Structured Flows for Causal Modelling and Statistical Inference

Anonymous authors
Paper under double-blind review

Abstract

Probabilistic Graphical Models (PGMs) specify the conditional-independence structure of a system, yet continuous normalizing flows (CNFs) are typically structure-blind. We present **StructFlow**, a framework that compiles an arbitrary directed PGM into a task-specific block-triangular CNF. From a single graphical specification, StructFlow produces either a *causal-alignment* model—preserving the generative factorization for interventional and counterfactual queries—or an *inference-alignment* model—reordered and moralized to capture the denser posterior dependencies induced by "explaining away". We formalize when each compilation admits tractable queries, show that block-triangular velocity fields preserve the directed acyclic graph (DAG) factorization, and support both static and dynamic conditioning paradigms. Rather than introducing a new flow primitive, our contribution is to *unify* the causal and inference directions for arbitrary DAGs behind a single automated compiler, and to *characterize* when structure helps: we show that the compiler's moralization step is necessary to represent collider posteriors, that sparse structure acts as a beneficial inductive bias in the low-data regime, and that the resulting inference engine is amortized—answering posterior queries roughly an order of magnitude faster than per-query markov chain monte carlo (MCMC). We demonstrate these properties on synthetic linear-Gaussian and nonlinear PGMs (up to eight nodes) and on MNIST. The implementation is released as open-source software.

1 Introduction

Probabilistic Graphical Models (PGMs) provide a language for describing the conditional (in)dependence structure of complex systems. However, there exists a fundamental tension between **generative (causal) factorization** and **discriminative (inference) factorization**. A single model can rarely perform both mechanistic simulation and complex posterior inference efficiently. For instance, a causal model ($X \to Y$) excels at interventional queries $p(Y|do(X))$ but requires expensive inversion or Markov chain Monte Carlo (MCMC) for inference $p(X|Y)$. Conversely, an inverse model ($Y \to X$) enables instant amortized inference but breaks the causal link, rendering it blind to the mechanisms of the world.

The root of this tension lies in the **Principle of Independent Mechanisms (PIM)**: the mechanism generating an effect, $P(\text{Effect}|\text{Cause})$, is independent of the distribution of the cause, $P(\text{Cause})$ (Peters et al., 2017). A flow optimized for the causal direction ($X \to Y$) learns a modular, reusable mechanism, but inverting it for inference ($P(\text{Cause}|\text{Effect})$) entangles these components via Bayes' rule, so forcing one set of parameters to capture both directions compromises both.

Beyond this modularity tension, causal models face a **structural mismatch** when used for inference. Causal directed acyclic graphs (DAGs) are often sparse, containing "colliders" ($X_1 \to Y \leftarrow X_2$) whose parents are marginally independent; but observing the child Y induces a dependency between X_1 and X_2 (the "explaining away" phenomenon (Koller & Friedman, 2009)), so the posterior is strictly denser than the prior. A flow built with the sparsity of the causal DAG lacks the computational paths (edges) required to represent these posterior correlations.

We propose **StructFlow** (**Struct**ural **Flow** Alignment), a framework for Continuous Normalising Flows that provides a unified *interface* and *methodology* for causal reasoning and statistical inference. By formalizing the mapping between PGM structures and flow architectures, we allow a user to define the system's dependencies once and "compile" two distinct model artifacts depending on their needs: a **World Model** (Causal Alignment) for mechanistic simulation, and an **Inference Engine** (Inference Alignment) that employs moralization to capture posterior correlations. Importantly, we train *separate* models for each task.

Our contributions are three-fold:

1. A **Unified Methodology** that automates the compilation of arbitrary PGMs into Structured continuous normalising flow (CNF) architectures, supporting both Causal (generative) and Inference (discriminative) tasks through a common interface (Table 1).
2. The **Implementation** and **Demonstration** of both Static and Dynamic conditioning mechanisms for arbitrary DAGs, generalizing previous work on block-triangular flows from a single observation-to-target conditioning split to arbitrary graphical structure.
3. An **Empirical Characterization** that isolates *why* structure matters: an ablation showing that the compiler's moralization step is necessary to represent collider posteriors (Section 4.4), and a demonstration that the resulting inference engine amortizes—answering posterior queries roughly $11\times$ faster than per-query MCMC beyond a small crossover (Section 4.6).

2 Related Work

The current work provides a unified *methodology* connecting two distinct lineages of flow-based modeling: Causal Autoregressive Flows for structural modeling, and Conditional Flow Matching for Bayesian inference.

Causal Representation Learning Recent progress in causal representation learning (Schölkopf et al., 2021) has emphasized the importance of learning latent causal structures from observational data. Our framework complements this by providing structured computational mechanisms for both causal reasoning and inference once the causal structure is known.

Neural ODEs and Continuous-Time Models The use of Neural Ordinary Differential Equations (ODEs) (Chen et al., 2018) has opened new avenues for modeling continuous dynamics and flexible function approximation. Our work leverages flow-based continuous models in a similar spirit, but with explicit structural constraints derived from graphical models.

Causal Generative Modeling The intersection of autoregressive flows and structural causal models (SCMs) was formalized by Khemakhem et al. (2021) with Causal Autoregressive Flows (CAREFL). They demonstrated that autoregressive architectures are structurally equivalent to SCMs, enabling interventional and counterfactual queries by fixing the base distribution (exogenous noise). Our **Causal Alignment** mode adopts this exact structural perspective but implements the transformation using Continuous Flow Matching instead of affine couplings. By replacing the CAREFL backbone with a continuous-time vector field, we retain the causal interpretability while gaining the flexibility of ODE-based generation. Similar structural constraints were explored by Wehenkel & Louppe (2021) and Pawlowski et al. (2020).

Flow Matching and Amortized Inference For statistical inference, Wildberger et al. (2023) demonstrated that Flow Matching can efficiently learn posteriors $p(\theta|x)$ for scientific simulators, though typically in a black-box manner. Most relevant to our work is the recent "Conditional Flow Matching for Bayesian Posterior Inference" by Zhai et al. (2025). They introduced the key innovation of **dynamic conditioning**, where the velocity field is conditioned on time-dependent trajectories of observed variables rather than static inputs. This allows for the construction of a triangular transport map that respects the dependency structure of the posterior. We extend the work of Zhai et al. (2025) by generalizing their method to arbitrary PGMs. Specifically, we show that by compiling a PGM into a valid sampling order, we can perform amortized inference by factorising the joint distribution into a sequence of tractable conditional flows.

Table 1: **Capability coverage.** No prior method spans both the causal and inference directions for arbitrary DAGs; StructFlow unifies them (and additionally provides an automated compiler and both static/dynamic conditioning, discussed in the text). ✓ = supported, × = not supported. "Explaining-away" denotes capturing the denser posterior dependencies (moralization) induced at colliders.

	Arbitrary DAG	Causal queries	Amortized inference	Explaining-away	Continuous-time
CAREFL (Khemakhem et al., 2021)	✓	✓	×	×	×
Causal NF (Javaloy et al., 2023)	✓	✓	×	×	×
StrNN (Chen et al., 2023)	✓	✓	×	×	×
Weilbach et al. (Weilbach et al., 2020)	✓	×	✓	✓	✓
Zhai et al. (Zhai et al., 2025)	×	×	✓	×	✓
StructFlow (ours)	✓	✓	✓	✓	✓

Two further structured approaches are closely related. Weilbach et al. (2020) construct structured CNFs for *amortized inference*, exploiting a minimally faithful inversion of the graph, but target the inference direction only (no causal/counterfactual queries); Javaloy et al. (2023) develop causal normalizing flows with identifiability guarantees but focus on the generative direction (no moralization-based inference); and Chen et al. (2023) use masked structured networks for density and causal-effect estimation but provide no amortized posterior sampling that accounts for explaining-away.

As Table 1 summarizes, each prior approach covers a *subset* of the capabilities we target. We do not claim a fundamentally new flow primitive; rather, our contribution is to *unify* the causal and inference directions for *arbitrary* DAGs behind a single automated compiler — including the moralization needed to handle explaining-away at colliders, and both static and dynamic conditioning paradigms.

Structured Generative Models Baptista et al. (2024) study the representation and learning of monotone triangular transport maps—approximations of the Knothe–Rosenblatt rearrangement whose sparsity pattern reflects the conditional-independence structure of the target—closely related in spirit to our graph-structured flows. Similarly, Bayesian Flow Networks (Graves et al., 2023) introduce structured generation with continuous-time updates.

3 Structured Flow Matching

3.1 Flow Matching Primitives

Continuous Normalizing Flows (CNFs) model a probability distribution by transforming a simple base density p_0 (e.g., standard Gaussian $\mathcal{N}(0, I)$) to a complex target density p_1 via a time-dependent vector field $v_t : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$. The flow $\phi_t : \mathbb{R}^d \to \mathbb{R}^d$ is defined by the ordinary differential equation:

$$\frac{d\phi_t(y)}{dt} = v_t(\phi_t(y)), \quad \phi_0(y) = y \tag{1}$$

A sample $y_0 \sim p_0$ is transported to $y_1 = \phi_1(y_0) \sim p_1$ by integrating this ODE from $t = 0$ to $t = 1$.

Flow Matching (Lipman et al., 2022) is a simulation-free objective for training CNFs. Instead of maximizing likelihood via expensive ODE integration, it regresses the parameterized velocity field v_t^ϕ onto a target conditional vector field u_t that generates a desired probability path p_t:

$$\mathcal{L}(\phi) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, y_1\sim p_1,\, y_0\sim p_0} \left[\left\| v_t^\phi(y_t) - u_t(y_t \mid y_0, y_1) \right\|^2 \right] \tag{2}$$

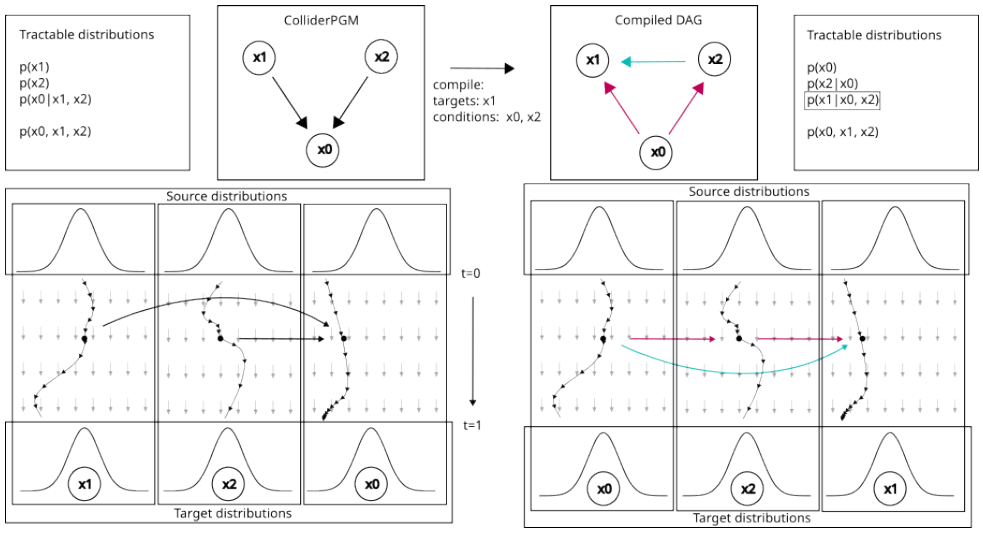


Figure 1: **The StructFlow Pipeline.** Given a causal graph (top left) and an inference query $p(x_1, x_2|x_0)$, StructFlow transforms the graph into a valid flow architecture. 1. The graph is **moralized** (adding the blue edge between x_1 and x_2) to account for the "explaining away" effect. 2. The flow is ordered such that the condition x_0 is integrated first, providing a dynamic context for the target variables.

where y_t is a sample from the probability path, defined by the linear interpolant $y_t = (1-t)y_0 + ty_1$, and the target velocity field is $u_t(y_t \mid y_0, y_1) = y_1 - y_0$.

3.2 Block-Triangular Architectures

We define a **Block-Triangular Flow** as a CNF whose velocity field respects a DAG structure $\mathcal{G}$. Let the state $y \in \mathbb{R}^d$ be partitioned into K disjoint blocks $y = [y^{(1)}, \ldots, y^{(K)}]$, ordered according to a topological sort of $\mathcal{G}$. The velocity field decomposes as:

$$\frac{dy_t^{(k)}}{dt} = v_\phi^{(k)}\big(y_t^{(k)}, \mathrm{Pa}_{\mathcal{G}}(y^{(k)})_t, t\big), \quad k = 1, \ldots, K \tag{3}$$

where $\mathrm{Pa}_{\mathcal{G}}(y^{(k)}) \subseteq \{y^{(1)}, \ldots, y^{(k-1)}\}$ denotes the parents of block k in the DAG $\mathcal{G}$.

Example (Chain). For the causal chain $X \to Y \to Z$ the blocks are $v_\phi^{(X)}(X_t, t)$, $v_\phi^{(Y)}(Y_t, X_t, t)$, and $v_\phi^{(Z)}(Z_t, Y_t, t)$: Z's velocity depends on Y but not directly on X, respecting the conditional independence $Z \perp X \mid Y$.

This constraint ensures the Jacobian $J = \frac{\partial \phi_1(y_0)}{\partial y_0}$ is block lower-triangular, with the log-determinant decomposing as $\log|J| = \sum_{k=1}^{K} \log|J^{(kk)}|$. This structure enables tractable marginal densities and efficient conditional sampling by fixing parent trajectories. However, this tractability is *asymmetric* and depends strictly on the variable ordering.

The block-triangular structure is realized purely through the connectivity of the velocity network, and it is preserved by the flow dynamics:

Proposition 3.1 (Structural Preservation)**.** *Suppose the velocity field is parameterized through a denoiser $\hat{\eta}_1$ whose k-th block depends only on the topological prefix, $\hat{\eta}_1^{(k)}(y, t) = \mathrm{NN}^{(k)}\big(y^{(1)}, \ldots, y^{(k)}, t\big)$. Then, for the linear interpolant $y_t = (1-t)y_0 + ty_1$, the induced velocity field v_t and the flow map ϕ_t are block-triangular with the same dependency structure. Consequently, the learned distribution factorizes as $p(y) = \prod_k p\big(y^{(k)} \mid y^{(1)}, \ldots, y^{(k-1)}\big)$, matching the Markov factorization of the DAG $\mathcal{G}$.*

Proof sketch. For the linear interpolant, $v_t(y) = c_1(t)\, y + c_2(t)\, \hat{\eta}_1(y, t)$ is an affine combination of y and the denoiser, so block k of v_t depends only on $y^{(1:k)}$. Triangularity of the velocity field implies triangularity of the flow map, hence the block-triangular Jacobian and the stated factorization. See Appendix D for the full proof. □

This is the architectural guarantee underlying StructFlow: constraining *which inputs each block of the network sees* is sufficient to make the entire transport respect the graph. The same structure, however, makes density computation *order-dependent*:

Theorem 3.2 (Asymmetry of Marginalization)**.** *In a Block-Triangular Flow with ordering $y^{(1)} \to \cdots \to y^{(K)}$, call the marginal $\log p(y_S)$ of a subset S* directly tractable *if it can be evaluated by the change-of-variables formula restricted to S (a single pass, with no integration over the remaining variables). Then $\log p(y_S)$ is directly tractable if and only if S is* ancestrally closed*, i.e., S contains all parents (and hence all ancestors) of each of its members. A topological* prefix *$S = \{1, \ldots, k\}$ is the canonical such set, and is the case used by our compiler. For any S that is not ancestrally closed, exact evaluation requires marginalizing out the missing ancestors, which is intractable in general.*

Proof Sketch. If S is ancestrally closed then $v^{(j)}$ for each $j \in S$ depends only on variables in S, so the evolution of y_S is self-contained, the Jacobian restricted to S remains triangular, and the remaining variables marginalize to unity without affecting $\log p(y_S)$. Conversely, if some $j \in S$ has a parent $i \notin S$, the marginal requires integrating that ancestor out:

$$p(y_S) = \int p(y_S \mid y_{\bar{S}})\, p(y_{\bar{S}})\, dy_{\bar{S}}, \tag{4}$$

and the conditional is a nonlinear function of $y_{\bar{S}}$ induced by the flow, so the integral has no closed form; a Monte Carlo estimate would re-sample the missing ancestors and thus re-incur the cost of the full joint model. See Appendix D for details. □

Implication. This explains why a single fixed architecture cannot efficiently serve both tasks: a causal model ordered $X \to Y$ computes $p(X)$ and $p(X, Y)$ efficiently but must marginalize over X for $p(Y)$, whereas efficient inference $p(X \mid Y)$ needs Y ahead of X—a task-specific re-ordering.

3.3 StructFlow: Structural Flow Alignment

A Probabilistic Graphical Model defines a generative factorization of a joint distribution $p(y) = \prod_k p(y^{(k)} \mid \mathrm{Pa}(y^{(k)}))$ via a DAG $\mathcal{G}$. We introduce **StructFlow** (Structural Flow Alignment), an algorithm that translates the abstract graph $\mathcal{G}$ into a concrete block-triangular flow architecture tailored to a specific probabilistic query. Depending on the desired capability, StructFlow produces one of two distinct model artifacts (each trained separately).

We make the following standard assumptions, which we state explicitly because they delimit the queries StructFlow can answer correctly.

Assumption 3.3 (Causal Sufficiency)**.** The DAG $\mathcal{G}$ contains no unobserved confounders: every common cause of two variables in $\mathcal{G}$ is itself a node of $\mathcal{G}$.

Assumption 3.4 (Markov Condition)**.** The data distribution factorizes according to $\mathcal{G}$; equivalently, each variable is independent of its non-descendants given its parents.

Assumption 3.5 (Faithfulness)**.** Every conditional independence that holds in the data distribution is entailed by d-separation in $\mathcal{G}$.

Assumptions 3.3–3.4 let the causal-alignment model identify interventional distributions (so that fixing parents in the flow corresponds to a do(·) operation). Faithfulness (Assumption 3.5) is used by the inference compiler to prune a candidate dependency whenever the corresponding variables are d-separated, making the compiled inference graph both *sufficient* (no posterior dependency is missed) and *minimal*. We assume $\mathcal{G}$ is known; structure discovery is orthogonal (Section 5). Counterfactual queries require an additional identifiability condition (Remark 3.7).

3.3.1 Causal Alignment

When the goal is mechanistic simulation or causal reasoning, StructFlow preserves the DAG structure directly. The flow blocks are ordered according to a topological sort of $\mathcal{G}$ (roots $\rightarrow$ leaves), and each block's velocity field receives inputs only from its DAG parents:

$$v_\phi^{(k)}(y_t^{(k)}, \mathrm{Pa}_{\mathcal{G}}(y^{(k)})_t, t) \tag{5}$$

Because this structure mirrors the causal mechanisms (respecting the Principle of Independent Mechanisms (Peters et al., 2017)), the resulting flow enables interventional queries $p(y \mid \mathrm{do}(x))$ by fixing parent values, and counterfactual queries via flow inversion (Section 3.5).

3.3.2 Inference Alignment

When the goal is amortized posterior inference $p(\Theta \mid Y)$ where Θ are latent variables and Y are observations, StructFlow must address two challenges:

Challenge 1: Ordering. By Theorem 3.2, the observations Y must precede the targets Θ in the flow ordering to enable tractable conditional density evaluation.

Challenge 2: Posterior Dependencies. The conditional independence structure of the posterior $p(\Theta \mid Y)$ can be *denser* than the prior: at a collider $X_1 \rightarrow Y \leftarrow X_2$ with $X_1 \perp X_2$ marginally, observing Y induces $X_1 \not\perp X_2 \mid Y$ ("explaining away"), and a flow respecting only the original DAG lacks the X_1–X_2 edge needed to capture this correlation.

The compilation pipeline for inference mode consists of three steps:

1. **Moralization:** Construct the moral graph $\mathcal{G}_m$ by adding undirected edges between all pairs of variables that share a common child in $\mathcal{G}$, then dropping edge directions. This ensures the graph can represent dependencies induced by conditioning.
2. **Re-ordering:** Compute a valid ordering where all observations Y precede all targets Θ. Within each group, we use reverse topological order from $\mathcal{G}$ to respect natural dependencies.
3. **Dependency Pruning:** For each target variable θ_k in the computed order, determine its minimal parent set using d-separation:

$$j \in \mathrm{Pa}_{\mathrm{flow}}(\theta_k) \iff \theta_k \not\perp_d z_j \mid \{z_1, \ldots, z_{k-1}\} \setminus \{z_j\} \tag{6}$$

   where $z_1, \ldots, z_{k-1}$ are all variables preceding θ_k. This ensures the inference flow is maximally sparse while remaining faithful to the posterior structure.

3.4 Conditioning Mechanisms

Given a block-triangular flow with ordering $[Y, \Theta]$ (observations first, targets second), we must specify how the target velocity field $v_\phi^{(\Theta)}$ receives information from the conditioning variables Y. Note that $v_\phi^{(\Theta)}$ denotes the *subset* of the full velocity field corresponding to the target variables Θ—i.e., the components of the vector field that govern the evolution of Θ. We implement two strategies:

Static Conditioning. In static mode, conditioning variables are first integrated to completion, then their *final values* $y_1 = \phi_1(y_0)$ are provided as fixed inputs to the target velocity field:

$$\frac{\partial \theta_t}{\partial t} = v_\phi^{(\Theta)}(\theta_t, y_1, t) \tag{7}$$

Sampling proceeds sequentially: first integrate Y from $t = 0$ to $t = 1$, then integrate Θ using the resulting y_1 as a static condition. For a flow with K blocks in the target set, this requires $K + 1$ sequential ODE solves.

Example. For the chain $Y \rightarrow \Theta_1 \rightarrow \Theta_2$, static sampling integrates Y to obtain y_1, then Θ_1 via $v^{(\Theta_1)}(\theta_t^{(1)}, y_1, t)$, then Θ_2 via $v^{(\Theta_2)}(\theta_t^{(2)}, \theta_1^{(1)}, t)$—one sequential ODE solve per block.

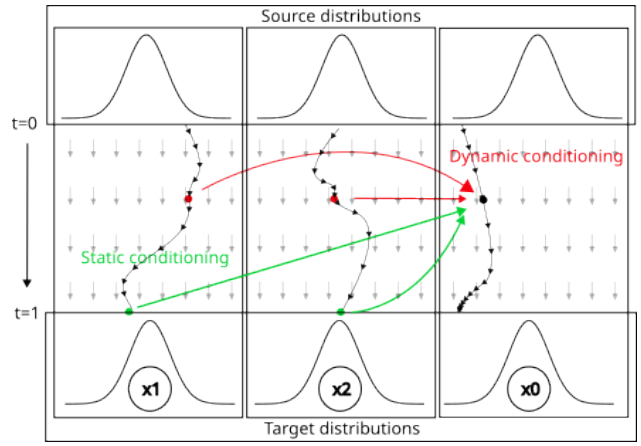


Figure 2: **Static vs. Dynamic Conditioning.** For inference in a collider $Y \leftarrow X_0 \rightarrow X_1$, static conditioning (green) integrates X_0 first to completion, then conditions subsequent blocks on $x_0^{(t=1)}$. Dynamic conditioning (red) integrates all variables jointly, with each block seeing the evolving trajectory $x_0^{(t)}$ at each time step.

Dynamic Conditioning. In dynamic mode, all variables are integrated *simultaneously* in a single ODE solve. The target velocity field is conditioned on the *time-evolving trajectory* of the conditioning variables:

$$\frac{d\theta_t}{dt} = v_\phi^{(\Theta)}(\theta_t, y_t, t) \tag{8}$$

where y_t co-evolves with θ_t. This requires the network to learn meaningful representations from partially-noised conditioning variables (at intermediate times $t < 1$), but enables a single parallel ODE integration for all variables.

3.5 Counterfactual Inference

Block-triangular flows enable counterfactual reasoning via the Abduction-Action-Prediction framework (Pearl, 2009) (illustrated schematically in Appendix Figure 6). The key insight is that the flow's bijectivity allows us to recover the unique exogenous noise u that generated an observation, then re-simulate under a hypothetical intervention.

Definition 3.6 (Counterfactual Query)**.** Given an observation (x, y) from a causal model $X \rightarrow Y$, the counterfactual query asks: "What would Y have been if X had been x' instead of x?" Formally, we seek $p(Y_{x'} \mid X = x, Y = y)$.

The procedure consists of three steps:

1. **Abduction:** Recover the exogenous noise by inverting the flow. Given observation y_{obs}, compute $u = \phi_1^{-1}(y_{\text{obs}})$ by integrating the ODE backward from $t = 1$ to $t = 0$:

$$\frac{dy_t}{dt} = -v_t(y_t), \quad y_1 = y_{\text{obs}} \implies u = y_0 \tag{9}$$

2. **Action:** Specify the intervention $\text{do}(X = x')$. In the flow, this means replacing the natural evolution of X with a fixed trajectory. For static flows, we simply set $x_1 = x'$. For dynamic flows, we must construct the full trajectory x'_t by integrating backward from x'.

3. **Prediction:** Integrate forward using the *original* noise u for non-intervened variables, but with the intervened variable fixed to its counterfactual trajectory:

$$\frac{d\theta_t}{dt} = v_\phi^{(\Theta)}(\theta_t, x'_t, t), \quad \theta_0 = u_\Theta \tag{10}$$

Remark 3.7 (Counterfactual identifiability). The abduction step recovers the exogenous noise $u = \phi_1^{-1}(y_{\text{obs}})$ that is consistent with the observation *under the learned model*. By the uniqueness of the Knothe–Rosenblatt rearrangement (Rosenblatt, 1952; Knothe, 1957), this u is unique given the variable ordering, but it need not coincide with the *true* exogenous noise of the data-generating process. Recovering the true mechanism—and hence guaranteeing that y_{cf} matches the ground-truth counterfactual—requires an additional identifiability condition beyond Assumptions 3.3–3.5, such as an additive-noise (or otherwise restricted) structural form, or access to interventional data (Khemakhem et al., 2021; Javaloy et al., 2023). Absent such a condition, our counterfactuals should be read as model-based estimates; we evaluate their accuracy empirically in Section 4.

4 Experiments and Discussion

Our framework allows us to define a PGM once and compile it into task-specific flows. We evaluate our framework in two primary regimes: *Inference Alignment* for statistical queries and *Causal Alignment* for mechanistic queries. We compare four methods:

- **CAREFL**: The baseline approach from Khemakhem et al. (2021) which uses affine autoregressive flows.
- **Guided**: An unstructured flow trained on the joint distribution without structural constraints.
- **Static**: Our structure-aligned flow with static conditioning (condition on x_1).
- **Dynamic**: Our structure-aligned flow with dynamic conditioning (condition on x_t).

We seek to answer the following questions:

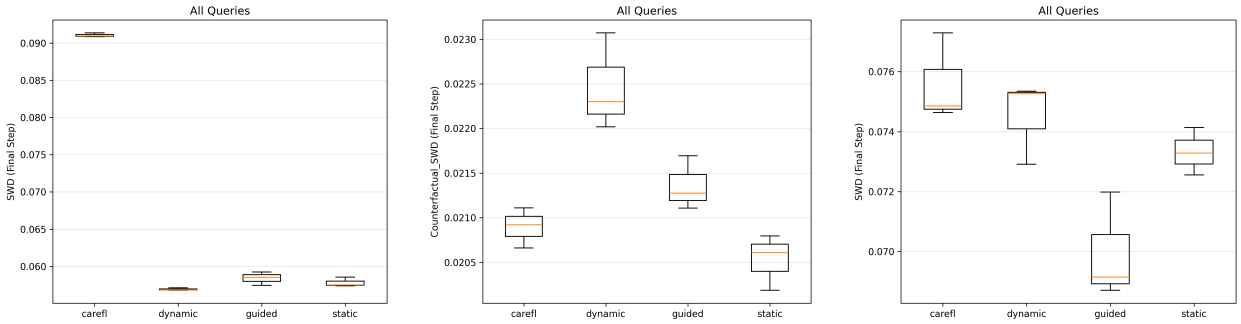
- **Q1 (Causal Hierarchy):** Can structure-aligned flows support all three levels of Pearl's causal hierarchy—observational, interventional, and counterfactual queries?
- **Q2 (Inference Engine):** Can flows compiled in *Inference Mode* provide accurate amortized inference?
- **Q3 (Algorithm Comparison):** How do static and dynamic conditioning compare to CAREFL and guided flows?

4.1 Experimental Setup

We train and evaluate on synthetic 4-node nonlinear PGMs with randomly generated structural equations, giving each model access to samples from the joint distribution (see Appendix E for details).

We report the Sliced Wasserstein Distance (SWD) as our primary metric, since the nonlinear PGMs do not admit tractable density evaluation for all queries of interest; it is normalized by $\sqrt{N\,D}$ to be comparable across queries of differing dimension and sample size (so absolute magnitudes are small—the meaningful signal is the *relative* ordering between methods). Where ground-truth densities are available (linear-Gaussian PGMs on causal queries), we additionally report the exact KL divergence (Appendix F), which corroborates the SWD ranking on a calibrated scale.

Reading the plots. Unless stated otherwise, box plots aggregate the indicated PGM and learner seeds (e.g. 3×3 at n=4), the box spanning the interquartile range over per-seed, per-query values. Given these seed counts we report the observed spread rather than significance tests; "comparable" means the seed distributions overlap substantially. The claims we make sharply—the moralization ablation (Section 4.4) and the sample-efficiency gap (Section 4.5)—are measured against *exact* ground-truth posteriors rather than via SWD rankings.



(a) Interventional $p(y \mid x)$ (b) Counterfactual (c) Inference $p(\text{pa} \mid \text{ch})$

Figure 3: **Normalized Sliced Wasserstein Distance on nonlinear 4-node PGMs**, aggregated over the supported conditional queries, 3 PGM seeds and 3 learner seeds. **(a)** interventional queries $p(y \mid x)$ and **(b)** counterfactual queries (Causal Alignment); **(c)** posterior inference queries $p(\text{parents} \mid \text{children})$ (Inference Alignment). The CNF variants (Static, Dynamic) match or outperform CAREFL on causal queries; all methods are comparable on these small, well-sampled inference problems.

4.2 Causal Alignment

For Causal Alignment, we set the structure of the flow to match the structure of the PGM. This alignment enables the flow to support interventional and counterfactual queries.

Interventional Queries (Level 2). When the flow structure matches the causal DAG, conditional sampling $p(y|x)$ is equivalent to the interventional distribution $p(y|do(x))$. This is because the structural alignment ensures that fixing x in the flow corresponds to surgically setting x in the underlying causal mechanism, breaking the influence of x's parents.

Counterfactual Queries (Level 3). Flow-based models enable counterfactual reasoning via the abduction–action–prediction procedure of Section 3.5: invert the flow to recover a latent noise consistent with the observation (x, y), modify the antecedent $x \to x'$, then re-simulate forward with the recovered noise. Standard conditional diffusion models cannot answer counterfactual queries, as they cannot uniquely identify the noise responsible for a specific observation. **Note:** We recover a noise representation consistent with the observation under our learned model, but this is not guaranteed to correspond to the *true* exogenous noise of the underlying system without additional identifiability assumptions (see Section 5).

Results. At the observational level (Level 1), all flow-based methods recover the joint distribution $p(y)$ comparably well, with no consistent separation between structured and unstructured variants (joint-distribution SWD in Appendix Figure 13); we therefore focus the main text on the interventional and counterfactual levels, where structure is load-bearing. Figures 3a and 3b show that the CNF variants (Static, Dynamic) match or outperform CAREFL on interventional and counterfactual queries on the nonlinear PGMs (per-graph breakdown in Appendix Figure 11). The gap is expected: CAREFL relies on **affine** autoregressive transformations $x' = s(x_{<i}) \cdot x + t(x_{<i})$ for identifiability, but a composition of these under a fixed ordering is itself affine and not a universal density approximator (Khemakhem et al., 2021, App. B–C), so with a base distribution fixed for identifiability it cannot reshape into an arbitrary non-Gaussian conditional regardless of the capacity of s, t. Our CNFs instead realize any smooth monotone noise-to-value map (and, with dimension augmentation, are universal approximators of homeomorphisms (Zhang et al., 2020)); enforcing causal structure through the *topology* of the vector field rather than the algebraic form of the coupling thus retains expressivity that affine flows lack.

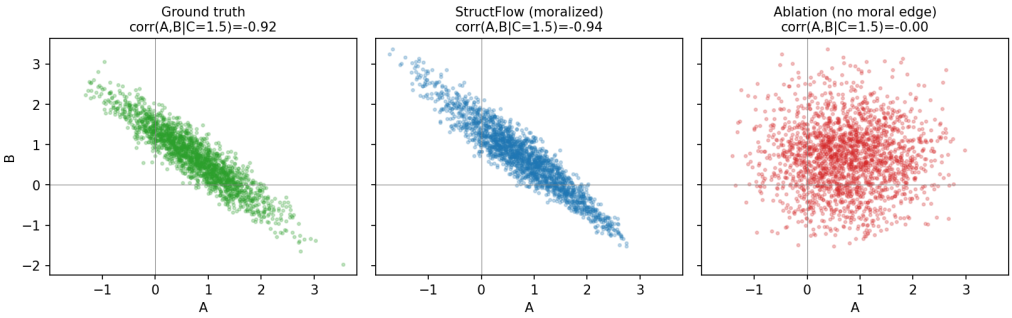


Figure 4: **Moralization is necessary for explaining-away.** Posterior samples of $p(A, B \mid C{=}1.5)$ for the collider $A \to C \leftarrow B$. **Left:** analytic ground truth (strong negative correlation). **Middle:** StructFlow with the moral edge recovers the correlation. **Right:** the ablation without the moral edge captures the marginals but collapses the correlation to ≈ 0.

4.3 Inference Alignment: Amortized Probabilistic Inference

For Inference Alignment, we use StructFlow to "invert" the DAG—edges point from children to parents, with additional edges via moralization. This enables efficient sampling from posterior distributions $p(\text{parents}|\text{children})$.

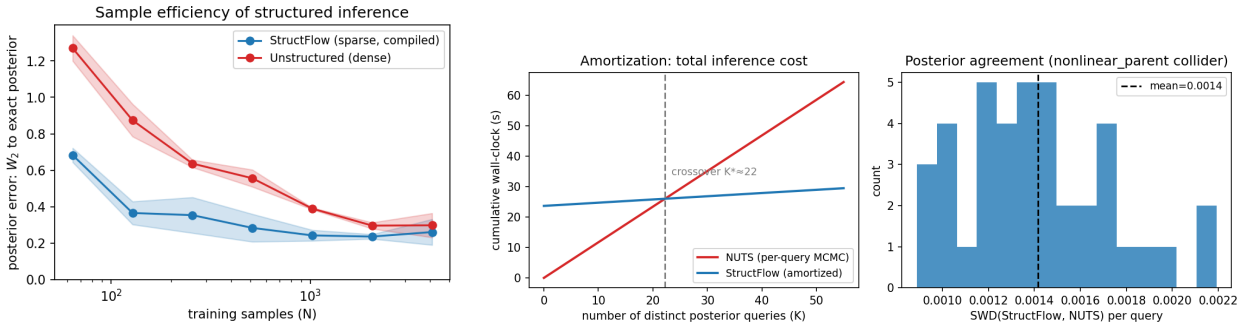
Results. Figure 3c shows that our structured flows can effectively learn to perform probabilistic inference, providing accurate amortized inference over the model's latent variables. On these small, well-sampled nonlinear PGMs all methods perform comparably—including the unstructured *Guided* baseline. This is expected rather than contradictory: the Guided flow places dependencies between *all* variables, so it implicitly contains the moral edges that the posterior requires. The ablation in Section 4.4 isolates this effect, showing that the relevant structural ingredient is exactly the moralization our compiler performs; and Section 4.5 shows that the benefit of *sparse* structure (over a dense, unstructured flow) emerges when training data are limited. The same qualitative picture holds on larger random 8-node nonlinear PGMs, confirming the framework is not limited to very small graphs (Appendix F, Figure 16); the binding constraint on scaling further is *evaluation*—generating MCMC ground truth for every conditional query (Section 5)—not training.

4.4 Ablation: Is Moralization Necessary?

The inference compiler adds moral edges between the parents of a collider to capture the posterior dependencies induced by "explaining away". We test whether these edges are truly necessary on a linear-Gaussian collider $A \to C \leftarrow B$ with $A, B \sim \mathcal{N}(0, 1)$ and $C = A + B + \sigma\epsilon$. Observing C induces a strong negative correlation between A and B, and the exact posterior $p(A, B \mid C)$ is available in closed form.

We compile the inference flow for the query $p(A, B \mid C)$ and compare the StructFlow architecture (which adds the moral edge $A{-}B$) against an ablated architecture with that edge removed—so the flow is forced to factorize as $p(A \mid C)\, p(B \mid C)$. Both are trained identically on the same joint samples.

Figure 4 shows the outcome. The ablated (non-moralized) flow recovers the correct posterior *marginals* but is structurally incapable of representing the posterior correlation, yielding $\text{corr}(A, B \mid C) \approx 0$. The moralized StructFlow recovers it closely (-0.94 vs. the analytic -0.92; the small overshoot is within the seed-to-seed variation of the learned flow). This isolates the contribution of the moralization step: the moral edge is not a heuristic convenience but a requirement for faithfully representing collider posteriors. We demonstrate the effect on this canonical collider because it admits a closed-form posterior, but the conclusion is *structural* rather than specific to this instance: without an edge between two colliding parents, a block-triangular flow has no computational path to couple them, so it cannot represent any induced posterior correlation regardless of the underlying mechanism.



(a) Structure as an inductive bias.

(b) Amortized inference vs. per-query MCMC.

Figure 5: **When and why structure pays off. (a)** Posterior error (2-Wasserstein to the exact posterior) versus training-set size for a linear-Gaussian chain: the compiler's sparse structure (blue) outperforms an unstructured dense flow (red) of the same family, with the largest advantage when data are scarce; the two converge as N grows (bands: ± 1 s.d. over seeds). **(b)** On a nonlinear collider. Left: cumulative wall-clock versus the number of distinct posterior queries; the amortized flow overtakes NUTS after $K^\star \approx 22$ queries. Right: per-query agreement between StructFlow and the NUTS reference (Sliced Wasserstein distance).

4.5 When Does Structure Help? A Sample-Efficiency Study

The ablation above shows that the *right* structure (with moral edges) is necessary for correctness. A complementary question is when sparsity is an *advantage*. On the small, well-sampled PGMs of Figure 3c, an unstructured flow can match a structured one because it has enough data to fit its extra (spurious) dependencies harmlessly. Structure should matter most when data are scarce, where its hard sparsity acts as an inductive bias that reduces variance.

We test this on a linear-Gaussian chain $x_0 \to x_1 \to \cdots \to x_5$, inferring $p(x_0, \ldots, x_4 \mid x_5)$ (the exact posterior is Gaussian). We compare two static flows of the same family that differ only in connectivity: the **sparse** structure produced by our compiler (5 edges) and a **dense**, fully-connected flow (15 edges). Both are trained identically across a range of training-set sizes N, and we measure the 2-Wasserstein distance between each flow's posterior and the exact posterior—an interpretable, zero-based metric.

Figure 5a shows the result. The structured (sparse) flow is more accurate at every N, and the gap is largest in the low-data regime (1.9× lower error at N=64), narrowing as N grows and the dense flow accumulates enough data to suppress its spurious dependencies. This is the expected behaviour of a correct inductive bias, and it explains the comparable performance of structured and unstructured flows in the well-sampled setting of Figure 3c: structure is most valuable precisely when data are limited. We present this single linear-Gaussian chain—which isolates the mechanism against an exact Gaussian posterior—as an illustration of the bias–variance trade-off that hard sparsity induces, not as a quantitative law; the magnitude of the advantage will depend on graph density and dimensionality.

4.6 Amortized Inference vs. MCMC

A practical advantage of Inference Alignment is *amortization*: the flow is trained once, after which any posterior query $p(\text{targets} \mid \text{conditions}{=}c)$ is answered with a single fast ODE solve, whereas a classical sampler such as the No-U-Turn Sampler (NUTS) must run a fresh Markov chain for every observed value c. On a nonlinear collider, Figure 5b (right) confirms that StructFlow's amortized posterior and a NUTS reference agree closely (mean Sliced Wasserstein distance 0.0014 across queries), while Figure 5b (left) shows the wall-clock trade-off: StructFlow pays a one-time training cost but is roughly 11× faster *per query*, so its cumulative cost overtakes per-query MCMC after a crossover of $K^\star \approx 22$ queries. The specific speedup and crossover are setup-dependent (Appendix E), but the qualitative conclusion is general: structured amortized inference is preferable whenever many posterior queries must be answered for the same model.

4.7 MNIST

We additionally evaluate on MNIST digit generation with the simple two-node structure `labels→images`, a sanity check at higher dimensionality (784 pixels vs. 4 scalar variables in the synthetic PGMs).

Results. All flow-based methods successfully learn the joint and marginal distributions (Figure 17, Appendix F), but dynamic conditioning exhibits notably higher SWD on the conditional $p(\text{images}|\text{labels})$ despite accurately representing the joint. This is a genuine distributional gap rather than an outlier artifact (full diagnostic in Appendix F): inspecting samples shows dynamic conditioning produces *perceptually crisp* digits that occasionally render the *wrong class* for the conditioning label—a broadly-distributed conditional error that raises SWD while leaving individual samples realistic. We trace this to dynamic conditioning's delayed conditioning signal (Section 5).

5 Future Work and Limitations

We discuss current limitations along with directions for further investigation.

Identifiability. The validity of counterfactuals relies on the learned flow approximating the true structural equations. The Knothe–Rosenblatt map (Rosenblatt, 1952; Knothe, 1957) is unique given an ordering but need not match the true causal mechanism without additional inductive biases or interventional data; understanding when the learned flow identifies the true structure remains open.

Known Structure. Our method requires the PGM structure (the existence and direction of edges), though not the structural equations. Where this structure is unknown, integrating structure learning with our framework is a natural extension.

Scaling to Large PGMs. We have evaluated only on small PGMs. How performance scales with the size and density of the graph—particularly for long chains, which may exhibit RNN-like vanishing/exploding gradients—remains to be studied.

Static vs Dynamic Conditioning. Dynamic conditioning exhibits higher mean and variance of SWD on conditional distributions—a genuine distributional gap rather than a metric artifact (Section 4). We conjecture this stems from its delayed conditioning signal: the velocity field receives $v(\theta_t, \text{pa}(\theta)_t)$, a partially-noised condition that only resolves as $t \to 1$, rather than the fully-formed $v(\theta_t, \text{pa}(\theta)_{t=1})$ of static conditioning. The network must then infer the eventual condition from noisy observations and execute rapid late-trajectory corrections—steep gradients characteristic of stiff ODEs. Characterizing this trade-off between conditioning information, parallelism, and numerical stability is an important direction.

6 Conclusion

We presented **StructFlow** (Structural Flow Alignment), a methodology for translating Probabilistic Graphical Models into task-specific Block-Triangular Flows: the same graphical structure compiles into a *Causal Alignment* model preserving the generative factorization for interventional and counterfactual queries, or an *Inference Alignment* model applying moralization to capture posterior correlations.

Our central—and deliberately measured—finding is that on small, well-sampled graphs structure is *competitive but not dominant*: an unstructured flow with enough data can match a structured one. The value of structure is nonetheless real and sharper than a headline metric suggests—a matter of *correctness* (moralization is necessary to represent collider posteriors at all), *sample efficiency* (hard sparsity helps when data are scarce), and *amortization* (one compiled engine answers many posterior queries far faster than per-query MCMC). Dynamic conditioning, while attractive for parallelism, incurs a genuine accuracy cost in high dimensions. Our small-graph evaluation is bounded chiefly by the cost of generating ground-truth posteriors, but the framework provides a foundation for scaling structured probabilistic reasoning to larger systems.

Reproducibility Statement

All theoretical claims are accompanied by proofs: Theorem 3.2 and Propositions 1–3 are proved in Appendix D, and the compilation procedure is given as explicit algorithms in Appendix C. The PGM-to-flow compiler, the static/dynamic/guided conditioning mechanisms, and all benchmarks are implemented in a single open-source library; an anonymized copy accompanies this submission and the code will be released publicly upon acceptance. Each analysis experiment is reproduced by a self-contained script with fixed seeds—the moralization ablation (Section 4.4), the sample-efficiency sweep (Section 4.5), and the amortization comparison (Section 4.6). The synthetic PGM benchmarks (Section 4) are generated from fixed seeds, and the exact architectures, optimizer settings, and parameter-budgeting procedure are documented in Appendix B. Ground-truth posteriors are computed in closed form where available (linear-Gaussian models) and via the No-U-Turn Sampler otherwise.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Ricardo Baptista, Youssef Marzouk, and Olivier Zahm. On the representation and learning of monotone triangular transport maps. *Foundations of Computational Mathematics*, 24:2063–2108, 2024.

Asic Q. Chen, Ruian Shi, Xiang Gao, Ricardo Baptista, and Rahul G. Krishnan. Structured neural networks for density estimation and causal inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pp. 2292–2300, 2013.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino Gomez. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023.

Adrián Javaloy, Pablo Sánchez-Martín, and Isabel Valera. Causal normalizing flows: from theory to practice. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Ilyes Khemakhem, Ricardo Monti, Robert Leech, and Aapo Hyvarinen. Causal autoregressive flows. In *International conference on artificial intelligence and statistics*, pp. 3520–3528. PMLR, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Herbert Knothe. Contributions to the theory of convex bodies. *Michigan Mathematical Journal*, 4(1):39–52, 1957.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Nick Pawlowski, Daniel C Castro, and Ben Glocker. Deep structural causal models for counterfactual inference. *Advances in Neural Information Processing Systems*, 33:857–869, 2020.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Murray Rosenblatt. Remarks on a multivariate transformation. *The Annals of Mathematical Statistics*, 23(3): 470–472, 1952.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

Antoine Wehenkel and Gilles Louppe. Graphical normalizing flows. In *International Conference on Artificial Intelligence and Statistics*, pp. 37–45. PMLR, 2021.

Christian Weilbach, Boyan Beronov, Frank Wood, and William Harvey. Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108 of *Proceedings of Machine Learning Research*, pp. 4441–4451. PMLR, 2020.

Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen R. Green, Jakob H. Macke, and Bernhard Schölkopf. Flow Matching for Scalable Simulation-Based Inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Percy S. Zhai, So Won Jeong, and Veronika Ročková. Conditional Flow Matching for Bayesian Posterior Inference. *arXiv preprint arXiv:2510.09534*, 2025.

Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural odes and invertible residual networks, 2020. URL `https://arxiv.org/abs/1907.12998`.

Appendix

A Appendix: StructFlow Details

We provide the implementation details for Inference Alignment of a DAG.

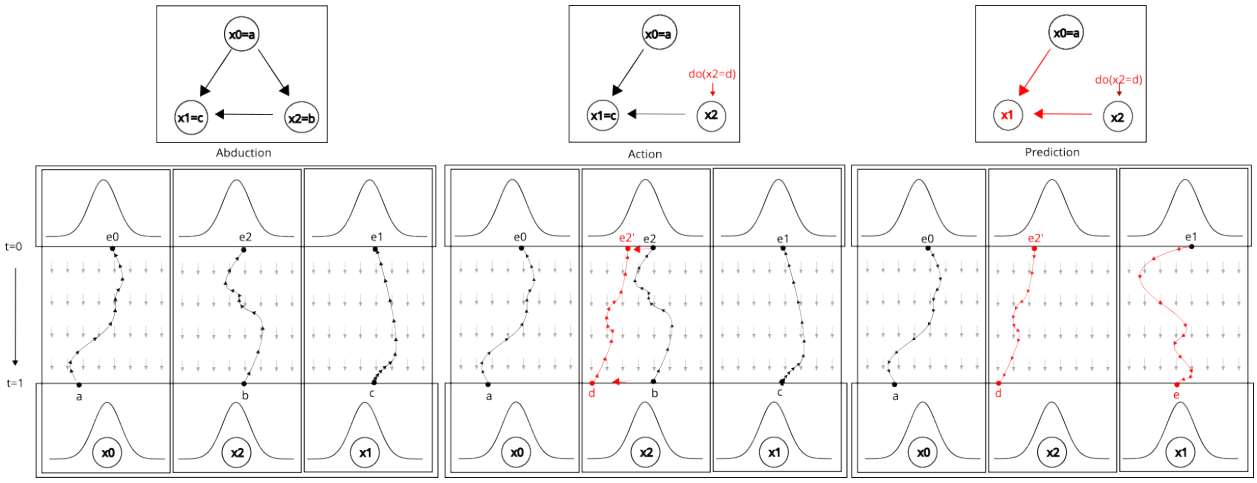


Figure 6: **Counterfactual Inference via Flow Inversion.** To answer "What would Y have been if X had been x'?", we perform three steps. **(1) Abduction:** Integrate backward from the observation (x, y) to recover the exogenous noise $u = (u_X, u_Y)$. **(2) Action:** Construct the intervention trajectory x'_t (red path) by integrating backward from x'. **(3) Prediction:** Integrate forward using the original noise u_Y but conditioned on the intervention trajectory x'_t. The result y_{cf} is the counterfactual outcome—what Y would have been for this specific individual under the hypothetical intervention.

A.1 Algorithm

The compilation process consists of three stages: Validation, Ordering, and Dependency Resolution.

1. Validation The compiler first verifies that the requested query is well-posed.

- **Variable Validity:** Targets Θ and Conditions Y must be valid subsets of the PGM variables.
- **Disjoint Sets:** Targets Θ and Conditions Y must be disjoint.

2. Variable Ordering To construct a valid flow, we must order the variables $[z_1, \ldots, z_K]$ such that no variable depends on a future one.

Algorithm 1 Variable Ordering Scheme

Input: PGM Dag $\mathcal{G}$, Conditions Y
Output: Ordered List $\mathcal{O}$
Compute Topological Sort $\mathcal{T}$ of $\mathcal{G}$
Construct Moral Graph $\mathcal{G}_m$ from $\mathcal{G}$
Sort Y by decreasing index in $\mathcal{T}$ (Reverse Topological)
Initialize $\mathcal{O} = Y$, Queue $Q = Y$
while Q is not empty **do**
  $u \leftarrow Q$.pop()
  Get neighbors of u in $\mathcal{G}_m$, sort by decreasing index in $\mathcal{T}$
  for neighbor v in sorted neighbors **do**
    if $v \notin \mathcal{O}$ **then**
      append v to $\mathcal{O}$ and Q
    end if
  end for
end while
Append remaining nodes from $\mathcal{T}$ in reverse order
return $\mathcal{O}$

We prioritize placing conditioning variables first so they can serve as fixed inputs, sorting them by reverse topological order. The subsequent BFS on the moral graph (exploring neighbors in reverse topological order) ensures that strongly coupled variables are grouped together, while the final reverse topological sort respects the dependencies of any remaining unconditioned sub-structures.

3. Dependency Pruning For every block z_k in the computed order, we determine its parent set $\mathrm{Pa}(z_k)$ using the d-separation criterion available in network analysis tools (e.g., NetworkX).

$$j \in \mathrm{Pa}(z_k) \iff z_k \not\perp_d z_j \mid \{z_1, \ldots, z_{k-1}\} \setminus \{z_j\} \tag{11}$$

This step ensures strict minimality of the flow's dependency graph.

A.2 Tractability

For a detailed theoretical derivation of the tractability conditions for marginals and conditionals in Block-Triangular Flows, please refer to Theorem 1 in Appendix D.

A.3 Examples

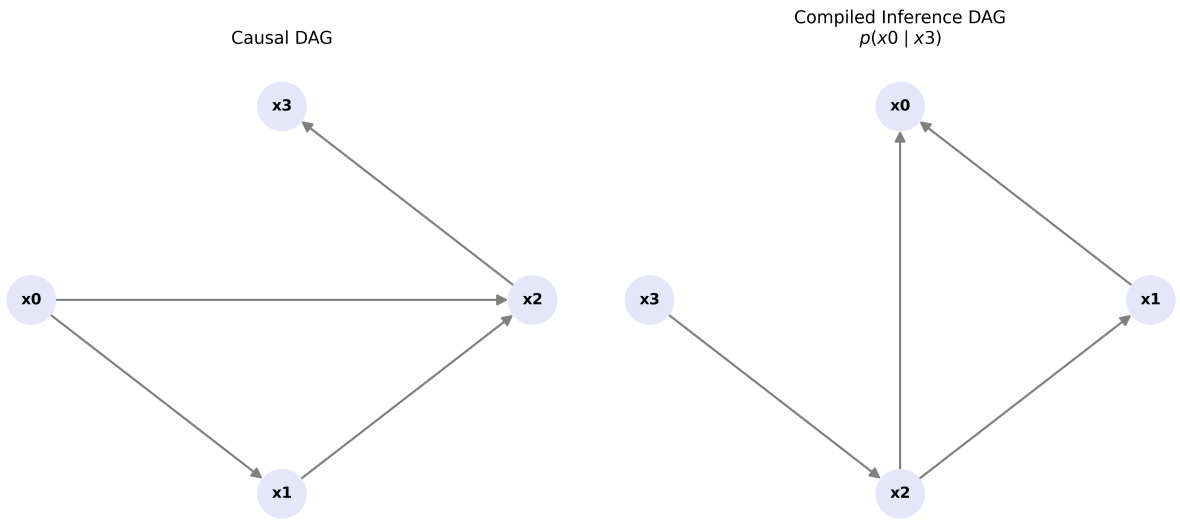


Figure 7: An example of StructFlow ($n = 4$, Seed 41). We start with the given structure / DAG. This can be used to define the dependencies within our structured flows (and competitors) - top row. Alternatively, we can compile these structures for inference, yielding a different DAG, which supports inference queries.

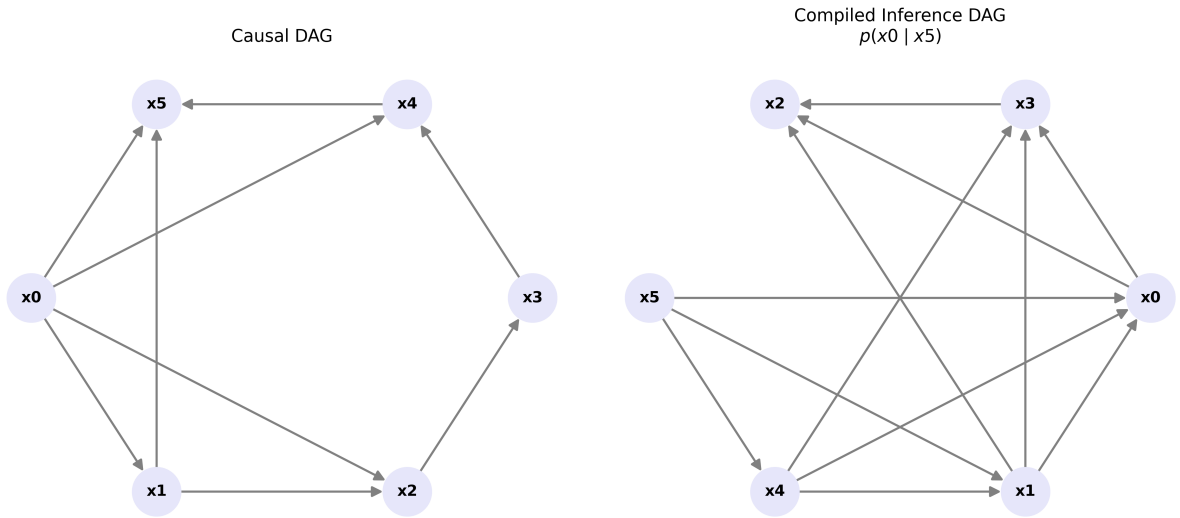


Figure 8: An example of StructFlow ($n = 6$, Seed 41). We start with the given structure / DAG. This can be used to define the dependencies within our structured flows (and competitors) - top row. Alternatively, we can compile these structures for inference, yielding a different DAG, which supports inference queries.

B Appendix: Architecture and Implementation

In this section, we describe the neural network architecture and optimization techniques used to parameterize the block-triangular flows.

B.1 Loss Function and Optimization

For all models, we use the Adam optimizer (Kingma & Ba, 2015) with a warmup-cosine decay learning rate schedule. This schedule provides an initial linear warmup period followed by a cosine annealing phase to ensure stable convergence.

The training objective for our one-sided flows follows the denoising-style loss for η_1-prediction:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1), y_1\sim p_{data}, \epsilon\sim\mathcal{N}(0,I)} \left[\|\hat{\eta}_1(y_t, t; \theta) - y_1\|^2 \right] \tag{12}$$

where $y_t = ty_1 + (1-t)\epsilon$ is the linear interpolant.

B.1.1 Optimal Transport Coupling

To accelerate convergence and reduce the variance of the gradient estimators, we employ optimal transport (OT) couplings to pair noise samples ϵ and data samples y_1 in the training mini-batches. Specifically, we use an Entropic Optimal Transport coupling (*Sinkhorn*) (Cuturi, 2013) which encourages straighter flow trajectories. This makes the velocity fields easier for the neural network to approximate and reduces the discretization error during numerical ODE integration.

B.2 Neural Network Architecture

For each variable in the PGM, we use a conditional block which computes the output (velocity or eta1 prediction) of that variable given its current state y_t, the diffusion time t, and the values of its parent variables $\{p_i\}$ in the dependency graph.

B.2.1 Residual Block Architecture

Across all our conditional blocks and networks, we employ a consistent architecture based on residual layers. The key design is:

- **Time Embedding:** The scalar time t is embedded into a high-dimensional vector using Fourier features (random Gaussian projection followed by sin/cos transformations).
- **Input Projection:** The concatenated input (local state, time embedding, and parent states) is projected to a hidden dimension.
- **Residual Stack:** A stack of 8 residual layers, where each layer computes $h \leftarrow h + \sigma(\text{Linear}(\text{LayerNorm}(h)))$. We use Layer Normalization (Ba et al., 2016) and the Swish (SiLU) activation function (Ramachandran et al., 2017; Elfwing et al., 2018) consistently throughout.
- **Output Projection:** A final linear layer projects the hidden representation to the target dimension.

This modular residual design is used consistently across all network variants (static conditioning, guidance, CAREFL conditioning, and parallel flows).

To further ensure comparability of the varying architectures (e.g., a single large joint flow versus many small conditional blocks), we adopt a parameter-budgeting approach. For a given budget (e.g., 100,000 parameters), we perform a search over the hidden width W such that the total number of trainable weights in the architecture most closely matches the budget.

B.3 Structured Flows

We verify that our models have the intended structure. And example is shown in 9.

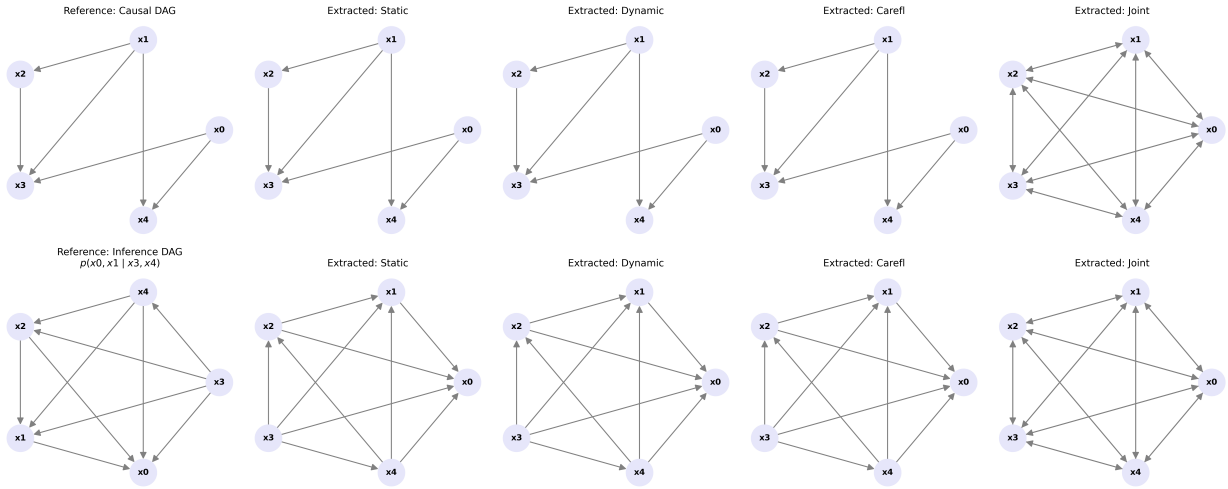


Figure 9: Given a PGM, we extract the DAG. This DAG is either; used as the structure of a causal flow architecture, or transformed into an inference aligned DAG and used to construct a inference capable flow architecture.

C Appendix: Algorithms for Sampling and Inference

In this section, we provide the detailed algorithms for sampling and inference using our Structured Flow Matching framework. We distinguish between two conditioning paradigms: `Static` (for sequential sampling) and `Dynamic` (for parallel sampling).

C.1 Static Conditioning

C.1.1 Static Sampling

In the Static mode, variables are sampled sequentially following a topological ordering of the dependency graph. Each variable's flow is conditioned on the *final* values ($t = 1$) of its parents. This minimizes the number of variables in any single ODE solve but requires K separate integration passes for K target variables.

Algorithm 2 Static Conditional Sampling

Require: Trained Velocity Field v_ϕ, Observations c, Target variables Θ
Ensure: Samples $\theta_{t=1}$
  $Order \leftarrow$ TopoSort(Θ with respect to Parents(Θ))
  $Context \leftarrow c$ (Initial static conditions)
  for $y \in Order$ **do**
    Sample noise $z_y \sim \mathcal{N}(0, I)$
    Define single-block ODE: $\frac{dy}{dt} = v_\phi^{(y)}(t, y, Context)$
    Integrate $t : 0 \to 1$ starting from $y(0) = z_y$
    $Context[y] \leftarrow y(1)$ (Update context for downstream blocks)
  end for
  return $Context[\Theta]$

C.1.2 Static Density Evaluation

For static conditioning, the conditional log-probability $\log p(y_{targets}|y_{conditions})$ is computed by integrating the flow ODE backward from $t = 1$ to $t = 0$. Based on Proposition 2 in Appendix D, we compute the log-density by accumulating the divergence of the target variables only. Because the flow has a block-triangular Jacobian,

the log-determinant term decomposes into a sum over individual blocks, allowing us to ignore the divergence of the conditioning variables.

C.2 Dynamic Conditioning

C.2.1 Dynamic Sampling

In the Dynamic mode, all variables are integrated simultaneously in a single monolithic ODE solve. This requires reproducing the full temporal trajectories ($t \in [0, 1]$) of the conditioning variables. While this approach requires a more complex trajectory-interpolation step, it allows the sampling process to be parallelized across blocks, which is advantageous for deep or highly connected graphs.

Algorithm 3 Dynamic Conditional Sampling

Require: Trained Velocity Field v_ϕ, Observations c, Target variables Θ
Ensure: Samples θ_{pred}
  $\mathcal{B} \leftarrow$ AllBlocks() (All variable blocks in the model)
  $\mathcal{C} \leftarrow$ Keys(c) (Condition blocks from observations)
  $\mathcal{S} \leftarrow$ RequiredSimulationBlocks($\Theta, \mathcal{C}$) (Ancestors of Θ not in $\mathcal{C}$)
  $z_{dict} \leftarrow \{\}$
  if conditions c provided **then**
    Construct $x^{(1)}_{cond}$ with observed values $c[\mathcal{C}]$ and zeros elsewhere
    Integrate backward $t : 1 \to 0$: $\frac{dx}{dt} = v_\phi(t, x)$ from $x(1) = x^{(1)}_{cond}$
    Extract $z_{cond} \leftarrow z(0)$ for condition blocks only
    $z_{dict} \leftarrow z_{cond}$
  end if
  ToSample $\leftarrow \{b \in \mathcal{S} : b \notin z_{dict}\}$ (Blocks needing fresh noise samples)
  for $b \in$ ToSample **do**
    Sample $z_b \sim \mathcal{N}(0, I)$
    $z_{dict}[b] \leftarrow z_b$
  end for
  Fill remaining blocks: For $b \in \mathcal{B} \setminus z_{dict}$, sample or set to zero
  Construct $x^{(0)} \leftarrow$ Flatten(z_{dict}) (Concatenate all noise blocks in order)
  Integrate forward $t : 0 \to 1$: $\frac{dx}{dt} = v_\phi(t, x)$ from $x(0) = x^{(0)}$
  $x^{(1)} \leftarrow x(1)$
  return Unflatten($x^{(1)}$)[Θ] (Extract target variables)

The zero padding is allowed as parents are not dependent on children, and the conditions are guaranteed to be parents. Therefore, the zero padding has no effect on the backward simulation of the conditions.

C.2.2 Dynamic Density Evaluation

In the Dynamic (parallel) mode, density evaluation requires reconstructing the full temporal trajectories of the conditioning variables. We first integrate the full system (targets + conditions) backward from $t = 1$ to $t = 0$ to obtain the trajectories $\tau_{all}(t)$. We then define a subset velocity field $v_T(y_T, t; \tau_C)$ where the conditioning variables are interpolated from their backward trajectories. Based on Proposition 2 in Appendix D, we compute the log-density by accumulating the divergence of the target variables only, properly accounting for the dynamic coupling between variables without integrating the full joint density.

D Appendix: Theoretical Proofs

D.1 Proof of Theorem 1 (Asymmetry of Marginalization)

Theorem 1. *In a Block-Triangular Flow ordered $z_1 \to \cdots \to z_K$, the marginal log-density $\log p(z_S)$ of a subset S is* directly tractable*—evaluable by the change-of-variables formula restricted to S, in a single pass and with no integration over the remaining variables—if and only if S is ancestrally closed (contains all parents, and hence all ancestors, of each of its members). A topological prefix is the canonical ancestrally-closed set.*

Proof. Let the variables z be partitioned into K blocks, ordered topologically. The flow transformation $\Phi : z(0) \to z(1)$ is defined by the ODE $\frac{dz}{dt} = v_t(z)$.

1. Forward Direction (Ancestrally Closed is Tractable) Let S be ancestrally closed. Due to the block-triangular structure, the velocity of any block $i \in S$ depends only on i and its parents, all of which lie in S:

$$v^{(i)}(z,t) = f\big(\{z^{(j)} : j \in \{i\} \cup \mathrm{Pa}(i)\}, t\big), \qquad \{i\} \cup \mathrm{Pa}(i) \subseteq S. \tag{13}$$

Consequently, the time-evolution of z_S is entirely self-contained; its trajectories do not depend on the variables outside S. We can treat z_S as a standalone flow system, and the marginal log-density is given by the change of variables formula restricted to S:

$$\log p(z_S(1)) = \log p(z_S(0)) - \int_0^1 \sum_{i \in S} \mathrm{Tr}\left(\frac{\partial v^{(i)}}{\partial z^{(i)}}\right) dt \tag{14}$$

Since all terms in the divergence sum are computable using only variables in S, the marginal density can be evaluated in a single pass without integration over the remaining variables. A prefix $S = \{1, \ldots, k\}$ is a special case.

2. Reverse Direction (Not Ancestrally Closed) Suppose some block $j \in S$ has a parent $i \notin S$, and write $\bar{S}$ for the complement. The joint factorizes as $p(z_S, z_{\bar{S}}) = p(z_S \mid z_{\bar{S}})\, p(z_{\bar{S}})$, and the marginal requires integrating out $\bar{S}$:

$$p(z_S) = \int p(z_S | z_{\bar{S}})\, p(z_{\bar{S}})\, dz_{\bar{S}}. \tag{15}$$

In a non-linear flow the conditional $p(z_S|z_{\bar{S}})$ is a nonlinear function of $z_{\bar{S}}$ with no closed-form integral; a Monte Carlo estimate would re-sample $z_{\bar{S}}$, re-incurring the cost of the full joint model. Hence the marginal is not directly tractable and is intractable in general. (We claim intractability of *exact, integration-free* evaluation, not a worst-case complexity lower bound.) □

D.2 Proof of Proposition 1 (Structural Preservation)

Proposition 1. *If the denoiser model $\hat{\eta}_1(y,t)$ respects a block-triangular dependency graph, then the induced velocity field $v_t(y)$ also respects the same block-triangular dependency graph.*

Proof. Let the state y be partitioned into blocks $y^{(1)}, \ldots, y^{(K)}$. By definition, a function is block-triangular if the k-th block outcome depends only on the prefix $y^{(1:k)} = [y^{(1)}, \ldots, y^{(k)}]$.

1. **Assumption:** Our neural network $\hat{\eta}_1$ is structured such that:

$$\hat{\eta}_1^{(k)}(y,t) = \mathrm{NN}^{(k)}(y^{(1)}, \ldots, y^{(k)}, t) \tag{16}$$

2. **Affine Property:** For the linear interpolant used in this work ($y_t = (1-t)y_0 + ty_1$), the velocity field $v_t(y)$ is an affine combination of the input y and the denoiser $\hat{\eta}_1$:

$$v_t(y) = \frac{\hat{\eta}_1(y,t) - y}{1-t} \tag{17}$$

This can be written more generally as $v_t(y) = c_1(t)y + c_2(t)\hat{\eta}_1(y,t)$. (The explicit $1/(1-t)$ form is singular at $t = 1$, but the argument below uses only the affine-in-$\big(y, \hat{\eta}_1\big)$ structure, i.e. the coefficients c_1, c_2, which is the well-defined form; the block-triangularity conclusion does not depend on the value at the endpoint.)

3. **Preservation:** The k-th block of the velocity field is:

$$v_t^{(k)}(y,t) = c_1(t)y^{(k)} + c_2(t)\hat{\eta}_1^{(k)}(y^{(1:k)}, t) \tag{18}$$

Since $\hat{\eta}_1^{(k)}$ only depends on the prefix $y^{(1:k)}$, and the term $y^{(k)}$ is itself part of that prefix, it follows that $v_t^{(k)}$ depends strictly on $y^{(1:k)}$.

Thus, the block-triangularity of the denoiser implies the block-triangularity of the velocity field. □

D.3 Derivation of Conditional Log-Density

Proposition 2. *For a Block-Triangular Flow in which the conditioning variables Y precede the targets Θ in the ordering (so that, by the block-triangular structure, the velocity of Y does not depend on Θ), the conditional log-density $\log p(\Theta|Y)$ is computed by accumulating the divergence of the velocity field with respect to Θ only.*

Proof. Consider the joint system $z = [Y, \Theta]$ ordered such that Y precedes Θ. The flow is defined by the system of ODEs:

$$\frac{dY}{dt} = v_Y(Y,t) \tag{19}$$

$$\frac{d\Theta}{dt} = v_\Theta(\Theta, Y, t) \tag{20}$$

To evaluate $\log p(\Theta(1)|Y(1))$, we condition on the observed trajectories of $Y(t)$ for $t \in [0,1]$. The block-triangular structure is exactly the hypothesis doing the work here: because Y precedes Θ, the velocity v_Y takes no Θ input, so Y's trajectory is independent of Θ. This holds for both conditioning paradigms—in static conditioning Y is integrated to completion first, and in dynamic conditioning Y co-evolves but its dynamics still do not see Θ. We can therefore view the evolution of Θ not as a coupled part of a joint system, but as a **non-autonomous** system where $Y(t)$ acts as a time-dependent external forcing term:

$$\frac{d\Theta}{dt} = \tilde{v}_t(\Theta) = v_\Theta(\Theta, Y(t), t) \tag{21}$$

The Jacobian of this conditional transformation $\Phi_Y : \Theta(0) \to \Theta(1)$ is simply the sub-block of the full Jacobian corresponding to the target variables:

$$J_\Theta = \frac{\partial \tilde{v}_t}{\partial \Theta} = \frac{\partial v_\Theta}{\partial \Theta} \tag{22}$$

Applying the instantaneous change of variables formula to this subsystem yields:

$$\log p(\Theta(1)|Y(1)) = \log p_0(\Theta(0)) - \int_0^1 \mathrm{Tr}\left(\frac{\partial v_\Theta}{\partial \Theta}(\Theta(t), Y(t), t)\right) dt \tag{23}$$

This confirms that the volume change (divergence) of the conditioning variables Y does not need to be computed or integrated, significantly reducing the computational cost for high-dimensional conditioning. □

D.4 Expressivity Gap: CAREFL vs. CNF

Proposition 3 (Expressivity Gap). *Let the true SCM be $Y = f(\mathrm{Pa}(Y), \epsilon)$ where $\frac{\partial^2 Y}{\partial \epsilon^2} \neq 0$ (i.e., the mechanism is nonlinear in the exogenous noise ϵ). Then CAREFL—an affine autoregressive flow under a fixed ordering—cannot approximate this mechanism with vanishing error, whereas CNFs can achieve arbitrarily small approximation error as capacity increases.*

Proof. CAREFL generates each node by an affine transformation of its own exogenous noise:

$$Y = a(\mathrm{Pa}(Y)) \cdot \epsilon + b(\mathrm{Pa}(Y)) \tag{24}$$

where a, b are arbitrary neural networks of the parent values. Crucially, this is not an artifact of using a single layer: Khemakhem et al. (2021) show (their Appendix C) that a composition of affine autoregressive flows under a fixed ordering is *itself* an affine autoregressive flow, so stacking layers preserves this form. The map is therefore **strictly linear in** ϵ:

$$\frac{\partial^2 Y}{\partial \epsilon^2} = 0 \quad \text{for all } a, b \tag{25}$$

If the true mechanism satisfies $\frac{\partial^2 Y}{\partial \epsilon^2} \neq 0$, then no choice of a and b can match the true function, and the approximation error is bounded away from zero regardless of the capacity of the networks a and b.

In contrast, Continuous Normalizing Flows (CNFs) parameterize the transformation as the solution to an ODE:

$$Y = \phi_1(\epsilon) = \epsilon + \int_0^1 v_t(\phi_t(\epsilon), \mathrm{Pa}(Y), t)\, dt \tag{26}$$

For the scalar exogenous noise of a structural equation, the flow of a Neural ODE realizes an arbitrary smooth *monotone* map $\epsilon \mapsto Y$, which is precisely the class of invertible scalar mechanisms; more generally, Neural ODEs are universal approximators of homeomorphisms given dimension augmentation (Zhang et al., 2020). In either case the mechanism can be approximated to arbitrary precision, so the approximation error $\to 0$ as the capacity of v_t increases. □

Remark D.1 (The gap holds for the full CAREFL model, not just one layer). Because composing affine autoregressive flows under a fixed ordering yields another affine autoregressive flow (Khemakhem et al., 2021, Appendix C), the identity $\partial^2 Y / \partial \epsilon^2 = 0$ holds for the entire stacked CAREFL model used in practice, not merely a single layer; the proposition is therefore not an artifact of restricting the baseline's depth. The companion fact that affine/additive flows are not universal density approximators is their Appendix B. The CNF escapes this constraint while retaining the single-transform structural reading on which our abduction step for counterfactuals (Section 3.5) relies.

E Appendix: Experimental Setup

E.1 Benchmarks

We evaluate our framework on two classes of synthetic Probabilistic Graphical Models (PGMs): Linear Gaussian PGMs and Non-linear PGMs with neural mechanisms. These benchmarks allow us to verify the correctness of the PGM Compiler and the accuracy of the learned flows against ground-truth distributions.

E.1.1 Linear Gaussian PGMs

The Linear Gaussian PGM is defined by the following structural equation:

$$x = Wx + b + S\epsilon \tag{27}$$

where $x \in \mathbb{R}^D$ is a vector of D variables, $W \in \mathbb{R}^{D \times D}$ is a strictly lower-triangular weight matrix representing the causal dependencies, $b \in \mathbb{R}^D$ is a bias vector, and $\epsilon \sim \mathcal{N}(0, I)$ is exogenous Gaussian noise. The matrix S scales the noise for each variable.

In our experiments, we use random PGMs which we generate by sampling edges from a lower-triangular matrix with a specified sparsity s. Weights and biases are sampled from a standard normal distribution, and noise scales are sampled uniformly from $[0.1, 1.0]$.

The joint distribution of a Linear Gaussian PGM is a multivariate Gaussian $p(x) = \mathcal{N}(\mu, \Sigma)$, where:

$$\mu = (I - W)^{-1}b, \quad \Sigma = (I - W)^{-1}SS^T(I - W)^{-T} \tag{28}$$

This analytical form allows for exact computation of all marginal and conditional distributions for comparison.

E.1.2 Non-linear Noise PGMs

To test the framework's ability to handle complex, non-linear dependencies, we implement a class of PGMs where mechanisms are **nonlinear functions of both parents and noise**. For each node i, the mechanism is:

$$x_i = f_i(\mathrm{Pa}(x_i), \epsilon_i) \tag{29}$$

where f_i is an invertible transformation with respect to the noise $\epsilon_i \sim \mathcal{N}(0, 1)$. This design produces **truly non-Gaussian conditionals** $p(x_i|\mathrm{Pa}(x_i))$, unlike standard non-linear PGMs where the conditional is necessarily Gaussian.

Mechanism Architecture: The mechanism f_i is implemented as a **rational quadratic spline** transformation, conditioned on parent values. The spline parameters (widths, heights, and derivatives) are computed by a small neural network:

$$(\text{widths, heights, derivatives}) = \mathrm{MLP}_\theta(\mathrm{Pa}(x_i)) \tag{30}$$

Each MLP consists of 2 hidden layers with a width of 32 units and uses **tanh** activation functions. The spline operates within a bounded domain $[-\text{bound}, \text{bound}]$ (typically bound $= 4.0$) with a configurable number of bins K (typically $K = 8$). Outside the bounded domain, the spline behaves as the identity transformation.

Log-Probability Computation: Unlike standard non-linear PGMs where the conditional is always Gaussian, the true conditional distribution here is non-Gaussian. We compute log-probabilities using the change of variables formula:

$$\log p(x_i|\mathrm{Pa}(x_i)) = \log p(\epsilon_i) - \log \left| \frac{\partial f_i}{\partial \epsilon_i} \right| \tag{31}$$

where $\epsilon_i = f_i^{-1}(x_i, \mathrm{Pa}(x_i))$ is the recovered noise obtained by inverting the spline transformation. The invertibility of the rational quadratic spline ensures that both the forward transformation and its inverse can be computed efficiently with accurate log-determinant Jacobian tracking.

Inference and Ground Truth: The marginals and posteriors of these Non-linear Noise PGMs are not analytically tractable. To evaluate our flows, we use **No-U-Turn Sampling (NUTS)** via the `blackjax` library to generate high-fidelity ground-truth samples from the joint distribution. We use 1000 samples with 500 warmup steps for each evaluation query.

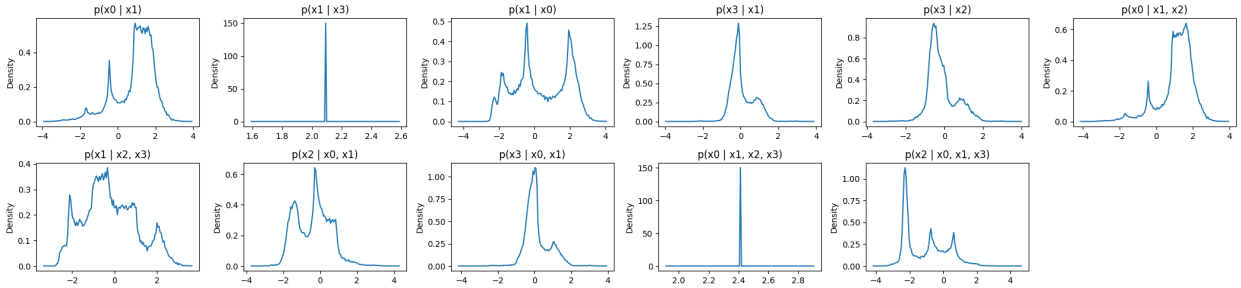


Figure 10: An example of a randomly sampled PGM with 4 nodes. Here we plot all the conditional distributions, with one target variable.

E.1.3 MNIST

The MNIST benchmark evaluates our framework on real-world image data, providing a practical test of the learned flows on high-dimensional observations. The MNIST dataset consists of 70,000 flattened grayscale handwritten digit images (28×28 pixels, i.e., 784 dimensions) with corresponding labels from 0 to 9.

PGM Structure and Variable Dimensions: The MNIST PGM follows a simple causal structure: label → image, meaning the digit label is the parent of the image data. The variable dimensions are:

- **Image**: 784 dimensions (flattened pixel values, typically normalized to $[0, 1]$)
- **Label**: 10 dimensions (one-hot encoded class labels)

Continuous Treatment of Labels: While the MNIST labels are inherently categorical (integers from 0 to 9), our flows treat the one-hot encoded label vectors as **continuous variables**. This easily allows the flow-based models to treat all variables uniformly within the continuous framework.

Dataset Split and Sampling: The dataset is partitioned into training (approximately 60,000 samples) and test (approximately 10,000 samples) splits, i.e. a 6:1 train/test ratio (the first 6/7 of the data are used for training). The `MNISTPGM` class provides conditional sampling via a `sample()` method:

- **Conditional sampling**: Given a label condition (either as integer index or one-hot encoding), the method samples uniformly from images matching that label within the specified train/test split.
- **Joint sampling**: Without conditions, the method samples both image and label pairs uniformly from the dataset.

E.2 Hyperparameters

Unless otherwise specified, all models (Joint, Static, and Dynamic) were trained using the same budget and optimization settings to ensure a fair comparison.

E.3 Baselines

E.3.1 CAREFL

We implement Causal Autoregressive Flows (CAREFL) (Khemakhem et al., 2021) as a baseline for structured causal modeling. Unlike the continuous-time ODE flows, CAREFL uses discrete affine transformations. For

Table 2: Hyperparameters used for synthetic PGM benchmarks.

Hyperparameter	**Value**
Optimizer	Adam
Initial Learning Rate	10^{-4}
Peak Learning Rate	10^{-3}
Ending Learning Rate	10^{-5}
Warmup Steps	10% of total
Batch Size	64
Architecture Depth (D)	8 layers

each node i with parents pa_i, the structural equation is:

$$y_i = g_i(pa_i) \cdot z_i + h_i(pa_i) \tag{32}$$

where $z_i \sim \text{Laplace}(0, 1)$ is the noise (following Khemakhem et al., 2021 for identifiability), and g_i, h_i are neural networks parameterized to output scaling and shift factors.

Implementation: We use the same `BlockTriangularStructure` to manage dependencies. The `log_prob` is computed by summing the base log-probabilities and the log-determinant terms ($-\sum \log |g_i|$). Sampling is performed by visiting nodes in topological order, computing g_i, h_i from the already sampled parents, and transforming sampled noise z_i.

E.3.2 Guided Baseline

Our generic baseline is a flow trained on the joint distribution, then conditioned on the required conditions using reconstruction guidance. The flow is trained on the unconditional joint distribution $p(y)$. During inference, to sample from $p(\Theta|Y_{cond})$, we use reconstruction guidance to steer the ODE trajectories towards the observed values Y_{cond}.

$$\mathcal{L}_{rec}(z) = \|z - \hat{z}\|^2 v(z, t) = v_{flow}(z, t) + \gamma\dot{\gamma}\nabla_z \mathcal{L}_{rec}(z) \tag{33}$$

where γ is from the interpolant noise schedule. This method allows for conditional sampling from a model trained only on the joint distribution, but lacks the structural guarantees of our framework.

F Appendix: Extended Results

F.1 Linear versus non-linear

We expect CAREFL to do well on the linear PGM problem as the structure of the PGM matches the assumptions / inductive bias within CAREFL - affine, exponential family. But since these are assumptions are restrictive, we expect CAREFL to perform poorly on the non-linear PGM problem.

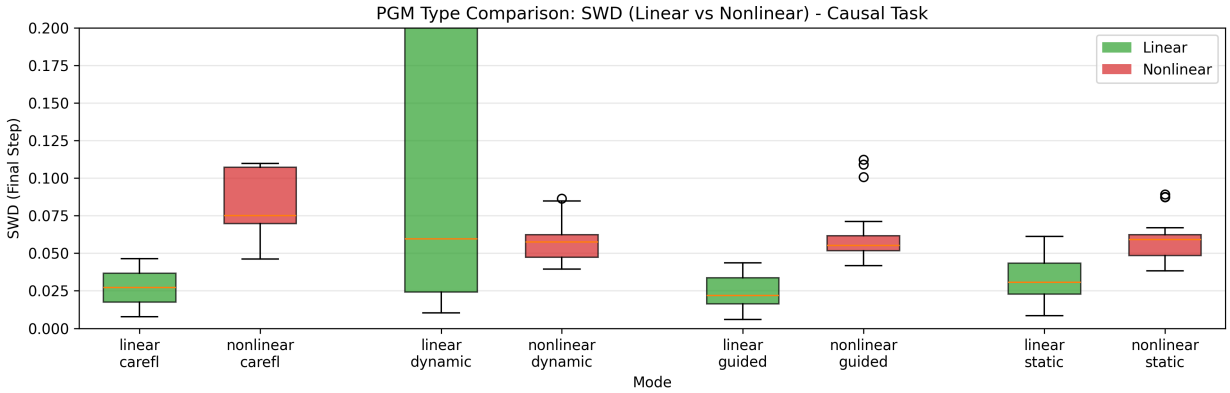


Figure 11: **Interventional Query Results.** Sliced Wasserstein Distance for conditional distributions $p(y|x)$ on linear and nonlinear PGMs. Results aggregated over 3 PGM seeds and 3 learner seeds.

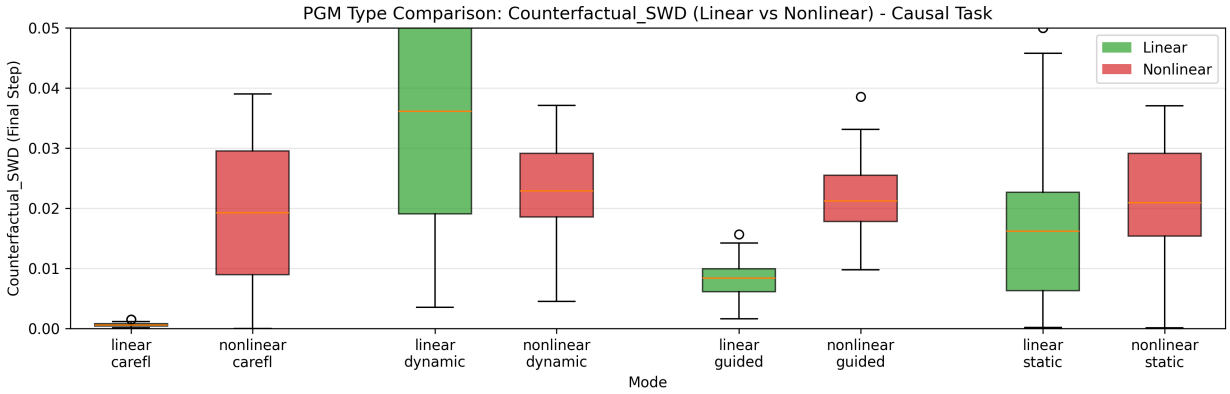


Figure 12: **Counterfactual Query Results.** Sliced Wasserstein Distance for counterfactual distributions on linear and nonlinear PGMs. Aggregated over 3 PGM seeds and 3 learner seeds.

F.2 Learning the joint distribution: causal structure versus inference structure

Is one direction (causal versus inference) better at learning the joint distribution than the other?

F.3 Granular PGM metrics

F.4 Scaling to 8-node PGMs

To check that the framework is not limited to very small graphs, we repeat the inference evaluation on random 8-node nonlinear PGMs (5 PGM seeds $\times$ 5 learner seeds). Figure 16 shows that the qualitative picture is unchanged from $n = 4$: all flow-based methods are closely comparable, with the structured (Static) flow tracking the unstructured baselines. This is consistent with our central finding—on well-sampled graphs,

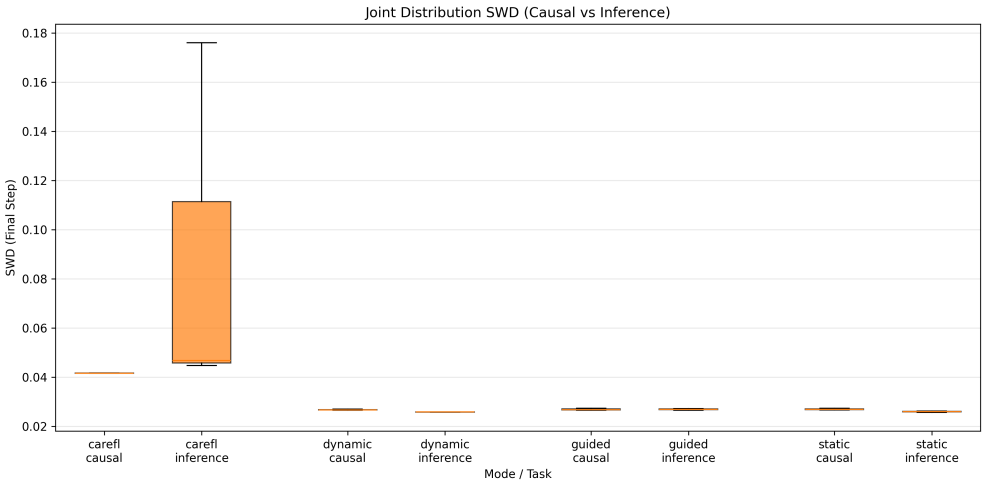


Figure 13: Non-linear PGM results. 4-node PGMs. Aggregated over all PGM seeds (n=3) and all learner seeds (n=3). Only the joint distribution SWD.

structure is a competitive but not dominant choice, and its advantage is most visible under limited data (Section 4.5). The binding constraint on scaling further is *evaluation*, not training: generating MCMC ground truth for every conditional query grows quickly with n (Section 5).

F.5 MNIST

Is the elevated dynamic-conditioning SWD an outlier artifact? Section 4 reports that dynamic conditioning attains a higher conditional SWD on $p(\text{image} \mid \text{label})$ than the other flows. We test whether this is driven by a few divergent pixels (which a squared-distance metric would over-weight) or by a genuine, broadly-distributed mismatch. Table 3 reports, for one representative trained model per method, the SWD recomputed after discarding the largest fraction of the squared sorted-difference terms that make up the SWD, together with the share of the total carried by the top 1% of terms. If outliers dominated, trimming would sharply reduce the score and the top-1% share would be anomalously high for dynamic conditioning. Neither holds: trimming the top 1% leaves the dynamic-vs-static ratio essentially unchanged ($3.05\times \rightarrow 3.07\times$), and dynamic conditioning's top-1% share ($\sim$10%) is no larger than that of the well-behaved static and guided baselines. The excess SWD is therefore spread across the distribution. Consistent with this, inspecting samples (Figures 19–21) shows that dynamic conditioning yields perceptually crisp digits that nonetheless sometimes render the wrong class for the conditioning label—a broadly-distributed conditional error, not a heavy tail. The diagnostic script is released with the code.

Table 3: **Outlier diagnostic for the MNIST conditional SWD** ($p(\text{image} \mid \text{label})$, one representative run per method, 1000 conditional samples). "Trim x%" recomputes the SWD after dropping the largest x% of its squared sorted-difference terms; "top-1% share" is the fraction of the total squared mass in those terms. Trimming barely moves any score and dynamic's top-1% share is typical, so the elevated dynamic SWD is a genuine distributional gap rather than an outlier artifact.

Method	**SWD**	**Trim 0.1%**	**Trim 1%**	**Top-1% share**
Guided	0.0118	0.0116	0.0111	11.8%
Universal	0.0119	0.0117	0.0109	17.6%
Static	0.0141	0.0140	0.0134	11.3%
CAREFL	0.0273	0.0270	0.0259	11.1%
Dynamic	0.0431	0.0427	0.0410	10.4%

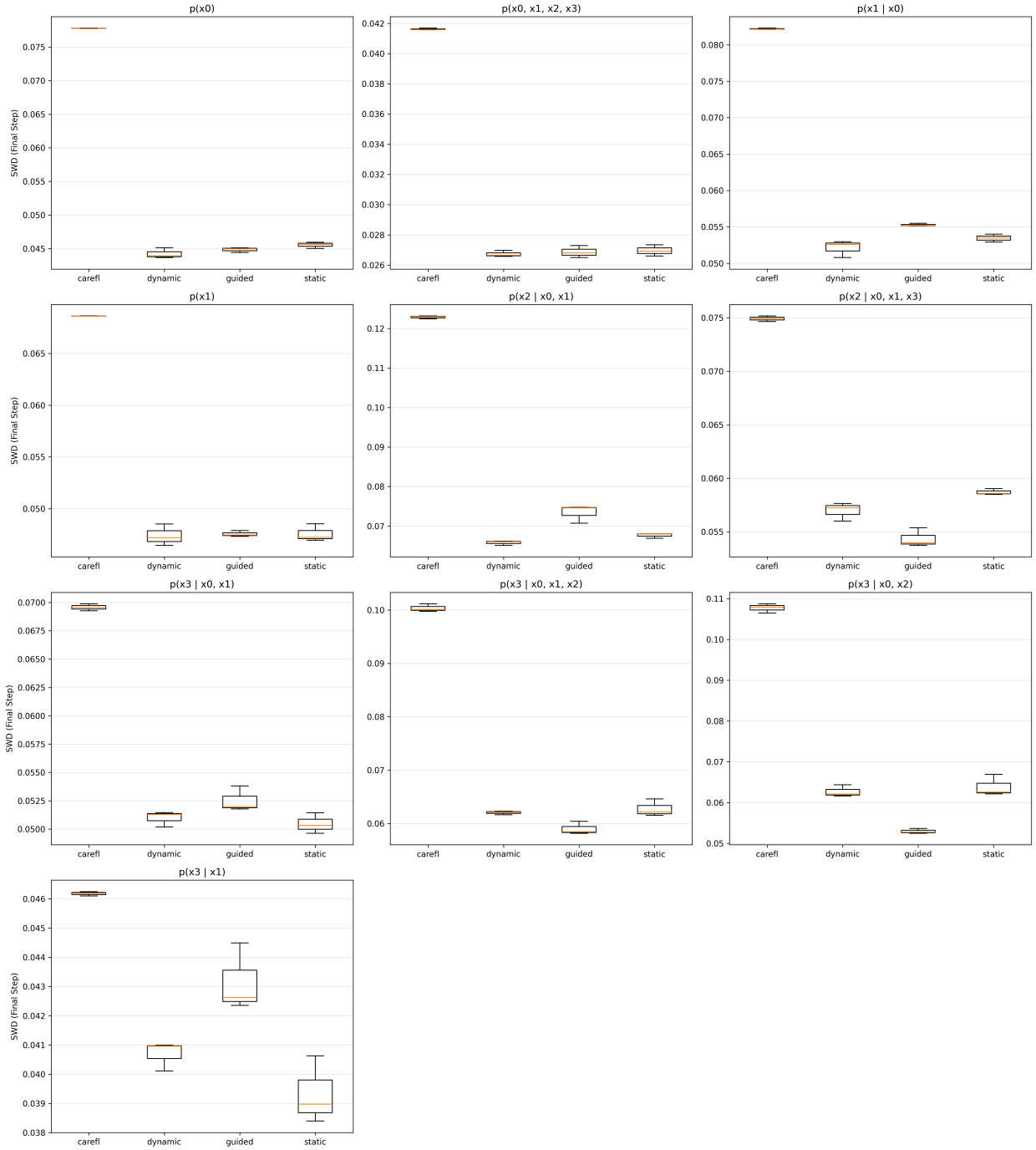


Figure 14: Nonlinear PGM results for the causal task. Aggregated over learner seeds (n=3).

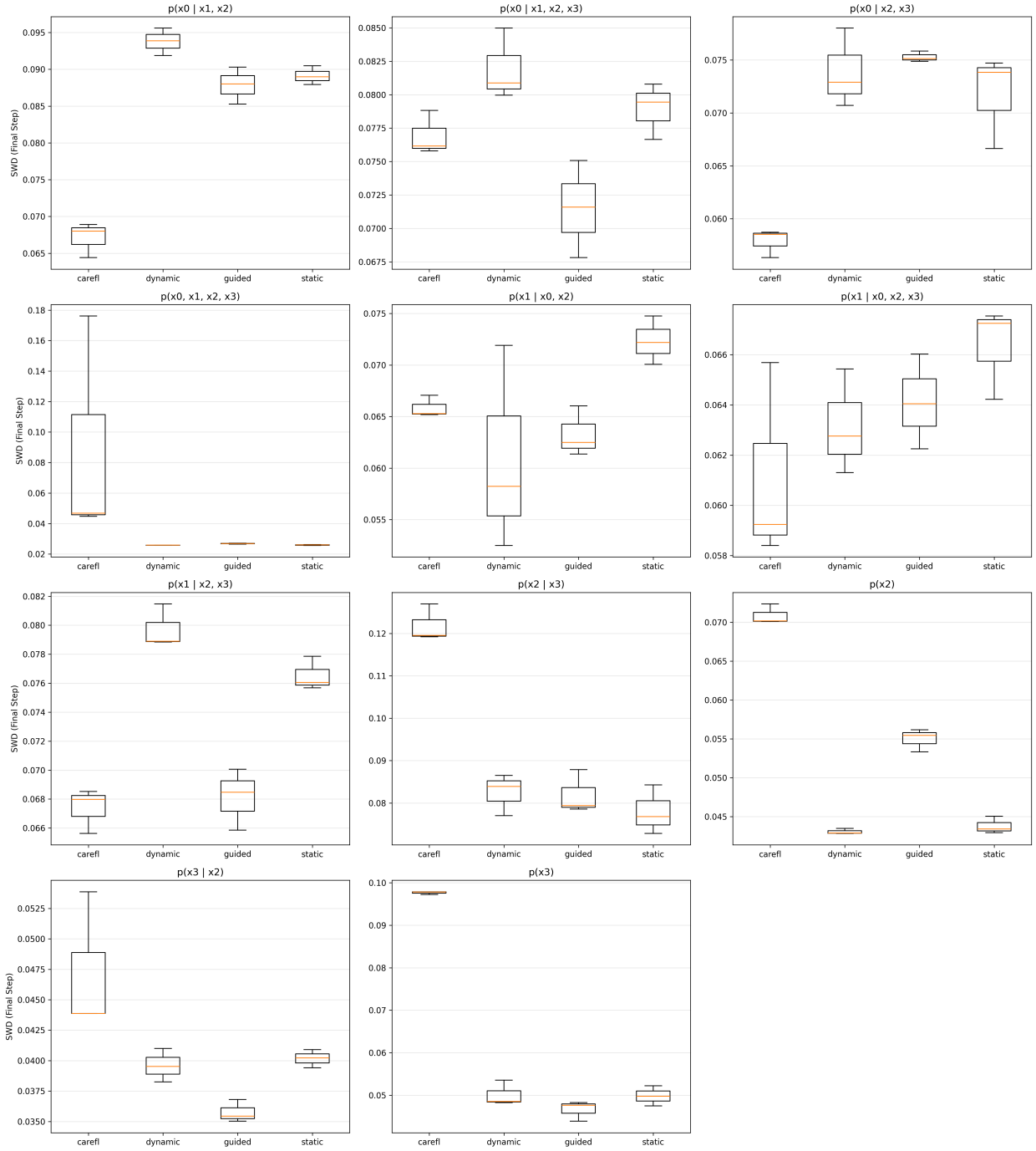


Figure 15: Nonlinear PGM results for the inference task. Aggregated over learner seeds (n=3).

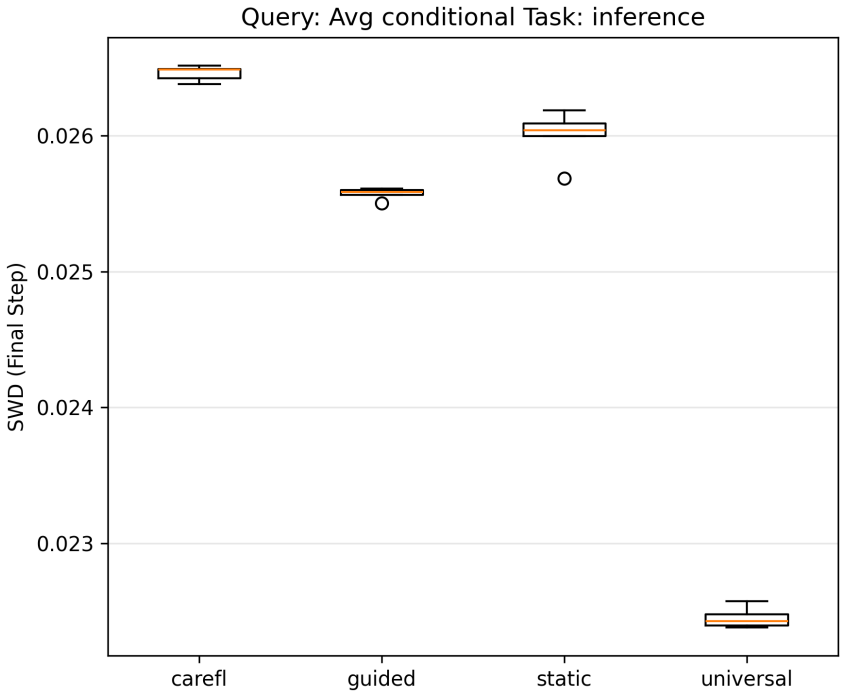


Figure 16: **Inference on 8-node nonlinear PGMs.** Normalized SWD on posterior queries, aggregated over conditional queries, 5 PGM seeds and 5 learner seeds. "Universal" denotes an unstructured joint-flow baseline. The structured (Static) flow remains competitive at $n = 8$, matching the $n = 4$ trend.

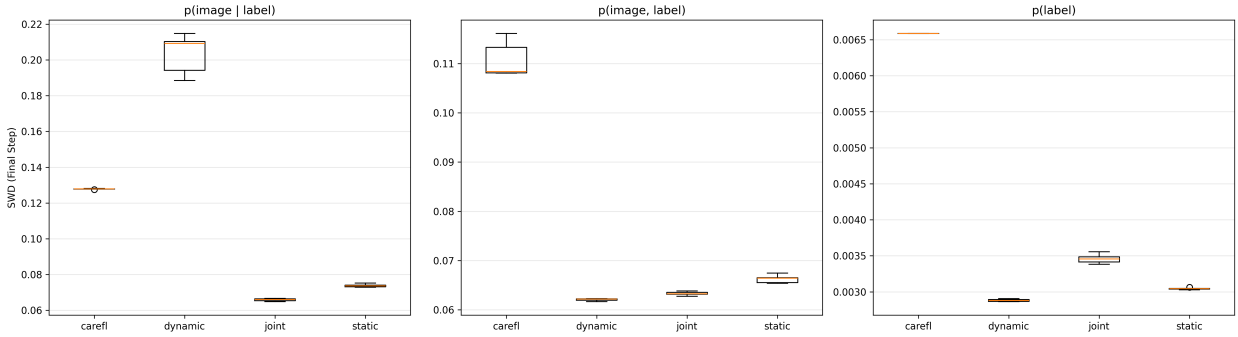


Figure 17: **MNIST Results.** Sliced Wasserstein Distance across joint, marginal, and conditional distributions for the structure `labels`→`images`. Aggregated over 5 seeds.

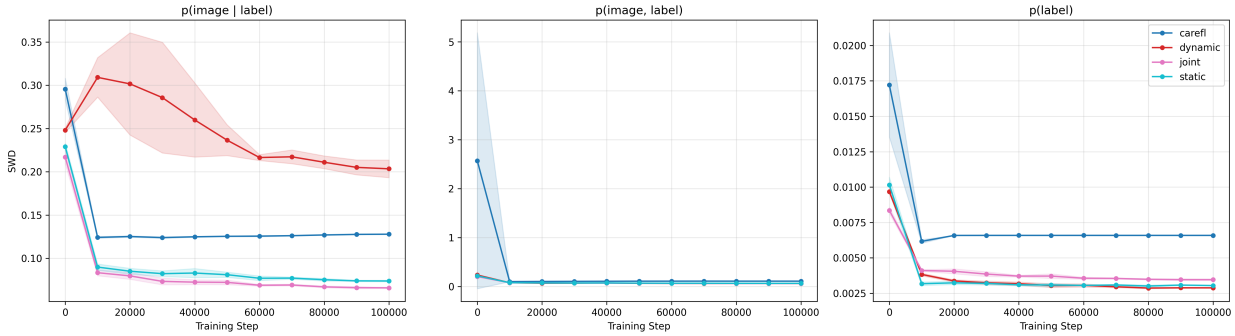


Figure 18: SWD evaluated on the test set as a function of training step. Aggregated over 5 seeds.

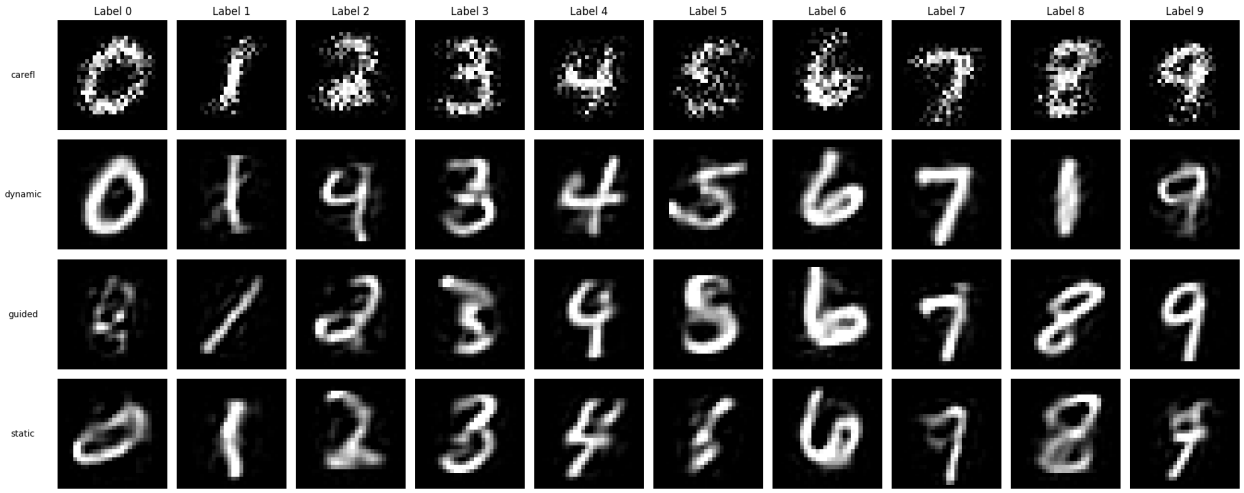


Figure 19: Samples from 4 models trained on MNIST.

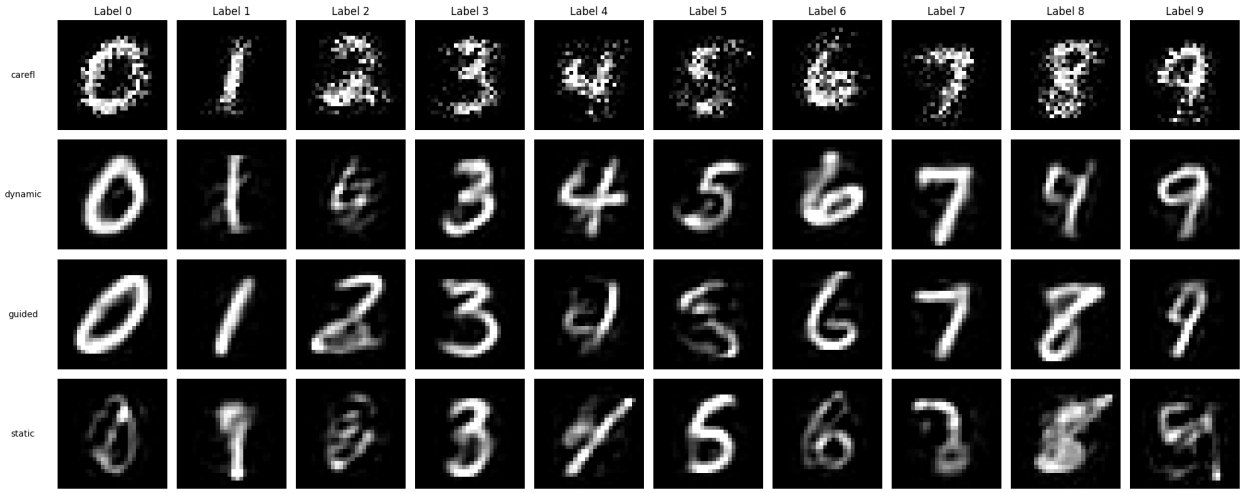


Figure 20: Samples from 4 models trained on MNIST.

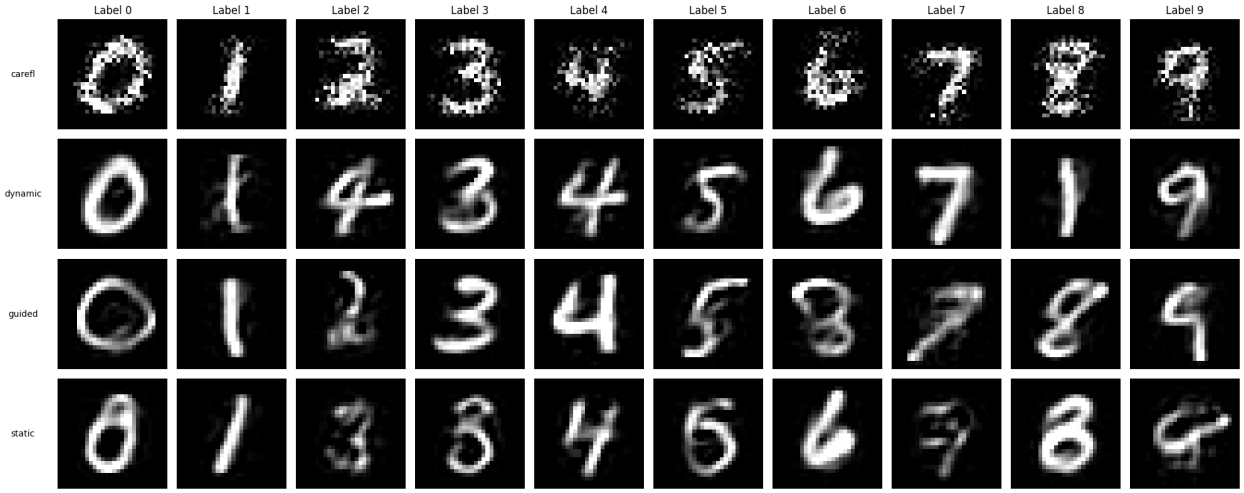


Figure 21: Samples from 4 models trained on MNIST.