-independent library ESS.

Summary	$\widehat{R}$	Bulk ESS	Tail ESS
`total_relations`	1.0222	95.1	–
`sorted_relation_counts[1]`	1.0180	116.2	–
`sorted_relation_counts[2]`	1.0222	95.0	–
Library category code (historical)	1.0221	95.6	95.0

NLL is 1.3140, versus 1.3136 for the generating parameters and 1.3137 with oracle training paths; all three predictors marginalize held-out paths. The joint-minus-generating paired difference is 0.0149 nats per trace (95% interval $[-0.0727, 0.0924]$).

All four 95% execution-parameter intervals contain their generating values (Table 8). The correlation between β and λ_{rep} is 0.787, and remains 0.705 when generating paths, structure, and composition are supplied.

Predictive checks and diagnostics. Four of six behavioural summaries fall within the reported predictive intervals; mean segment length and revisitations per block do not. Execution-parameter diagnostics satisfy $\widehat{R} \leq 1.006$ and bulk ESS ≥ 1130, but structural criteria are not met (Table 9) and path diagnostics are unavailable. Recovery frequencies remain finite-run summaries.

E.4 Frozen-Model Transfer to Unseen Skill Compositions

Five independently generated RPOS libraries each contain three skills over five shared CPA labels ($d = 2$, $\rho = 0.5$). Each provides 120 training traces using the six $i \to j \to i$ compositions ($i \neq j$), plus 60 familiar and 60 unseen test traces; unseen compositions use distinct $i \to j \to k$ triples. All adjacent transitions occur in training. Segment lengths are 3–12 and matched across splits, with no shared complete action sequences. Non-oracle models receive neither boundaries nor skill labels, and all parameters are frozen at test time.

Fitting and scoring. RPOS and total-order RPOS use four 20,000-sweep chains, discarding 1,000 sweeps and thinning by five. Total-order RPOS ($d = 1$) changes both structural support and prior. Two-stage RPOS fixes SEQ-HSMM training MAP paths; the oracle fixes true training paths. These controls use 4,000 sweeps, discard 500, and thin by five. Prediction marginalizes test paths and retains C_J. Complete-trace probabilities are averaged before taking logs, using 100 draws per MCMC chain and equal weights across chains or optimization seeds. NLL is occurrence-weighted within each library and averaged equally across libraries. Paired intervals use 10,000 library-bootstrap resamples and exclude MCMC error.

RPOS has lower unseen NLL than every non-oracle baseline on all five libraries (Table 10). Its source closure F_1 is 0.8588 and unseen assignment accuracy is 0.7629, using source-only label alignment. Unseen boundary F_1 is 0.3455, versus 0.3841 for total-order RPOS; its paired difference from SEQ-HSMM includes zero (Figure 6). Differences between methods' familiar-to-unseen NLL changes are also unresolved.

Table 10: **Prediction without retraining.** Mean NLL in nats per action across five libraries. Δ = unseen minus familiar NLL; D = RPOS minus comparator on unseen compositions. Intervals are paired 95% library-bootstrap intervals.

Method	Familiar	Unseen	Δ	D	95% interval for D
RPOS	1.3888	1.3685	−0.0202	–	–
Total-order RPOS	1.5590	1.5281	−0.0308	−0.1596	[−0.1687, −0.1474]
SEQ-HSMM	1.4302	1.4003	−0.0299	−0.0318	[−0.0447, −0.0189]
CompILE-CPA+SM	1.5253	1.5008	−0.0244	−0.1323	[−0.1541, −0.1125]
Two-stage	1.4314	1.4143	−0.0171	−0.0458	[−0.0768, −0.0222]
Oracle-trained	1.3750	1.3549	−0.0201	+0.0136	[+0.0104, +0.0169]
Bigram	1.4516	1.4270	−0.0246	−0.0584	[−0.0746, −0.0431]

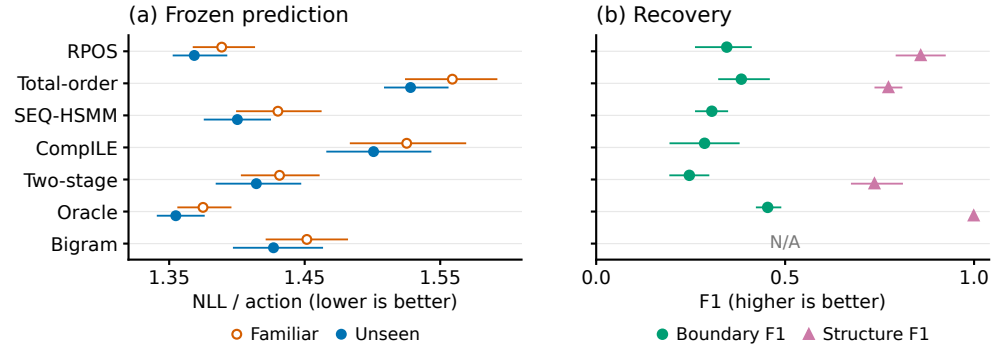


Figure 6: **Frozen prediction and recovery.** Means and 95% library-bootstrap intervals. Recovery uses unseen boundary F_1 and source closure F_1, with source-only label alignment. Missing markers denote unavailable results; intervals exclude MCMC error.

F TaskBench Workflow Evaluation

F.1 Data and Fitting

We use 48 TaskBench Multimedia workflows (4–7 actions; 193 incomparable pairs; underlying undirected graphs are forests). Each is fitted separately with one skill ($K = 1$) and each complete trace as one known instance. RPOS infers structure and execution parameters, not segmentation. Learners receive only action identities and sequences. The intersection follow-up reuses existing data and compatible fits, without evaluating independently sampled training.

An independent version-tracking simulator requires valid, current inputs. Normal traces have no returns; recurrent traces have 4–8. Benign returns preserve outputs; repair traces include at least two invalidations and one preserving return, and may interrupt unfinished repairs. Execution ends with all returns completed and outputs valid. Nested training sets contain two or eight traces per condition ($n = 6, 24$). Gold-guided first-occurrence orders cover both directions of every incomparable pair: training is coverage-controlled, not IID. Independent random streams generate 3,456 held-out traces, 24 per workflow and condition.

Recurrent models use $d = 3$, $\rho = 0.5$, $\beta = 0$, $\epsilon = 0.02$, $\lambda_{\text{back}} = 0.5$, and a uniform prior on $\theta = (\kappa, \lambda_{\text{rep}}) \in \mathcal{G} = \{0.25, 0.75, 1\} \times \{1, 3, 6\}$. Here κ controls invalidation and scales return cost; the ablation fixes $\kappa = 0$ with the same repetition prior. Endpoints are evaluated directly. CompILE-CPA+SM and SEQ-HSMM use three segment types, first-order block emissions, widths 1–12, and seeds 11, 29, 47. CompILE uses a 32-unit masked LSTM and 5,000 Adam updates at 10^{-3}; SEQ-HSMM uses MAP-EM, Dirichlet(1.5) priors, and at most 200 iterations. Bigram and IID integrate Dirichlet(0.5) predictions. No evaluation-based seed or checkpoint selection is used.

Table 11: **Recurrent study on 48 workflows.** (a) Exact closure counts, mean relation F_1, and mean directed errors. (b) Length-conditioned NLL/action, averaged equally over workflows; Repeat denotes benign repetition.

	$n = 6$			$n = 24$		
(a) Closure recovery						
Method	Exact	F_1	Errors	Exact	F_1	Errors
RPOS	47/48	0.9981	0.0208	48/48	1.0000	0.0000
RPOS, $\kappa = 0$	48/48	1.0000	0.0000	48/48	1.0000	0.0000
Intersection	48/48	1.0000	0.0000	48/48	1.0000	0.0000
(b) Predictive NLL						
Method	Normal	Repeat	Repair	Normal	Repeat	Repair
RPOS	0.6906	1.3044	1.2814	0.6886	1.2856	1.2713
RPOS, $\kappa = 0$	0.7094	1.2559	1.3590	0.7142	1.2386	1.3433
Intersection+RFS	0.6654	1.2950	1.2708	0.6886	1.2855	1.2713
Intersection+RFS, $\kappa = 0$	0.6837	1.2433	1.3498	0.7143	1.2386	1.3433
CompILE-CPA+SM	1.0334	1.5878	1.4769	0.9534	1.6010	1.4398
SEQ-HSMM	1.0776	1.4223	1.3657	0.9645	1.4094	1.3263
Bigram	1.0961	1.5189	1.4010	1.0154	1.4734	1.3374
IID	1.5747	1.5246	1.5353	1.5472	1.5089	1.5178

F.2 First-Occurrence Intersection Controls

For training set $\mathcal{D}_n$, let $t_x(a)$ be action a's first position in trace x. Intersection retains precedence shared by all traces:

$$(a, b) \in \widehat{H} \iff a \neq b \text{ and } t_x(a) < t_x(b) \quad \text{for all } x \in \mathcal{D}_n. \tag{38}$$

Missing actions trigger a completeness failure. The intersection is a strict partial order without thresholding. Valid first-occurrence orders preserve true relations; two-way coverage excludes false ones, guaranteeing exact closure recovery in the recurrent study. The violation score $s_{\text{int}}(x) = -\sum_{(a,b)\in\widehat{H}} \mathbf{1}\{t_x(a) > t_x(b)\}$ ranks repeat-free sequences but has no NLL.

Intersection+RFS uses the same recurrent frontier-state likelihood with $\widehat{H}$ fixed. Execution weights are fitted from training data only:

$$w_\theta \propto p(\theta) \prod_{x\in\mathcal{D}_n} p(x \mid |x|, \widehat{H}, \theta), \qquad \sum_{\theta\in\mathcal{G}} w_\theta = 1, \tag{39}$$

$$p(x \mid |x|, \mathcal{D}_n, \widehat{H}) = \sum_{\theta\in\mathcal{G}} w_\theta p(x \mid |x|, \widehat{H}, \theta). \tag{40}$$

The nine settings are enumerated exactly (three for $\kappa = 0$), without reusing RPOS parameter estimates. Log-sum-exp calculations average probabilities before logarithms. RFS normalizes over all strings of the given length, without a segmentation normalizer or restriction to legal orders. Segmental baselines retain their own length normalizers.

F.3 Recovery and Recurrent Prediction

NLL is occurrence-weighted within workflow/condition, then averaged equally across workflows. Prediction averages complete-trace probabilities over RPOS draws or three baseline fits. RPOS graphs use a $1/3$ threshold, descending-probability insertion with cycle rejection and node-index ties, then closure. Intersection recovers all closures and predicts better at $n = 6$; at $n = 24$, paired differences are below 0.00005 nats/action in magnitude and their intervals include zero (Tables 11 and 13). Invalidation helps repair but harms benign-repeat prediction. A $1/2$ threshold or whole-graph mode reduces RPOS's $n = 6$ recovery to $46/48$; other counts are unchanged.

Table 12: **Ordering-combination prediction.** NLL is per action; Δ is unseen minus seen NLL. AUROC ranks unseen valid orders above invalid controls. Metrics use scheduler weights and equal workflow averages.

	$n = 6$			$n = 24$		
Method	Unseen	Δ	AUROC	Unseen	Δ	AUROC
(a) All 30 workflows						
RPOS	0.5529	−0.1730	0.9967	0.6174	−0.1920	0.9950
RPOS, $\kappa = 0$	0.5538	−0.1665	0.9965	0.6104	−0.1808	0.9931
Intersection+RFS	0.6466	−0.2220	0.9915	0.6276	−0.1889	0.9949
Intersection+RFS, $\kappa = 0$	0.6347	−0.2132	0.9558	0.6171	−0.1797	0.9592
Total-order RFS	2.3051	0.7978	0.9669	2.3133	0.7942	0.7231
CompILE-CPA+SM	1.5358	−0.1859	0.8535	1.5143	−0.1529	0.8926
SEQ-HSMM	1.3468	−0.0314	0.9337	1.3660	−0.1035	0.9476
Bigram	1.3541	−0.0771	0.8127	1.5262	−0.1529	0.8646
IID	1.5295	0.0000	0.5000	1.4884	0.0000	0.5000
Intersection violation score	–	–	0.9870	–	–	0.9906
(b) Fixed 22-workflow subset						
RPOS	0.4802	−0.2129	1.0000	0.4521	−0.3187	1.0000
RPOS, $\kappa = 0$	0.4805	−0.2096	1.0000	0.4523	−0.3030	1.0000
Intersection+RFS	0.4611	−0.3737	1.0000	0.4520	−0.3271	1.0000
Intersection+RFS, $\kappa = 0$	0.4610	−0.3565	1.0000	0.4522	−0.3096	1.0000
Total-order RFS	2.3027	0.6042	0.9679	2.2990	0.5977	0.6723
CompILE-CPA+SM	1.5179	−0.2188	0.9574	1.5075	−0.1805	0.9437
SEQ-HSMM	1.3470	−0.0324	0.9804	1.3645	−0.1109	0.9817
Bigram	1.3765	−0.0829	0.8468	1.5627	−0.1785	0.9147
IID	1.5099	0.0000	0.5000	1.4690	0.0000	0.5000
Intersection violation score	–	–	1.0000	–	–	1.0000

F.4 Generalization to Unseen Ordering Combinations

Thirty workflows admit two incomparable pairs with four legal joint orientations. One combination is withheld; each other combination supplies two or eight training draws ($n = 6, 24$). Nested draws are sampled *with replacement* from the uniform-ready-action scheduler, conditioned on the combination and reserved training support. All traces are complete and repeat-free; n counts multiplicity, with 3–5 and 3–7 distinct strings, respectively.

Seen tests reserve disjoint strings from training combinations with spare support; unseen tests cover the withheld combination's full legal support. Both categories are enumerated and weighted by the scheduler's conditional probabilities, not uniformly over permutations. Nearest-Kendall invalid controls preserve both selected orientations and violate exactly one true relation; ties share the base order's weight. NLL uses expected log loss per action; weighted AUROC gives half credit to ties. Both average workflows equally. Total-order RFS (the total-order RPOS control) exactly integrates all $m!$ labelled total orders and nine execution settings under a uniform joint prior, with weights fitted to training likelihoods.

Although both selected pairs have two-way coverage, unselected pairs introduce unseen orientations in eight workflows at $n = 6$ and seven at $n = 24$. The fixed 22-workflow subset requires every orientation needed by unseen valid orders to appear in training at both budgets. Valid orders then violate no intersection constraints, whereas invalid controls do; intersection AUROC 1 follows from this condition, not exact graph recovery.

Table 12 reports ordering results. RPOS and intersection have identical point closures in all 60 workflow–budget cases: $13/30$ and $15/30$ are exact ($13/22$ and $14/22$ on the subset). RPOS has lower mean unseen NLL on the full set, but Intersection+RFS is better on the subset (Table 13). Their full-set unseen-minus-seen gap differences have intervals containing zero. Thus, the advantage concerns predictive distributions, not point recovery or compositional novelty specifically. RPOS still beats total-order and sequence models on the subset; the no-invalidation ablation performs similarly on this repeat-free task.

Table 13: **RPOS minus Intersection+RFS NLL.** Negative differences favor RPOS; intervals are paired 95% workflow-bootstrap intervals conditional on fits and evaluation data.

Study (workflows)	n	Condition	Difference	Lower	Upper
Main, 48	6	Normal	0.025198	0.018104	0.033160
Main, 48	6	Benign repeat	0.009366	0.005510	0.014126
Main, 48	6	Repair	0.010591	0.007085	0.014452
Main, 48	24	Normal	−0.000017	−0.000110	0.000071
Main, 48	24	Benign repeat	0.000048	−0.000005	0.000114
Main, 48	24	Repair	0.000031	−0.000043	0.000102
Composition, 30	6	Unseen NLL	−0.093765	−0.172078	−0.025805
Composition, 30	24	Unseen NLL	−0.010216	−0.018586	−0.003487
Sensitivity, 22	6	Unseen NLL	0.019174	0.013327	0.025405
Sensitivity, 22	24	Unseen NLL	0.000089	0.000031	0.000154

F.5 Uncertainty and Additional Checks

RPOS uses four chains: 1,000 burn-in sweeps, thinning by five, and two consecutive diagnostic passes or a 64,000-sweep cap. Passes require rank-normalized $\widehat{R} \leq 1.01$, bulk ESS at least 400 for scalars and 200 for relations, and tail ESS at least 200. Across RPOS and its ablation, 187/192 recurrent fits and 119/120 ordering fits pass; all are included. Identical constant summaries are excluded from numerical gates but do not establish mixing. Predictions are finite-run estimates. Pointwise 95% intervals use 10,000 paired workflow-bootstrap resamples, not chains or draws, and exclude MCMC, optimization, and data-regeneration uncertainty.

Repair-timing follow-up. Specified after the main results but before new fitting, this study uses six repair traces per workflow. A counting dynamic program uniformly samples benign controls matching action counts and first-occurrence order; normal controls execute that order once. RPOS and its ablation recover 48 and 48 closures on normal traces, 43 and 48 on benign traces, and 46 and 47 on repairs. RPOS's repair-minus-benign recovery gain is 0.0625 (95% interval $[0, 0.1458]$); subtracting the ablation gain gives 0.0833 ($[0.0208, 0.1667]$). The primary interval includes zero; the interaction does not establish better absolute recovery. All 288 fits are included, with 277 passing diagnostics. Three of four interaction-contributing workflows have unresolved exploration. This tests timing, not held-out prediction.

G Retail Evaluation

G.1 Data, Fitting, and Prediction

The task-disjoint split contains 372 training traces from 93 tasks and 64 development traces from 16 tasks. Twenty additional traces from five tasks are scored with frozen training-only fits, without test-based selection or refitting. The modelling export contains 7,085 occurrences over 24 CPA labels. It combines v0.5/v1.0 annotations with unchanged definitions, retains one annotation stream per trace, preserves occurrence order and multiplicity, and includes only `MATCH_EXISTING` records. The export and split are identified by `MCMC_INPUT_MANIFEST.json` and `MAIN_TABLE_SPLIT.json`. Since 27 development traces contributed to vocabulary development or auditing, development scores are internal comparisons.

RPOS uses all 24 actions in each skill, $d = 4$, widths 2–12, $\delta_B = 0.15$, and fixed execution parameters $(\beta, \omega, \lambda_{\text{rep}}, \lambda_{\text{back}}, \epsilon) = (1.5, \text{logit}(0.85), 0.8, 0.25, 0.02)$. Gaussian utility rows condition on ρ, whose prior has $\text{Beta}(1, 1/6)$-density-weighted masses on 399 equally spaced points in $[0.0025, 0.9975]$. Initial and off-diagonal transition probabilities have $\text{Dirichlet}(1)$ priors, with $P_{kk} = 0$. Four chains per capacity run for 30,000 sweeps, discarding 3,000 and retaining every fifth draw. Each sequence baseline uses three fits per capacity; computational budgets differ.

Prediction sums over labelled segmentations and retains the length normalizer. Within each RPOS chain, complete-trace probabilities are averaged before taking logarithms. NLL is normalized by total CPA occurrences; means and SDs summarize fit-level scores, retaining every evaluated fit. RPOS's

Table 14: **Retail prediction.** Length-conditioned NLL per CPA occurrence (↓) on 64 development and 20 final-test traces. Parentheses give across-fit SDs where reported; – denotes an unreported score or an inapplicable capacity.

Method	K	Development	Final test
RPOS	4	1.821 (0.024)	1.627
RPOS	5	1.795 (0.021)	1.600 (0.043)
RPOS	6	1.790 (0.037)	1.615
RPOS	8	1.737 (0.022)	1.590 (0.013)
SEQ-HSMM	4	1.471 (0.015)	–
SEQ-HSMM	5	1.466 (0.008)	1.305 (0.025)
SEQ-HSMM	6	1.460 (0.004)	–
SEQ-HSMM	8	1.473 (0.004)	–
CompILE-CPA+SM	4	1.833 (0.166)	1.629 (0.167)
CompILE-CPA+SM	5	1.781 (0.169)	1.658 (0.249)
CompILE-CPA+SM	6	1.787 (0.082)	–
CompILE-CPA+SM	8	1.861 (0.122)	–
Bigram	–	–	**1.303**

mean development NLL decreases from 1.821 at $K = 4$ to 1.737 at $K = 8$, while SEQ-HSMM and bigram have lower final-test NLL (Table 14).

G.2 Learned Libraries and Structural Analysis

Figure 7 shows the five explicit partial-order skills from training-only fit s102. This illustration uses all five skills, whereas the displayed CompILE-CPA+SM fit assigns 95.8% of occurrences to two codes (Figure 8). Across 12 CompILE fits, the two most-used codes account for at least 92% of occurrences in 11. These comparisons describe learned assignments, not agreement with reference skill labels.

Empirical ordering. For 100 training-reference pairs with at least 90% directional consistency and 60 co-occurring traces, generated-sequence first-occurrence agreement is 87–97 pairs for RPOS (mean 91), 86 for bigram, 71–88 for SEQ-HSMM, and 68–88 for CompILE-CPA+SM, under common length conditioning. These are empirical regularities, not gold dependencies; no reliable cross-model advantage is established.

Sampling diagnostics. For the four $K = 5$ runs over sweeps 20,005–30,000, split-$\widehat{R}$ compares the two halves of each run. Initial-skill entropy has $\widehat{R} = 1.000$–1.001, but training likelihood has $\widehat{R} = 1.020$–1.939 and the comparable-pair fraction reaches 1.522. Structural summaries also differ across seeds; the results remain finite-run estimates rather than calibrated structural uncertainty. This diagnostic window is separate from the predictive sampling window.

G.3 Source-Library Reuse and Adaptation

A post-hoc follow-up reuses the 20 previously evaluated test traces. Each of five tasks supplies trial 0 as support and trials 1–3 as queries. All four $K = 5$, $d = 4$ source chains are included. Shared fixes the source library (U, ρ) and adapts only (π, P); Support-only learns all four quantities from the same support trace without source-trained parameters. Frozen leaves the source model unchanged. Query traces are used only for scoring.

After 20 composition updates, Shared improves query NLL on every task over both support-only budgets, with mean gains of 0.6203 and 0.4987 nats per occurrence (Table 15). However, Frozen predicts better than Shared on every task; 200 updates do not reverse this ordering. Thus, source training helps under the tested budgets, but composition adaptation adds no predictive benefit. All three adapted sequence baselines have lower query NLL than RPOS Shared.

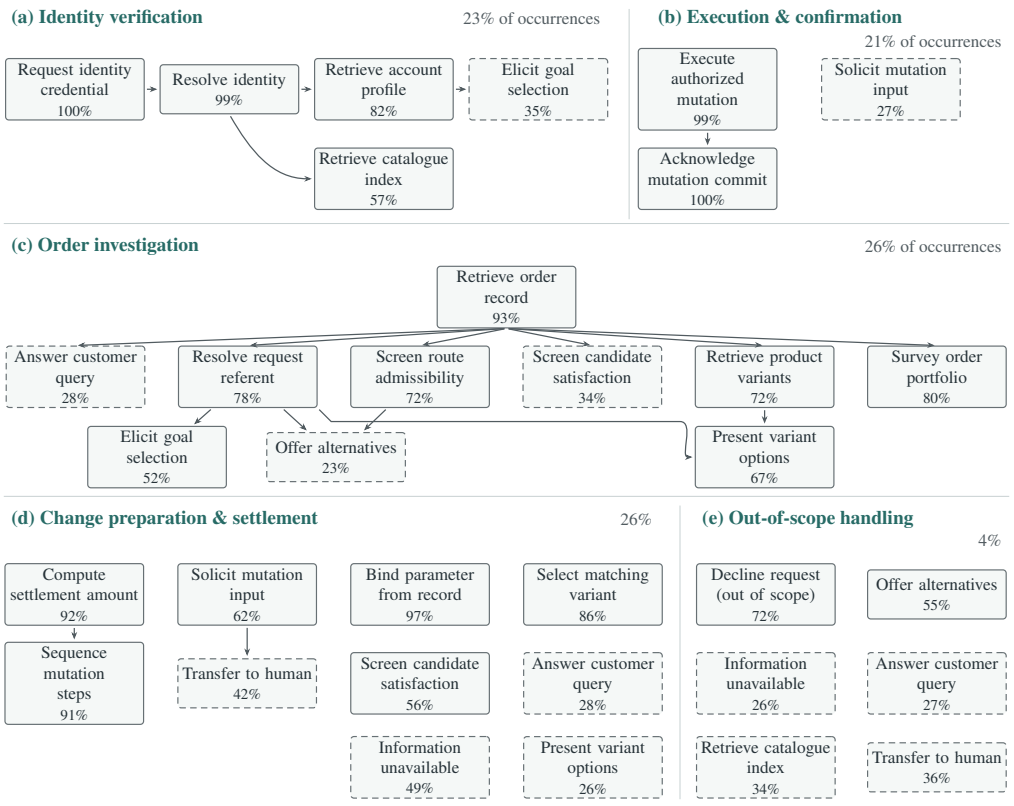


Figure 7: **A learned Retail skill library.** Training-only s102 fit ($K = 5$), selected and named post hoc. Nodes show action-specific responsibility shares ($\geq 20\%$; dashed below 50%); headings give total skill usage. Arrows are transitively reduced after thresholding within-run relation frequencies at 0.70. Missing arrows do not establish independence.

Table 15: **Reuse and adaptation on five Retail tasks.** Length-conditioned query NLL per CPA occurrence (↓), averaged over fits within tasks and then equally across tasks. FROZEN/SHARED use four RPOS source chains; SUPPORT-ONLY and each learned baseline use three fits. Target-fitting budgets are not compute-matched.

Method / setting	Target-task fitting	Query NLL
RPOS FROZEN	None	1.5920
RPOS SHARED	20 composition updates	1.7798
RPOS SHARED	200 composition updates	1.7749
RPOS SUPPORT-ONLY	2,000 sweeps	2.4001
RPOS SUPPORT-ONLY	10,000 sweeps	2.2785
SEQ-HSMM SHARED	One expected-count update	1.3698
CompILE-CPA+SM SHARED	Inferred-path composition	1.7283
Bigram SHARED	0.9 source + 0.1 support	**1.2933**

H COMPUTATIONAL EFFICIENCY

We reduce the cost of semi-Markov path calculations and recurrent block-likelihood evaluation without approximating the likelihood, subsampling traces, lowering numerical precision, or restricting the stated duration support. Implementation optimizations are checked against a frozen reference. Changes to the frequency of structural updates are reported separately, since they alter the sampling schedule rather than merely its implementation.

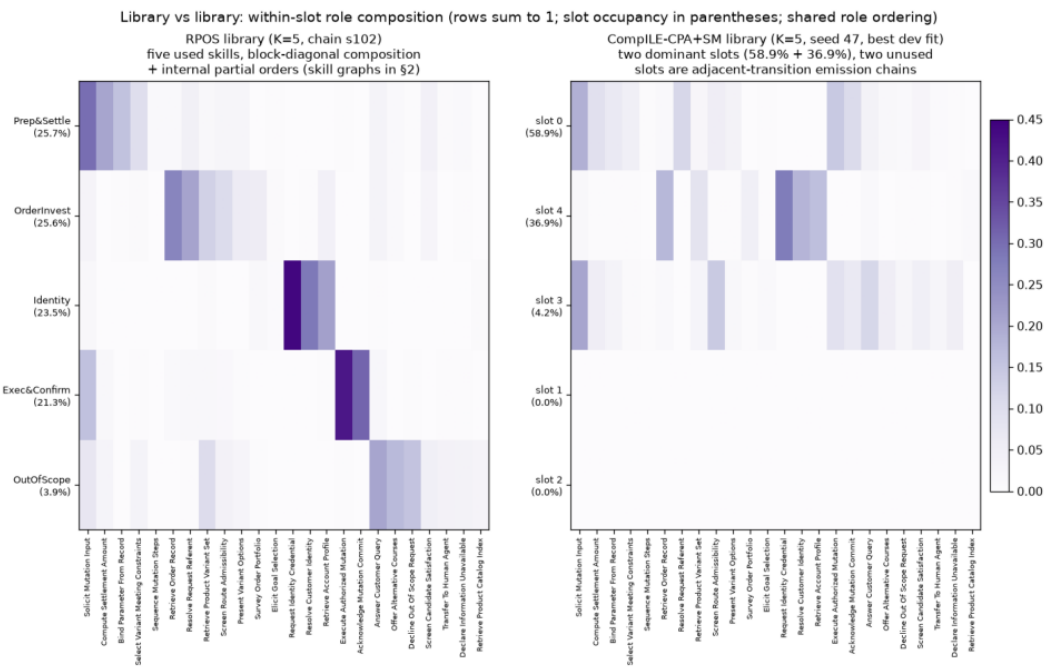


Figure 8: **Action composition within skills.** RPOS (seed 102) and CompILE-CPA+SM (seed 47), both at $K = 5$. Columns share the same CPA order; nonempty rows sum to one, and row labels give occurrence shares. Fits were selected for illustration; CompILE's best development fit is shown. Matrices summarize assignments, not RPOS emission parameters.

H.1 Algorithmic Reductions in the Forward Pass

For N traces of length J, maximum segment width D, and K skills, direct evaluation of the forward recursion with precomputed block scores costs $O(NJDK^2)$. The sum over predecessor skills depends on the preceding boundary but not on the next block's ending position. We therefore compute

$$I_n(a, k) = \text{logsumexp}_{h=1}^{K} \left[\log \alpha_n(a, h) + \log P_{hk}\right], \qquad 1 \le a < J_n, \tag{41}$$

once for each interior boundary a and successor skill k, and reuse it for every legal block starting at $a + 1$. Blocks starting at the beginning of the trace instead use $\log \pi_k$.

This factorization reduces the forward-pass cost to $O(NJK^2 + NJDK)$ without removing paths or changing their weights. For unequal trace lengths, the cost is $O\left(\sum_{n=1}^{N} J_n(K^2 + DK)\right)$. These bounds exclude block-likelihood construction and the remaining work required by structural proposals.

Additional optimizations batch equal-length traces, inline the frequently evaluated log-sum-exp reduction, and reuse emission tables when their inputs are unchanged. Batched results are returned to the original trace order before backward sampling, preserving the order in which random numbers are consumed.

H.2 Compiled Likelihood Kernel

Recurrent block replay is implemented as a C99 function called through `ctypes`. For B blocks of width w with m supported actions, four action-sized scratch buffers replace the reference implementation's (B, w, m) feature tensors and (B, m, m) predecessor temporary. The kernel is compiled with `-O3`, `-fno-fast-math`, and `-ffp-contract=off`. Source and shared-library hashes are checked against a build receipt to detect stale binaries. Each candidate is evaluated under its own proposed parameters; the Metropolis–Hastings (MH) acceptance calculation is unchanged.

H.3 Update Scheduling and Cache Scope

Path-marginal structural proposals integrate over segmentations and skill assignments rather than condition on a single stored path. After every such attempt, accepted or rejected, forward-filtering backward-sampling (FFBS) immediately refreshes the explicit paths before any subsequent path-conditioned update. The update cadence controls how often this complete transition is applied; it does not omit the conditional refresh.

Emission-table reconstruction is skill-local. When only a row of U_k changes, the affected block scores belong to skill k; scores for the other $K-1$ skills remain reusable if their inputs are unchanged. A runtime assertion checks this cache-invalidation rule. The forward recursion must still account for the complete library, since its normalizer couples all skills through alternative labelled paths. No separate runtime gain is attributed to skill-local reuse.

Cache keys must identify the scoring context of the cached quantity. In the historical oracle-conditional setting, a $(\text{skill}, \text{canonical order})$ key is sufficient because the remaining inputs are fixed. It is not a general-purpose key for the corpus normalizer, which also depends on the other skill structures, execution parameters, and composition probabilities.

H.4 Structural-Update Cost and Memory

Update costs. At the controlled synthetic operating point, separately timed candidate-table reconstruction takes $33.84\,\text{s}$, compared with $32.77\,\text{s}$ for a Full-Cond structural sweep. These comparable durations indicate that candidate-score construction is a major computational cost. Because the measurements were obtained separately, their ratio is not a within-sweep percentage attribution.

With a structural-update cadence of one sweep in ten, the recorded amortised Full-Marg/Full-Cond runtime ratio is 1.044. Its denominator is the corresponding amortised Full-Cond runtime, not the time of an ordinary plain sweep. A separate historical path-marginal utility-update audit reports 8–15% additional cost when inserting a marginal proposal every 5–10 sweeps, and approximately $11\times$ the cost when using the marginal kernel as a complete replacement for the conditional kernel. These measurements use different configurations and comparison baselines; they are not combined into a single speedup.

Memory. Candidate scores are stored in dense $(J, J+1, K)$ arrays per trace. The implemented storage therefore requires $O(NJ^2K)$ memory for equal-length traces, despite the duration restriction. Storing only legal segment widths would reduce this requirement to $O(NJDK)$, but this banded layout is not implemented in the audited code and is not claimed as a measured improvement.

I Differentiable Fitting and Strict Projection

We evaluate a post-hoc gradient-based alternative to structural MCMC, retaining exact summation over skill boundaries and assignments.

I.1 Relaxation and Prediction

For distinct actions i, j in skill k, we replace the precedence indicator by

$$D_{ij}^{(k)} = \sigma\left(-\gamma\tau \log \sum_{r=1}^{d} \exp\left[-\frac{U_{ir}^{(k)} - U_{jr}^{(k)}}{\tau}\right]\right), \qquad D_{ii}^{(k)} = 0. \tag{42}$$

Here σ is logistic, $\tau > 0$ smooths the minimum coordinate margin, and $\gamma > 0$ controls sharpness. These strengths enter the execution model; they are not posterior edge probabilities. Suppressing k, let v denote output validity. The relaxed features are

$$F_j(v) = \prod_i [1 - D_{ij}(1 - v_i)], \qquad R_j(v) = \sum_i D_{ji}(1 - v_i), \qquad B_j(v) = \kappa \sum_i D_{ji} v_i, \tag{43}$$

where $\kappa = \sigma(\omega)$. We use log weights $\ell_j = \log F_j + \beta \log(1 + R_j) - \lambda_{\text{rep}} v_j - \lambda_{\text{back}} B_j$ and probabilities $\widetilde{p}(j \mid v, k) = (1-\epsilon)\,\text{softmax}(\ell)_j + \epsilon/m_k$. Each block starts at $v = 0$; observing c sets $v'_c = 1$ and $v'_j = v_j(1 - \kappa D_{cj})$ for $j \neq c$.

Table 16: **Differentiable synthetic results.** Held-out NLL per occurrence, reported as means (sample SDs) over four starts. Exact is the number of projected libraries matching truth up to skill-label permutation. Projection receives no refitting; 0.0000 denotes an SD below 5×10^{-5}.

Corpus	Schedule	Soft NLL	Projected NLL	Exact /4
Confirmatory	Fixed	1.4600 (0.0015)	1.4584 (0.0001)	4
Confirmatory	Continuation	1.4603 (0.0006)	1.4585 (0.0001)	4
Full recurrent	Fixed	1.3266 (0.0104)	1.3276 (0.0155)	2
Full recurrent	Continuation	1.3176 (0.0000)	1.3142 (0.0000)	4

We maximize normalized training log likelihood plus parameter log priors, differentiating through the semi-Markov recursion. The loss is divided by the training occurrence count. We use full-batch Adam with learning rate 0.03 for 30 updates and 0.01 thereafter, clipping the gradient norm at 5. Evaluation uses $p_\nu(x^{(n)} \mid J_n, \widehat{\Theta}) = Z_{n,\nu}(\widehat{\Theta})/C_{J_n}(\delta_B)$, for $\nu \in \{\mathrm{soft}, \mathrm{hard}\}$, with NLL divided by total action occurrences. Strict projection replaces D by the coordinate-dominance orders $H_k = h(U_k)$, without refitting, and recomputes the path sum. This is not a threshold at $D_{ij} = 0.5$.

I.2 Synthetic Recovery and Prediction

The confirmatory and full-recurrent corpora each contain 100 training and 45 held-out traces from three skills over five actions ($d = 2$, $\rho = 0.5$, widths 3–12). The former fixes $(\beta, \kappa, \lambda_{\mathrm{rep}}, \lambda_{\mathrm{back}}) = (1.5, 0.85, 0.8, 0.25)$; the latter additionally optimizes $(\beta, \omega, \lambda_{\mathrm{rep}}, \lambda_{\mathrm{back}})$. Four matched starts compare fixed $(\tau, \gamma) = (0.1, 5)$ with continuation through $(0.5, 1)$, $(0.25, 2)$, and $(0.1, 5)$ over 20%, 20%, and 60% of the budget. Fits stop at 6,000 updates or 900 seconds; schedule progress is the larger fraction of these two budgets consumed. Checkpoints are selected by the final-target training objective, not held-out performance.

Continuation recovers the generating library after strict projection in all four starts on both corpora (Table 16). On the full-recurrent corpus, it improves exact recovery from $2/4$ to $4/4$ and lowers soft held-out NLL from 1.3266 to 1.3176.

Confirmatory continuation averages 12.8 minutes per fit and achieves soft NLL 1.4603, compared with 1.4591 for the historical four-chain MCMC predictive mixture. The historical runtime is 3.24 hours per chain, but different timing scopes and unverified hardware equivalence preclude a matched-speedup claim. The MCMC mixture uses 1,000 selected draws from each of four 150,000-sweep chains, with 50,000 warm-up sweeps and thinning by five before predictive subsampling. Historical NLLs include C_J; point-fit means average losses rather than probabilities.

I.3 Retail Fits and Strict Projection

We use the same 372 training and 64 internal-development traces as the Retail capacity scan, with 24 actions and four starts at each $K \in \{4, 5, 6, 8\}$. Fits optimize (U, π, P) at $d = 4$ with widths 2–12 and the same fixed execution parameters as the confirmatory study. The coordinate prior sums out ρ over 399 equally spaced points in $[0.0025, 0.9975]$, with masses proportional to the $\mathrm{Beta}(1, 1/6)$ density. Each fit uses 30,000 updates with $(\tau, \gamma) = (0.25, 2), (0.1, 5), (0.05, 10)$ over successive 20%, 20%, and 60% stages. Evaluation uses the terminal checkpoint; all 16 starts are retained.

Two local relations recur in every fitted library: requesting identity credentials before resolving identity, and executing an authorized mutation before acknowledging its commit. Each is examined in the skill most responsible for its source action, with both actions meeting the 20% display threshold. These are reproducible fitted patterns, not verified Retail structure recovery.

Soft prediction is competitive at $K = 4$ (Table 17). Strict projection without refitting increases NLL in all 16 fits; at $K = 5$, the mean rises from 1.8191 to 2.2698. Thus, the soft model's predictive performance does not automatically carry over to its projected partial-order library.

Table 17: **Retail prediction and strict projection.** Internal-development NLL per occurrence; means (sample SDs) across four fits per capacity. Soft and projected predictions use the same fitted parameters. MCMC averages probabilities within each chain before computing its NLL.

K	Soft	Strict projection	MCMC mixture
4	1.8053 (0.0157)	2.0454 (0.2288)	1.8210 (0.0237)
5	1.8191 (0.0225)	2.2698 (0.2426)	1.7949 (0.0209)
6	1.8309 (0.0296)	2.1688 (0.2346)	1.7900 (0.0369)
8	1.8382 (0.0529)	2.1720 (0.1328)	1.7366 (0.0217)

Table 18: **Primary execution results.** 120 paired tasks using frozen candidate A. Costs are estimates under archived prices, not verified invoices.

Metric	N: stepwise	G: executor
Strict successes	103/120	120/120
Business completions	119/120	120/120
Episodes with policy violations	6	0
Model requests	986	160
Input + output tokens	2,772,594	361,204
Estimated cost, no cache discount (CNY)	2.422430	0.392743
Estimated cost, recorded cache discount (CNY)	1.095095	0.261390

J Downstream Execution and Scheduling

J.1 Online Model Use and Task Quality

We compare a stepwise agent (N) with the partial-order execution system (G) on 120 paired synthetic Retail tasks from 23 known business families. Both use `qwen3.8-flash`, identical requests and tools, and separate copies of the same initial environment. G combines the learned `K5_s102` library with 14 prepared API templates, business-stage prerequisites, and policy checks. It uses the model for slot extraction, user updates, and fallback, rather than selecting every action through model reasoning. The library and controller are frozen before fresh task parameters are generated; expected outcomes remain private to the evaluator. This comparison evaluates the integrated system, not the learned relations alone.

Strict success requires correct outcomes and reporting, authentication, policy compliance, required evidence and user-event exposure, and no execution failure. All episodes and online calls are included; offline preparation is excluded.

G reduces online tokens by 86.97% and requests by 83.77% (Table 18). Its 160 calls comprise 120 initial extractions, 27 updates, and 13 fallback requests; 91 of 93 no-update tasks succeed with one call. Of N's 17 strict failures, 12 involve final-answer checks, six policy violations, and one the business outcome; categories overlap.

Ten thousand paired bootstrap resamples within business families give a 95% interval of $[0.1111, 0.1513]$ for the ratio of total tokens G/N. No pair has N succeed while G fails. Under independent parameter draws, the one-sided 95% upper bound on this harmful-discordance probability is 2.47%, satisfying the prespecified five-percentage-point non-inferiority criterion for the evaluated family mixture.

Supporting rechecks. On the same 120 tasks with `qwen3.8-max`, G achieves $120/120$ strict successes versus $113/120$ for N, with 87.76% fewer tokens. A Flash repeat gives $119/120$ versus $103/120$; provider errors leave some usage unknown, so no complete efficiency ratio is reported. These rechecks are not independent task samples and are not pooled with the primary result. Complete supporting and diagnostic results remain in the accompanying experiment report.

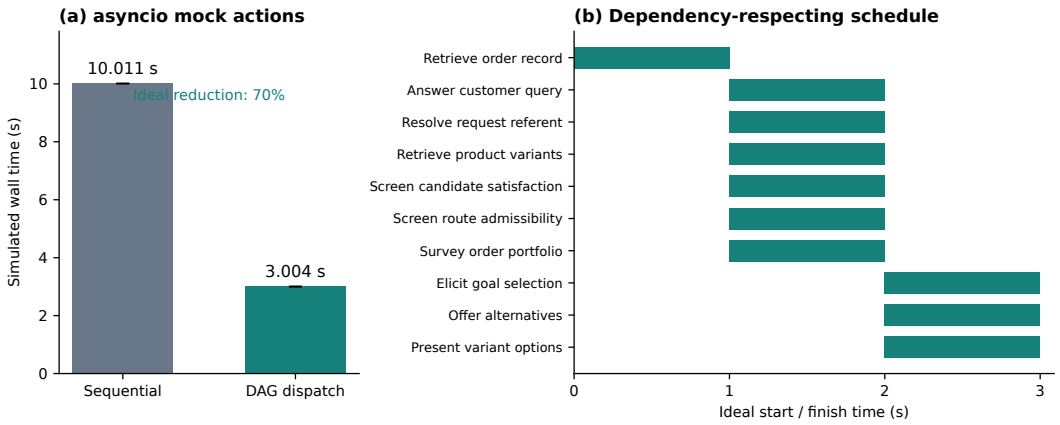


Figure 9: **Parallel scheduling with one-second mock jobs.** (a) Mean wall time over three paired repetitions; whiskers show observed ranges. (b) Ideal schedule respecting all displayed predecessors without a worker cap.

J.2 Reproducibility

The accompanying package exports 1,341 N/G records across EXP-0016–EXP-0027, including three zero-request startup records, with the actual code snapshot for every run. The offline audit reconciles 6,444 request receipts to the ledger; all 1,146 primary receipts additionally match the stored provider responses. Stored actions, tool outputs, and assessment details from all 844 N/G episodes in EXP-0024–EXP-0027 replay against their corresponding frozen environments. Reproduction verifies archived observations; it does not generate fresh model responses. Source hashes, per-task data, failure checks, family tables, and figure-generation code accompany the report.

J.3 Illustrative Parallel Scheduling

The displayed Order Investigation graph from fit s102 contains ten nodes and eleven edges, with a three-step critical path. For ten required one-second mock jobs, dependency-respecting parallel dispatch reduces mean completion time from $10.011\,\mathrm{s}$ to $3.004\,\mathrm{s}$ over three paired repetitions: approximately 70% less time ($3.33\times$ speedup; Figure 9). Both schedules execute all ten jobs and respect the displayed dependencies. Jobs use `asyncio.sleep(1.0)` with sufficient concurrency and no additional execution constraints; this measures mock-job latency, not live API latency or a reduction in total work.