# AMORTIZED FACTOR INFERENCE NETWORKS FOR POSTERIOR INFERENCE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Amortized inference promises fast test-time Bayesian inference, but existing methods typically target a fixed model or a restricted model family. Extending amortization to new model families often requires retraining or test-time fine-tuning. In this paper, we ask: is it possible to build a single inference network capable of generalizing across varying priors, likelihoods, latent dimensions, and observation counts? We introduce Amortized Factor Inference Networks (AFINs), which describe a model as a list of typed prior and likelihood factors and map this description, together with the observations, to a Gaussian posterior approximation through an encode–merge–decode network whose learned parameters do not depend on the latent dimension or the number of observations. Experiments on synthetic and real-data benchmarks show that a single trained AFIN provides competitive posterior approximations at substantially lower test-time cost than NUTS and per-task variational inference. Optional importance sampling can further improve accuracy.

## 1 INTRODUCTION

Simulation-based neural posterior estimation trains a network on simulated latent values and observations to predict an approximate posterior for a new dataset (Papamakarios and Murray, 2016; Greenberg et al., 2019; Radev et al., 2022; Cranmer et al., 2020). Maximum-likelihood training minimizes expected forward KL divergence from the true posterior to the approximation. This favors coverage of posterior mass, which suits later refinement by importance sampling (Jerfel et al., 2021).

However, training is expensive and usually specific to one model, so changing a prior or likelihood can require retraining. Recent work extends amortization to varying observation counts (Reuter et al., 2025; Gloeckler et al., 2024; Vetter et al., 2025), model parameters (Whittle et al., 2026), or probabilistic programs (Wu and Goodman, 2022). We ask whether one trained network can handle changes in prior and likelihood families, their parameters, latent dimensions, and observation counts jointly (see Table 1 for a comparison with existing methods).

We introduce *Amortized Factor Inference Networks* (AFINs), which take both the model and its observations as inputs. We consider models with one global latent vector and observations that are conditionally independent given it. Each model is described by one prior factor and one likelihood factor per observation, with a distribution type and numerical parameters for each factor. Each network is trained on a chosen set of factor types.

Conjugate exponential-family updates suggest a simple way to organize inference: represent each factor by its natural-parameter contribution, add the contributions, and recover the posterior. AFIN extends this idea beyond conjugate models with a learned **encoder** that maps each factor's inputs to an embedding, a **merge** that lets the embeddings interact through attention before summing them, and a **decoder** that maps the sum to a Gaussian posterior approximation. The factor inputs and predicted posterior parameters also change size with the latent dimension. We therefore use parameter-tied multilayer perceptrons (MLPs) and attention modules that share weights across latent coordinates (Zaheer et al., 2017; Maron et al., 2019), allowing input and output dimensions to change without changing the learned parameter shapes.

To train AFIN across models, we sample a model, draw a latent value from its prior, and generate observations from its likelihoods. Training maximizes the log density that the predicted posterior



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.

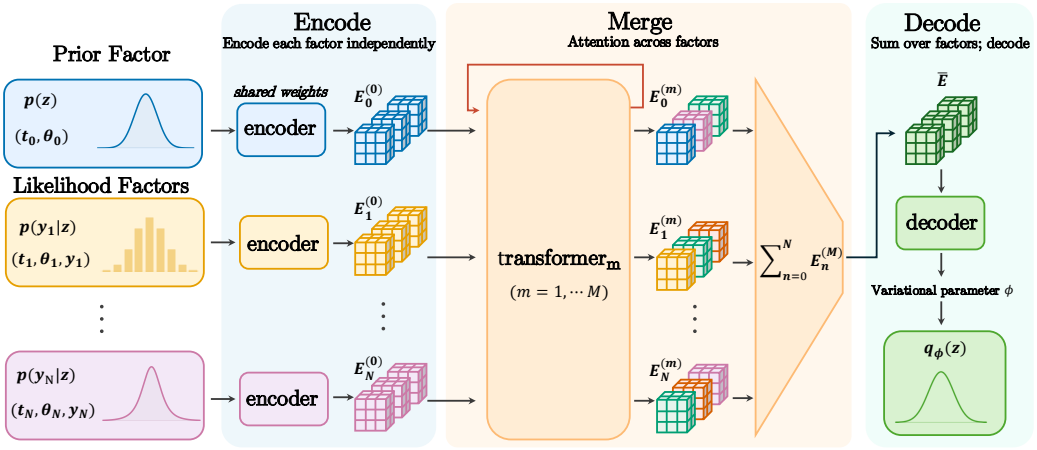



Figure 1: AFIN overview. The **encoder** maps the prior's type and parameters, and each likelihood's type, parameters, and observed value, to node and pair embeddings. The **merge** lets these embeddings exchange information through attention, then sums them over factors. The **decoder** maps the sum to the mean and precision of a Gaussian approximate posterior $q_\phi(z)$. Sharing weights across latent coordinates allows the same trained network to handle different dimensions.

assigns to the latent value that generated those observations. The latent value is known from simulation, so training requires no posterior estimates from another inference method.

Our contributions are:

- We introduce a factor-based approach that jointly amortizes inference across supported prior and likelihood families, parameter values, latent dimensions, and observation counts.
- We adapt permutation-equivariant MLPs and attention into an encode–merge–decode architecture whose learned parameter shapes are independent of the latent dimension and the number of observations.
- One trained AFIN produces posterior approximations in a single forward pass, matches much longer NUTS and variational runs after importance-sampling refinement at a fraction of their cost, extrapolates in the number of observations, and extends to a new likelihood type through a small adapter.

## 2 Problem Setup

Standard neural posterior estimation amortizes inference over datasets from a fixed probabilistic model. Let $m$ denote the model, $z$ its latent variables, and $y$ its observations, with joint distribution $p(z, y \mid m)$. An inference network $f_w$, with learned weights $w$, maps $y$ to the parameters $\phi$ of an approximate posterior $q(z; \phi)$. The network is trained on simulated pairs $(z, y)$ with the objective

$$\max_w \mathbb{E}_{p(z,y|m)} \left[\log q\big(z; f_w(y)\big)\right]. \tag{1}$$

The aim is for $q(z; f_w(y))$ to approximate $p(z \mid y, m)$ for new observations from this same model.

We seek to amortize inference over models as well as datasets. The network therefore receives both the observations $y$ and the model $m$. During training, we sample $m$ from a chosen distribution $p(m)$, then simulate $(z, y)$ from that model, giving the objective

$$\max_w \mathbb{E}_{p(m,z,y)} \left[\log q\big(z; f_w(y, m)\big)\right], \qquad p(m, z, y) = p(m)p(z, y \mid m). \tag{2}$$

The aim is for the same learned weights $w$ to yield accurate posterior approximations across the chosen range of models and datasets. At inference time, $m$ is known, and the target is $p(z \mid y, m)$.

We focus on models with one shared latent vector $z \in \mathbb{R}^D$ and $N$ observations $y = (y_1, \ldots, y_N)$ that are conditionally independent given $z$. We specify each model by $m = (D, N, t_{0:N}, \theta_{0:N})$, with

joint density

$$p(z, y \mid m) = p(z \mid t_0, \theta_0) \prod_{n=1}^{N} p(y_n \mid z, t_n, \theta_n), \tag{3}$$

where $(t_0, \theta_0)$ describes the prior factor and $(t_n, \theta_n)$ describes the likelihood factor for observation $y_n$. The label $t_n$ identifies the factor type, such as a Gaussian prior or a Bernoulli-logit likelihood. The known numerical parameters $\theta_n$ specify the particular factor, including quantities such as means, scales, or covariates. The type labels belong to a chosen finite set, and the parameters may include scalars, vectors, or matrices. Different observations may have different likelihood types.

A model with $D = 1$ and $N = 3$ can combine continuous and binary observations as follows

$$\begin{aligned}
&z \sim \mathcal{N}(0, \sigma_0^2), && t_0 = \texttt{"gaussian prior"}, && \theta_0 = (0, \sigma_0),\\
&y_1 \sim \mathcal{N}(z, \sigma_1^2), && t_1 = \texttt{"gaussian likelihood"}, && \theta_1 = (\sigma_1),\\
&y_2 \sim \mathrm{StudentT}_{\nu_2}(x_2 z, \sigma_2), && t_2 = \texttt{"student-t likelihood"}, && \theta_2 = (x_2, \nu_2, \sigma_2),\\
&y_3 \sim \mathrm{Bernoulli}(\mathrm{sigmoid}(x_3 z)), && t_3 = \texttt{"bernoulli-logit likelihood"}, && \theta_3 = (x_3).
\end{aligned}$$

Here, $\sigma_0, \sigma_1, \sigma_2 > 0$ are scales, $\nu_2 > 0$ is the degrees of freedom, and $x_2, x_3 \in \mathbb{R}$ are known covariates. All three likelihoods depend on the same latent variable $z$.

To define $p(m)$, we choose how to sample the latent dimension, observation count, factor types, and their parameters; Section 6.1 describes these choices and the dimension-normalized form of Equation (2) that we optimize in practice.

The central challenge is to build one inference network whose learned parameters work across these variations: the number of factors changes with $N$, their inputs differ by type, and the input and output dimensions change with $D$. The next section develops an architecture for this purpose.

# 3 Amortized Factor Inference Networks

## 3.1 Architectural motivation

Conjugate inference provides a simple template: summarize each factor, add the summaries, and recover the posterior from their sum. To make this precise, consider an exponential family $r(z \mid \eta) \propto \exp(\eta^\top T(z))$ with constant base measure. Here, $T(z)$ contains the sufficient statistics, and $\eta$ contains their coefficients, called natural parameters. Suppose the log prior and each log likelihood are linear in these statistics, up to terms independent of $z$. Denote their coefficient vectors by $\eta_{t_0}(\theta_0)$ and $\eta_{t_n}(\theta_n, y_n)$. Multiplying the factors adds these coefficients, giving the posterior update

$$p(z \mid y, m) = r\Big(z \;\Big|\; \eta_{t_0}(\theta_0) + \sum_{n=1}^{N} \eta_{t_n}(\theta_n, y_n)\Big). \tag{4}$$

Here, we **encode** each factor as its natural-parameter contribution, **merge** by addition, and **decode** the sum as a posterior distribution in the family $r$. For example, when the prior and all likelihoods are Gaussian in $z$, each factor contributes a vector in $\mathbb{R}^D$ and a matrix in $\mathbb{R}^{D\times D}$, and the posterior is the Gaussian whose precision-weighted mean and precision are the sums of these contributions.

Beyond conjugate models, log factors still add, but two things break. The sum need not correspond to any tractable family, and a tractable summary of one factor depends on where the other factors place their mass. AFIN preserves the encode–merge–decode organization while learning all three steps: how to represent each factor, how to let factors exchange information before they are combined, and how to map the result to a posterior approximation.

## 3.2 Architecture

The **encoder** maps each factor's inputs to an embedding, the **merge** lets the embeddings interact before summing them, and the **decoder** predicts posterior parameters $\phi = f_w(y, m)$. Figure 1 shows these three steps. This section explains what each step does and why; Section 4 and Section B give the constructions.

**Encoder.** In a conjugate model, the natural parameters of a factor contain all the information about that factor needed for inference, in a form shared by every factor. We want to preserve this uniform representation, since it is what allows a single merge step to combine factors of different types. However, realistic models do not have simple low-dimensional sufficient statistics. We therefore use high-dimensional embeddings of each factor, without requiring those embeddings to be the parameters of any simple family of distributions. This gives the network the flexibility to represent whatever information is useful for combining factors, without building in a conjugate assumption.

What size should the embedding of a factor be? For typical factors, the size of the parameters $\theta_n$ grows linearly or quadratically with the latent dimension $D$; a Gaussian prior, for example, has a mean vector and a precision matrix. We therefore use an embedding $E_n$ containing *node features* for each latent coordinate and *pair features* for each pair of coordinates

$$E_n = \left(E_n^{\text{node}}, E_n^{\text{pair}}\right), \qquad E_n^{\text{node}} \in \mathbb{R}^{D\times C}, \qquad E_n^{\text{pair}} \in \mathbb{R}^{D\times D\times C}, \tag{5}$$

where $C$ is a fixed number of learned features per coordinate or pair. For a Gaussian prior, the node features can hold the mean and the diagonal of the precision matrix, while the pair features can hold its off-diagonal entries, which describe dependence between coordinates. The same shapes serve every factor type, so factors as different as a Gaussian prior and a Bernoulli-logit likelihood are represented in a common format.

A single embedding network cannot be applied to factors of varying type $t_n$, because factor types differ in the number, shape, and meaning of the parameters $\theta_n$ and observations $y_n$. One could learn a separate encoder for each factor type. To reduce the number of type-specific parameters, we instead use a small type-specific *adapter* that produces an initial embedding, followed by a larger network that is shared across all factor types. The encoder output, the initial embedding of factor $n$, is

$$E_n^{(0)} = \texttt{encoder}(t_n, \theta_n, y_n) = \texttt{net}\big(\texttt{adapter}_{t_n}(\theta_n, y_n)\big), \tag{6}$$

where the prior factor has no observation and its adapter takes $\theta_0$ alone. The adapter handles the input format of its factor type, while the shared network performs most of the computation and lets information pass between node and pair features within the factor. Both `net` and $\texttt{adapter}_t$ share their learned parameters across latent coordinates, so they apply at any $D$ (Section 4).

**Merge.** In a conjugate model, merging is addition: the natural parameters of all factors are summed, and nothing else is needed. The direct analogue would be to sum the $N + 1$ factor embeddings. We instead let the embeddings interact through attention before summing them, for two reasons.

First, factors depend on each other. Outside conjugate models, a factor has no exact fixed-size summary, so its embedding is an approximation, and that approximation should be accurate where the posterior lies, which all factors decide together. A sum of independently computed embeddings can capture this dependence only indirectly, through the decoder acting on the sum; attention lets each factor's embedding be updated with information from all others before the sum.

Second, self-attention without positional encodings is equivariant to the order of its tokens, and the final sum makes the merged embedding invariant to it, which matches the model: observations are conditionally independent given $z$, so the posterior does not depend on their order.

We apply $M$ Transformer blocks, $\texttt{transformer}_m$ in Figure 1, treating each factor's node and pair arrays as one token, and then sum the outputs to obtain $\bar{E} = \sum_{n=0}^{N} E_n^{(M)}$. Both node and pair features enter the attention weights, and the projections are parameter-tied MLPs preserving the node and pair shapes, so the same blocks operate at any $D$ (Section 4). The sum removes the factor axis, giving the decoder one $D \times C$ node array and one $D \times D \times C$ pair array for any $N$, and keeps the additive form of the conjugate update once the factors have exchanged information.

**Decoder.** In a conjugate model, the summed natural parameters are already the parameters of the posterior, which lies in the prior's family, so decoding is a fixed conversion. Here, the merged embedding $\bar{E}$ is a collection of learned features, not the parameters of a distribution. The `decoder` maps it to an approximate posterior $q_\phi(z) = q(z; \phi)$ that supports the two operations we need: density evaluation for the training objective in Equation (2), and sampling for posterior inference. We use a Gaussian, $q_\phi(z) = \mathcal{N}(z; \mu, \Lambda^{-1})$, with mean $\mu \in \mathbb{R}^D$ and precision $\Lambda \in \mathbb{R}^{D\times D}$. Its parameters follow the structure of the embedding: shared MLPs predict the mean and the diagonal of $\Lambda$ from

per-coordinate features, and the off-diagonal entries of $\Lambda$ from per-pair features. To guarantee that $\Lambda$ is positive definite, the decoder symmetrizes it and sets each diagonal entry above the sum of the absolute off-diagonal entries in its row. This restricts $q_\phi$ to Gaussians with strictly diagonally dominant precision matrices. A Gaussian cannot represent every posterior, but it is fast to evaluate and sample, and Section 3.3 describes how importance sampling can correct it when needed.

Because shared weights preserve the association between features and coordinates throughout the network, the resulting Gaussian is equivariant to permutations of the latent coordinates: reordering the coordinates in the model description reorders the predicted mean and the rows and columns of the precision matrix in the same way (Section B).

### 3.3 Inference

Given a new model $m = (D, N, t_{0:N}, \theta_{0:N})$ and its observations $y$, one forward pass of AFIN computes $\phi = f_w(y, m) = (\mu, \Lambda)$, the mean and precision of $q_\phi(z) = \mathcal{N}(z; \mu, \Lambda^{-1})$, which can be used directly as the final posterior approximation without any per-problem optimization.

Maximizing Equation (2) minimizes expected forward KL, which penalizes low density where the posterior has mass. Standard ELBO-based VI instead minimizes reverse KL, so the two objectives can produce different approximations even within a Gaussian family (Blei et al., 2017). This mass-covering tendency motivates using $q_\phi$ as an importance-sampling proposal (Jerfel et al., 2021).

When further refinement is useful, we apply self-normalized importance sampling (SNIS) to samples from $q_\phi$. In our experiments, this can improve accuracy at modest additional test-time cost, requiring only sampling and density evaluations, with no further optimization. Each sample is weighted by the prior times the likelihoods divided by the proposal density, and the normalized weights give posterior estimates. We report direct predictions as *AFIN* and the refinement as *AFIN+SNIS*.

## 4 Parameter Sharing Across Dimensions

The modules of Section 3.2 must accept any latent dimension $D$ with one fixed set of weights. We achieve this by treating the latent coordinates like the elements of a set: the same computation is applied at every coordinate, and coordinates communicate only through averages, so no weight shape depends on $D$, and the module is equivariant to permutations of the coordinates.

**Parameter-tied MLPs.** For a map from $x \in \mathbb{R}^D$ to $o \in \mathbb{R}^D$, instead of $D$-dependent weight matrices, we compute, with $x_i$ the $i$-th entry of $x$ and $j = 1, \dots, H$ indexing hidden units,

$$h_{ij} = \rho\big(b_j + w_j x_i + \bar{w}_j \,\mathrm{mean}(x)\big), \qquad o_i = c + \sum_{j=1}^{H} v_j h_{ij}, \tag{7}$$

where $\rho$ is an activation function, $\mathrm{mean}(x)$ averages the $D$ entries, and the learned $b, w, \bar{w}, v \in \mathbb{R}^H$ and $c \in \mathbb{R}$ are shared by every coordinate. The local term lets each output depend on its own input and the pooled term on all the others; this is the permutation-equivariant layer of Deep Sets (Zaheer et al., 2017), with mean pooling. AFIN applies the same principle to the $D \times C$ node array and the $D \times D \times C$ pair array of a factor: one MLP acts at every node and another at every pair, and each receives, besides its local features, a few $C$-dimensional averages of the rest of the factor, such as the average of its row of the pair array or the node features of its two coordinates. These averages are a subset of the linear permutation-equivariant maps between vectors and matrices characterized by Maron et al. (2019); Section B lists the exact inputs.

**Transformer blocks.** Attention exchanges information across factors, treating each factor's node and pair arrays as one token, in a standard pre-normalization Transformer block (Vaswani et al., 2017) whose query, key, value, output, and feed-forward maps are the parameter-tied MLPs above. Attention scores average the query–key inner products over coordinates and coordinate pairs, and the resulting weights average the value tokens coordinate by coordinate, so factors mix while the coordinate structure inside each factor is untouched; without positional encodings the block is also equivariant to factor order. Computation is asymptotically quadratic in $D$ and in $N$, and memory quadratic in $D$ but linear in $N$, while the learned parameters depend on neither (Section B).

Table 1: Which aspects of a model can change at test time without retraining: observations $y$, their number $N$, latent dimension $D$, factor parameters $\theta$, and factor types $t$ (✓ supported, ○ partial, - unsupported); the last column marks methods whose authors report NUTS-comparable accuracy (with importance sampling for AFIN) and is not a controlled comparison.

| Method | $y$ | $N$ | $D$ | $\theta$ | $t$ | $\approx$ NUTS |
|---|---|---|---|---|---|---|
| Bayesian ICL (Reuter et al., 2025) | ✓ | ○ | - | - | - | ✓ |
| Masked Language Inference (Wu and Goodman, 2022) | ✓ | ✓ | ✓ | ✓ | ✓ | - |
| Masked Language Inference-Fit (Wu and Goodman, 2022) | ✓ | - | - | - | - | ✓ |
| Simformer (Gloeckler et al., 2024) | ✓ | ○ | - | - | - | ✓ |
| Distribution Transformers (Whittle et al., 2026) | ✓ | ✓ | - | ○ | - | ✓ |
| NPE-PFN (Vetter et al., 2025) | ✓ | ○ | ○ | - | - | ✓ |
| AFINs (ours) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## 5 Related Work

Bayesian posterior inference is usually run per instance, by MCMC such as HMC and NUTS (Neal, 2011; Hoffman and Gelman, 2014; Carpenter et al., 2017) or by variational inference (Blei et al., 2017; Hoffman et al., 2013; Ranganath et al., 2014; Rezende and Mohamed, 2015). Simulation-based inference learns from simulated pairs instead, from ABC to neural posterior, likelihood, and ratio estimators and globally amortized estimators such as BayesFlow and JANA (Beaumont et al., 2002; Cranmer et al., 2020; Papamakarios and Murray, 2016; Greenberg et al., 2019; Papamakarios et al., 2019; Hermans et al., 2020; Durkan et al., 2020; Tejero-Cantero et al., 2020; Lueckmann et al., 2021; Radev et al., 2022; 2023), but it amortizes over datasets from one simulator. Table 1 compares methods that widen the scope of one trained network by what can change at test time without retraining: a variable set of observations within a task family (Reuter et al., 2025; Gloeckler et al., 2024; Vetter et al., 2025), additionally the prior (Whittle et al., 2026), or arbitrary probabilistic-program text for Masked Language Inference (Wu and Goodman, 2022), whose NUTS-level accuracy however requires per-task fitting. Section F discusses further related interfaces (Müller et al., 2022; Chang et al., 2025; Elsemüller et al., 2024; Le et al., 2017; Wu et al., 2020); AFIN differs in its input, an explicit list of typed factors with their parameters, which lets one network cover every column of Table 1, including NUTS-comparable accuracy, without per-task fitting. Of these, only NPE-PFN accepts our tasks without retraining, and Section 6.5 compares with it.

Architecturally, the parameter-tied MLP of Section 4 is the Deep Sets equivariant layer (Zaheer et al., 2017), its node and pair summaries are a subset of the equivariant maps of Maron et al. (2019), and attention over a set without positional encodings is the Set Transformer (Lee et al., 2019), extended to several sets by Selby et al. (2022).

## 6 Experiments

All results use one trained AFIN; only the new-likelihood study trains further. Sections C to E give the setup and further results.

### 6.1 Setup

**Training.** We train one 4.55M-parameter AFIN with a Gaussian decoder on freshly simulated tasks with $D \in \{1, \dots, 16\}$ and $N \in \{1, \dots, 256\}$, sampled with extra weight on large $D$ and small $N$. Prior types are `gaussian prior`, `student-t prior`, and `laplace prior`; likelihood types are `gaussian likelihood`, `student-t likelihood`, `bernoulli-logit likelihood`, and `binomial-logit likelihood`, and half of the models with more than one observation mix several of them. We minimize the dimension-normalized loss

$$\mathcal{L}(w) = \mathbb{E}_{p(m,z,y)}\left[-\tfrac{1}{D}\log q(z; f_w(y, m))\right]. \tag{8}$$

The log density of a $D$-dimensional Gaussian is a sum over $D$ coordinates, so without the $1/D$ the highest-dimensional tasks would dominate the loss; dividing by $D$ puts every task on a per-coordinate scale. Since the simulated $z$ is an exact posterior draw given $(m, y)$, this loss minimizes the expected

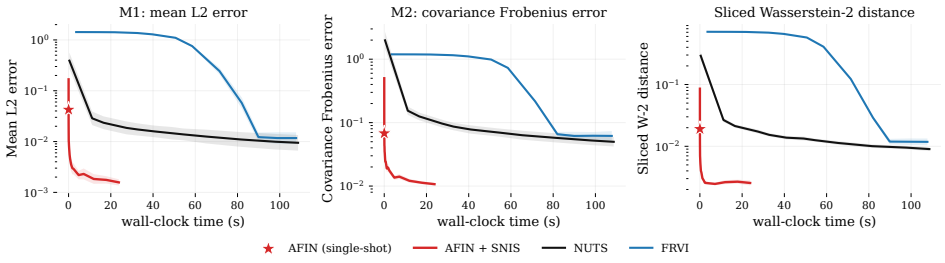



Figure 2: **One forward pass of AFIN is comparable to full-rank VI at a small fraction of the cost, and importance sampling greatly improves its accuracy.** Posterior error against inference time on the synthetic suite, averaged over 16 prior–likelihood combinations and three sizes. Points along each curve increase that method's sampling or optimization budget.

forward KL divergence weighted by $1/D$, with no reference posteriors needed. We train for $100{,}000$ AdamW steps (Loshchilov and Hutter, 2019) (batch size 32, four-step gradient accumulation, cosine decay from $2 \times 10^{-4}$), under 24 hours on one H100 GPU, and evaluate exponential-moving-average weights.

**Baselines, references, and metrics.** We compare with the No-U-Turn Sampler (NUTS; Hoffman and Gelman, 2014), full-rank Gaussian variational inference (FRVI), and on real data flow-based variational inference (IAF, MAF, and NSF; Kingma et al., 2016; Papamakarios et al., 2017; Durkan et al., 2019), each at seven learning rates, keeping the best at each budget. AFIN and all variational baselines run in PyTorch, so the timed NUTS is Pyro's; NumPyro's NUTS is far faster once compiled but pays a fixed JAX compilation cost per model, so it is reported separately in Section D.1, where AFIN stays well ahead of it. We report posterior mean error (M1), covariance error (M2), and sliced Wasserstein-2 distance ($\mathrm{SW}_2$) to long NUTS references; lower is better. Times are per-task sampling or optimization on an A100 GPU, excluding AFIN's pretraining; all results use three seeds with one-standard-deviation bands.

### 6.2 Accuracy and cost across model families

**Synthetic models.** The first question is whether one network is accurate across the family it was trained on, and at what cost. The suite crosses the four prior configurations with the four likelihood types, all with covariates, at an easy $(D = 4, N = 256)$, a medium $(D = 8, N = 64)$, and a hard $(D = 16, N = 1)$ size, for 48 settings. In Figure 2, one AFIN forward pass, a few hundredths of a second, matches converged FRVI on covariance error and is within a few times of it on the mean; longer NUTS and FRVI runs improve on it, and AFIN+SNIS closes much of the gap without optimization. The direct prediction is also nearly calibrated: its 90% intervals cover $0.875$ of the true coordinates in distribution.

**Real data.** The second question is whether it transfers to real datasets whose sizes and likelihood mixes it never saw. The UCI benchmark has 12 Bayesian regression and classification tasks (Table 11); two mix a continuous and a binary likelihood in one model, and six have $N > 256$, beyond the training range. In Figure 3, the direct prediction is within an order of magnitude of the converged baselines on most tasks and further off on Seeds Binary, Machine CPU, and Mesquite; AFIN+SNIS reaches a lower $\mathrm{SW}_2$ than every baseline at the budgets shown on all 12 tasks, within a few seconds.

### 6.3 Importance sampling refinement

Forward-KL training should make AFIN a mass-covering proposal, so importance sampling should gain more from it than from reverse-KL FRVI. Figure 4 tests this, with the cost of building each proposal included in its time: AFIN+SNIS dominates at small budgets, where FRVI has not yet produced a usable proposal, and ends with a lower mean error than FRVI+SNIS at the same number of proposals even though its proposal starts worse.

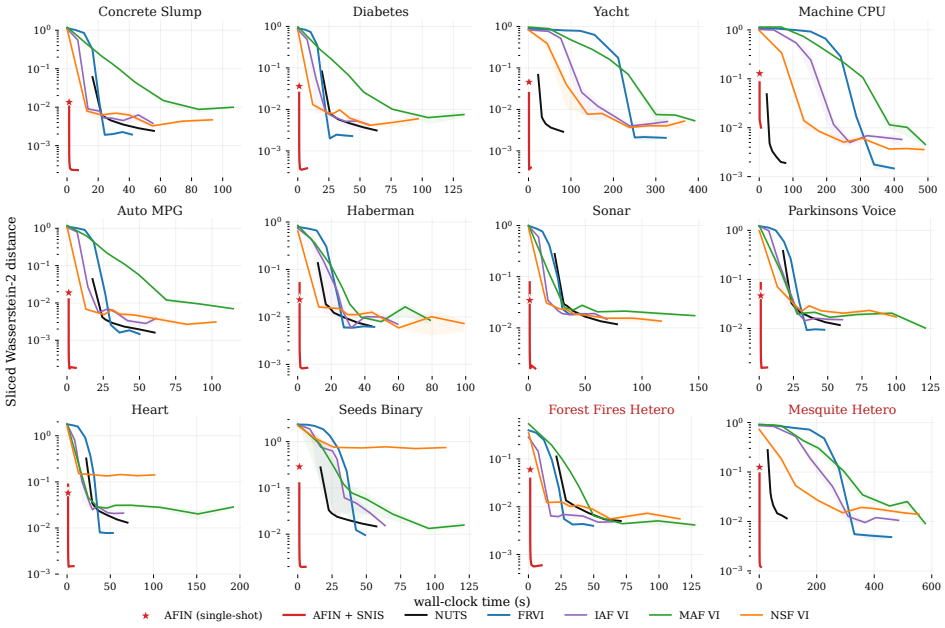



Figure 3: **On real data, AFIN with importance sampling reaches lower error than every baseline at the budgets shown, on all 12 tasks, within seconds.** Sliced Wasserstein-2 distance to a long-run NUTS reference against inference time on 12 UCI tasks. Red titles mark models that mix two likelihood types; six tasks have $N > 256$, beyond the training range.

### 6.4 Beyond the training distribution

**A new likelihood type.** Each factor type enters through its own adapter, so a type the network has never seen can be added by training that adapter alone. We test this with a Poisson likelihood with a log link, unseen in pretraining, training either a new adapter with the network frozen or adapter and network jointly, and evaluate on 24 held-out Poisson tasks (Table 2). Full fine-tuning is more accurate directly, while adapter-only training costs about 60% of the time and leaves every original factor type untouched; after SNIS the two are indistinguishable. A new likelihood therefore costs one adapter and about five GPU-hours.

Table 2: Adding a Poisson likelihood: errors over 24 tasks and three seeds, direct and after SNIS (4,096 proposals); best in bold.

| Training mode | M1 | M2 | $SW_2$ | GPU-h |
|---|---|---|---|---|
| Adapter only | 0.117 | 0.156 | 0.0591 | **4.95** |
| +SNIS | **0.0319** | **0.113** | **0.0301** | |
| Full fine-tuning | 0.0785 | 0.140 | 0.0458 | 8.39 |
| +SNIS | 0.0331 | 0.115 | 0.0305 | |

**Larger models and datasets.** Does it hold up beyond the sizes it trained on? Table 3 evaluates the same checkpoint on six configurations with $D \leq 32$ and $N \leq 512$, grouped by whether $D$, $N$, or both exceed the training ranges of 16 and 256. When only $N$ grows, or both grow, AFIN+SNIS has the lowest mean error on all three metrics, within noise of the best baseline on several, in under half a second against 32–157 s for FRVI and NUTS. Growing only $D$ is harder: the direct prediction degrades, SNIS lowers the mean and $SW_2$ errors but raises the covariance error, and NUTS is most accurate at far larger cost. Coverage follows the same boundary, 0.91 at $N \leq 400$ and 0.57 at $N \leq 1{,}024$.

Table 3: Extrapolation in $D$ and $N$: $SW_2$ (top), time per task (bottom); best in bold.

| Split | AFIN | AFIN+SNIS | NUTS | FRVI |
|---|---|---|---|---|
| OOD $D$ | 0.031 | 0.024 | **0.012** | 0.042 |
| | 0.02 s | 0.24 s | 308 s | 27 s |
| OOD $N$ | 0.027 | **0.0013** | 0.0025 | 0.0025 |
| | 0.02 s | 0.25 s | 95 s | 32 s |
| OOD $D, N$ | 0.040 | **0.0039** | 0.0052 | 0.0041 |
| | 0.10 s | 0.45 s | 157 s | 100 s |

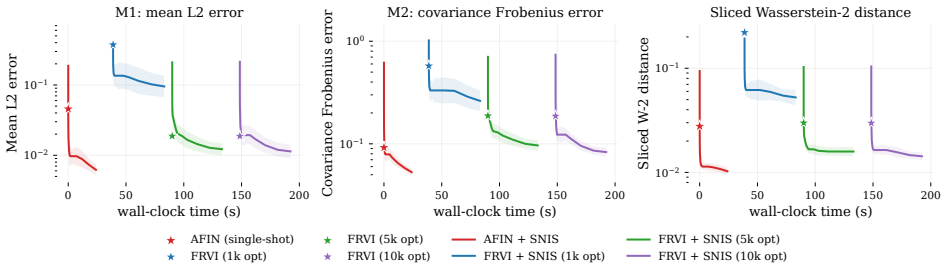



Figure 4: **AFIN gains more from importance sampling than full-rank VI.** SNIS refinement on the 48 synthetic settings; stars mark the proposals before correction (FRVI after 1k, 5k, or 10k steps) and curves add proposal samples.

### 6.5 Amortized baseline

An in-context learner fed a task's simulations also needs no training; does it match AFIN? NPE-PFN (Vetter et al., 2025) does this with a frozen tabular foundation model (Hollmann et al., 2025). We ran it on the 144 synthetic tasks with 1k or 10k context simulations and 2,048 samples (Table 4). AFIN is more accurate on every metric in under a second; NPE-PFN takes 13 s, or 115 s with the larger context, which halves its mean error but not to AFIN's.

Table 4: NPE-PFN with 1k or 10k context simulations and AFIN on the 144 synthetic tasks, with mean time per task; best in bold.

| Method | M1 | M2 | $SW_2$ | s/task |
|---|---|---|---|---|
| AFIN | **0.053** | **0.105** | **0.041** | **0.95** |
| NPE-PFN 1k | 0.242 | 0.301 | 0.113 | 13.2 |
| NPE-PFN 10k | 0.130 | 0.254 | 0.076 | 115 |

### 6.6 Ablations

Do attention across factors and the pair array earn their place? Table 5 removes each: attention is replaced by a plain sum of the encoded factors, the conjugate template of Section 3.2 with a learned encoder and decoder, and the pair array is dropped, leaving node features only. Both matter, and their effects compound: dropping the pair array doubles the covariance error, since the decoder must then build the precision from node features alone, dropping attention triples $SW_2$ and raises the mean error fivefold, and dropping both is worst on every metric. The variants are smaller, so the gaps bound module and capacity together.

Table 5: Ablations on the synthetic suite: direct-prediction errors of variants trained with the same simulator and schedule; the first row is AFIN; best in bold.

| Embedding | Merge | M1 | M2 | $SW_2$ |
|---|---|---|---|---|
| Node+pair | Attention | **0.053** | **0.105** | **0.041** |
| Node only | Attention | 0.073 | 0.208 | 0.049 |
| Node+pair | Sum | 0.290 | 0.248 | 0.132 |
| Node only | Sum | 0.314 | 0.276 | 0.142 |

## 7 Discussion

AFIN describes a model as a list of typed factors and turns one simulation-trained network into a posterior approximator for every model the list can express, with weights that depend on neither $D$ nor $N$. One trained network gives a usable posterior in one forward pass across priors, likelihoods, and sizes, matches much longer NUTS and variational runs after importance sampling at a fraction of their cost, transfers to real data, gains a new likelihood through a cheap adapter, and beats an in-context foundation-model baseline. Three limitations remain, each a direction for future work: the scope is one global latent vector with conditionally independent observations and known factor parameters, and could extend to hierarchical models and local latent variables; the decoder is Gaussian, which limits the posteriors it can express, and a mixture or flow decoder would lift this while keeping a density for SNIS; and cost grows quadratically with $D$ and $N$, with extrapolation in $D$ degrading accuracy first, which training at larger sizes would address.

### AI USE STATEMENT

We used generative AI tools (large language models) to assist with editing this paper, including identifying errors and correcting grammar and typos; with writing and debugging parts of the experiment and analysis code; and with cross-checking details in the paper, such as the exact numbers reported in the Appendix, against the codebase. We did not use generative AI tools for research ideation, for designing the method or experiments, or for generating, analyzing, or interpreting experimental results. All AI-assisted text was rewritten or checked by the authors, and all AI-assisted code was reviewed and tested by the authors. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

The architecture is specified in Sections 3.2 and 4 and, module by module with parameter counts, in Section B. The training simulator, including every sampling distribution, is given in Section C, and the optimization settings in Section 6.1. Section D gives the baseline tuning protocol, the reference posteriors with their convergence diagnostics, the metric definitions, and the exact task lists for the synthetic, UCI, and extrapolation benchmarks; Section E gives the protocol of the Poisson study. Anonymized source code is available at `https://anonymous.4open.science/r/ICLR_AFIN-F43C/`.

## REFERENCES


Abhinav Agrawal and Justin Domke. Disentangling impact of capacity, objective, batchsize, estimators, and step-size on flow vi. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2025.

Michael Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *Journal of Machine Learning Research*, 2025.

Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 2002.

Michael Betancourt. Diagnosing suboptimal cotangent disintegrations in Hamiltonian Monte Carlo. arXiv:1604.00695, 2016.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 2017.

Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and radon Wasserstein barycenters of measures. *Journal of Mathematical Imaging and Vision*, 51(1):22–45, 2015. doi: 10.1007/s10851-014-0506-3.

Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 2017.

Paul Edmund Chang, Nasrulloh Ratu Bagus Satrio Loka, Daolang Huang, Ulpu Remes, Samuel Kaski, and Luigi Acerbi. Amortized probabilistic conditioning for optimization, simulation and inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2025.

Małgorzata Charytanowicz, Jerzy Niewczas, Piotr Kulczycki, Piotr Kowalski, and Szymon Łukasik. Seeds. UCI Machine Learning Repository, 2010.

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *International Conference on Learning Representations (ICLR)*, 2023.

Paulo Cortez and Aníbal Morais. Forest Fires. UCI Machine Learning Repository, 2007.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 2020.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations (ICLR)*, 2017.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *Neural Information Processing Systems (NeurIPS)*, 2019.

Conor Durkan, Iain Murray, and George Papamakarios. On contrastive learning for likelihood-free inference. In *International Conference on Machine Learning (ICML)*, 2020.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 2004.

Lasse Elsemüller, Hans Olischläger, Marvin Schmitt, Paul-Christian Bürkner, Ullrich Köthe, and Stefan T. Radev. Sensitivity-aware amortized bayesian inference. *Transactions on Machine Learning Research*, 2024.

Jacob Feldmesser. Computer Hardware. UCI Machine Learning Repository, 1987.

Tomas Geffner, George Papamakarios, and Andriy Mnih. Compositional score modeling for simulation-based inference. In *International Conference on Machine Learning (ICML)*, 2023.

Andrew Gelman and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Analytical Methods for Social Research. Cambridge University Press, Cambridge, 2007.

J. Gerritsma, R. Onnink, and A. Versluis. Yacht Hydrodynamics. UCI Machine Learning Repository, 1981.

Manuel Gloeckler, Michael Deistler, Christian D. Weilbach, Frank Wood, and Jakob H. Macke. All-in-one simulation-based inference. In *International Conference on Machine Learning (ICML)*, 2024.

David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning (ICML)*, 2019.

S. Haberman. Haberman's Survival. UCI Machine Learning Repository, 1976.

Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free MCMC with amortized approximate ratio estimators. In *International Conference on Machine Learning (ICML)*, 2020.

Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 2014.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025.

Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. Heart Disease. UCI Machine Learning Repository, 1989.

Ghassen Jerfel, Serena Wang, Clara Wong-Fannjiang, Katherine A. Heller, Yian Ma, and Michael I. Jordan. Variational refinement for importance sampling using the forward kullback-leibler divergence. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Joohwan Ko and Justin Domke. Model-informed flows for bayesian inference. In *Neural Information Processing Systems (NeurIPS)*, 2025.

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, pages 3744–3753. PMLR, 2019.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *International Conference on Learning Representations (ICLR)*, 2023.

Max Little. Parkinsons. UCI Machine Learning Repository, 2007.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Uncertainty in Artificial Intelligence (UAI)*, pages 362–369, 2001.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations (ICLR)*, 2022.

Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2011.

George Papamakarios and Iain Murray. Fast $\epsilon$-free inference of simulation models with bayesian conditional density estimation. In *Neural Information Processing Systems (NeurIPS)*, 2016.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Neural Information Processing Systems (NeurIPS)*, 2017.

George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 2021.

R. Quinlan. Auto MPG. UCI Machine Learning Repository, 1993.

Stefan T. Radev, Ulf K. Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Stefan T. Radev, Marvin Schmitt, Valentin Pratz, Umberto Picchini, Ullrich Köthe, and Paul-Christian Bürkner. JANA: Jointly amortized neural approximation of complex bayesian models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2023.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

Arik Reuter, Tim G. J. Rudner, Vincent Fortuin, and David Rügamer. Can transformers learn full bayesian inference in context? In *International Conference on Machine Learning (ICML)*, 2025.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, 2015.

Terry Sejnowski and R. Gorman. Connectionist Bench (Sonar, Mines vs. Rocks). UCI Machine Learning Repository, 1988.

Kira A Selby, Ahmad Rashid, Ivan Kobyzev, Mehdi Rezagholizadeh, and Pascal Poupart. Learning functions on multiple sets using multi-set transformers. In *Uncertainty in Artificial Intelligence (UAI)*, pages 1760–1770. PMLR, 2022.

Louis Sharrock, Jack Simons, Song Liu, and Mark Beaumont. Sequential neural score estimation: Likelihood-free inference with conditional score based diffusion models. *International Conference on Machine Learning (ICML)*, 2024.

Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 2020.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Neural Information Processing Systems (NeurIPS)*, 2017.

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian Analysis*, 16(2):667–718, 2021. doi: 10.1214/20-BA1221.

Aki Vehtari, Daniel Simpson, Andrew Gelman, Yuling Yao, and Jonah Gabry. Pareto smoothed importance sampling. *Journal of Machine Learning Research*, 25(72):1–58, 2024.

Julius Vetter, Manuel Gloeckler, Daniel Gedon, and Jakob H. Macke. Effortless, simulation-efficient bayesian inference using tabular foundation models. In *Neural Information Processing Systems (NeurIPS)*, 2025.

George Whittle, Juliusz Ziomek, Jacob Rawling, and Michael A. Osborne. Distribution transformers: Fast approximate bayesian inference with on-the-fly prior adaptation, 2026.

Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. *Neural Information Processing Systems (NeurIPS)*, 2023.

Mike Wu and Noah Goodman. Foundation posteriors for approximate probabilistic inference. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Mike Wu, Kristy Choi, Noah D. Goodman, and Stefano Ermon. Meta-amortized variational inference and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

I-Cheng Yeh. Concrete Slump Test. UCI Machine Learning Repository, 2007.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

# A NOTATION

Table 6: Main notation. The prior is factor $n = 0$ and the likelihoods are factors $n = 1, \ldots, N$.

| Symbol | Meaning |
|---|---|
| *Model* | |
| $z \in \mathbb{R}^D$ | Latent vector; $D$ is the latent dimension. |
| $y = (y_1, \ldots, y_N)$ | Observations, one likelihood factor each. |
| $m = (D, N, t_{0:N}, \theta_{0:N})$ | Model description: sizes, factor types, and factor parameters. |
| $t_n, \theta_n$ | Type label and known numerical parameters of factor $n$. |
| $p(z \mid t_0, \theta_0)$, $p(y_n \mid z, t_n, \theta_n)$ | Prior and likelihood factors. |
| *Inference network* | |
| $f_w$, $\phi = f_w(y, m)$ | AFIN with learned weights $w$, and the posterior parameters it predicts. |
| $q_\phi(z) = \mathcal{N}(z; \mu, \Lambda^{-1})$ | Gaussian posterior approximation with mean $\mu$ and precision $\Lambda$. |
| $\mathcal{L}(w)$ | Training loss, Equation (8). |
| $\texttt{adapter}_t$, $\texttt{net}$ | Type-specific adapter and shared network that form the `encoder`. |
| $E_n = (E_n^{\text{node}}, E_n^{\text{pair}})$ | Embedding of factor $n$: a $D \times C$ node array and a $D \times D \times C$ pair array. |
| $C$ | Number of features per coordinate or pair. |
| $\texttt{transformer}_m$, $M$ | $m$-th merge block and the number of blocks; $E_n^{(m)}$ is the embedding after block $m$. |
| $\bar{E} = \sum_n E_n^{(M)}$ | Merged embedding passed to the `decoder`. |
| *Modules* | |
| $H$ | Hidden width of the coordinate-wise MLPs. |
| $b_Q, b_K, b_V, b_{\text{out}}, b_{\text{FFN}}$ | Node-pair maps inside a merge block. |
| $Q_n, K_n, V_n, \widetilde{V}_n$ | Query, key, value, and attended value of factor $n$. |
| h, $u$ | Number of attention heads and per-head width. |
| sym, LN | Symmetrization of a pair array; layer normalization over features. |
| *Evaluation* | |
| M1, M2, $\text{SW}_2$ | Posterior mean error, covariance error, and sliced Wasserstein-2 distance, Section D.3. |
| SNIS | Self-normalized importance sampling with $q_\phi$ as the proposal. |

# B ARCHITECTURE DETAILS

This appendix specifies the modules of Sections 3.2 and 4. Every learned map is a small MLP applied with shared weights at each latent coordinate or coordinate pair, so no parameter shape depends on $D$ or $N$.

**Node-pair embeddings.** A factor embedding is a pair of arrays $E = (E^{\text{node}}, E^{\text{pair}})$ with $E^{\text{node}} \in \mathbb{R}^{D \times C}$ and $E^{\text{pair}} \in \mathbb{R}^{D \times D \times C}$. We write $E_i^{\text{node}} \in \mathbb{R}^C$ for the features of coordinate $i$ and $E_{ij}^{\text{pair}} \in \mathbb{R}^C$ for those of the pair $(i, j)$. Pair arrays are kept symmetric by $\text{sym}(X)_{ij} = \frac{1}{2}(X_{ij} + X_{ji})$ after every update.

**Coordinate-wise MLPs.** A coordinate-wise MLP $g : \mathbb{R}^{C_{\text{in}}} \to \mathbb{R}^{C_{\text{out}}}$ is applied independently at every coordinate of a node array, $[g(X)]_i = g(X_i)$, or at every pair of a pair array, $[g(X)]_{ij} = g(X_{ij})$, with the same weights. Unless stated otherwise, these MLPs have two hidden layers of width $H = 192$ with SiLU activations, and every embedding has $C = 40$ features.

**Summaries.** The parameter-tied maps combine local features with the following averages of a node-pair embedding, each in $\mathbb{R}^C$,

$$\begin{aligned} \mathrm{row}(E^{\mathrm{pair}})_i &= \tfrac{1}{D}\textstyle\sum_j E^{\mathrm{pair}}_{ij}, \qquad \mathrm{col}(E^{\mathrm{pair}})_j = \tfrac{1}{D}\textstyle\sum_i E^{\mathrm{pair}}_{ij}, \quad \mathrm{diag}(E^{\mathrm{pair}})_i = E^{\mathrm{pair}}_{ii}, \\ \overline{E}^{\mathrm{pair}} &= \tfrac{1}{D^2}\textstyle\sum_{i,j} E^{\mathrm{pair}}_{ij}, \qquad \overline{E}^{\mathrm{node}} = \tfrac{1}{D}\textstyle\sum_i E^{\mathrm{node}}_i. \end{aligned} \tag{9}$$

For a symmetric pair array the row and column averages coincide; we keep both because the same maps are also applied to intermediate arrays that need not be symmetric.

**Node-pair maps.** A node-pair map takes a node-pair embedding and returns one. It uses two coordinate-wise MLPs $g^{\mathrm{node}}, g^{\mathrm{pair}} : \mathbb{R}^{6C} \to \mathbb{R}^C$ whose inputs are

$$\begin{aligned} \zeta^{\mathrm{node}}_i &= \big[E^{\mathrm{node}}_i,\ \mathrm{diag}(E^{\mathrm{pair}})_i,\ \mathrm{row}(E^{\mathrm{pair}})_i,\ \mathrm{col}(E^{\mathrm{pair}})_i,\ \overline{E}^{\mathrm{pair}},\ \overline{E}^{\mathrm{node}}\big], \\ \zeta^{\mathrm{pair}}_{ij} &= \big[E^{\mathrm{pair}}_{ij},\ \mathrm{row}(E^{\mathrm{pair}})_i,\ \mathrm{col}(E^{\mathrm{pair}})_j,\ E^{\mathrm{node}}_i,\ E^{\mathrm{node}}_j,\ \overline{E}^{\mathrm{pair}}\big], \end{aligned} \tag{10}$$

and returns $\big(g^{\mathrm{node}}(\zeta^{\mathrm{node}}),\ \mathrm{sym}(g^{\mathrm{pair}}(\zeta^{\mathrm{pair}}))\big)$. A *node-pair layer* adds this output to its input as a residual update and symmetrizes the pair array again. Every average is invariant to permutations of the coordinates and every local term moves with its coordinate, so both constructions are permutation-equivariant.

### B.1 Encoder

**Adapters.** The adapter $\texttt{adapter}_t$ turns the numerical inputs of a factor of type $t$ into an initial node-pair embedding. It builds a descriptor vector for every coordinate, $\xi^{\mathrm{node}}_i \in \mathbb{R}^{\chi^{\mathrm{node}}}$, and for every coordinate pair, $\xi^{\mathrm{pair}}_{ij} \in \mathbb{R}^{\chi^{\mathrm{pair}}}$, whose lengths depend on the type but not on $D$. The descriptors pass through $\mathrm{symlog}(x) = \mathrm{sign}(x)\log(1+|x|)$ and then through two coordinate-wise MLPs with one hidden layer of width 64 that output $C$ features; the pair output is symmetrized. Factor types with different input forms use different adapters. A Gaussian prior with diagonal or full covariance, and a Gaussian likelihood with or without covariates, count as separate forms, giving four prior and five likelihood adapters in our experiments, with 55,376 parameters in total. For example, a diagonal Gaussian prior $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ uses

$$\xi^{\mathrm{node}}_i = [\mu_i,\ \log\sigma_i,\ \mu_i/\sigma_i], \qquad \xi^{\mathrm{pair}}_{ij} = [\mu_i,\ \log\sigma_i,\ \mu_i/\sigma_i,\ \mu_j,\ \log\sigma_j,\ \mu_j/\sigma_j,\ \mathbf{1}\{i=j\}], \tag{11}$$

so $\chi^{\mathrm{node}} = 3$ and $\chi^{\mathrm{pair}} = 7$. Likelihood adapters receive the observation and, where present, the covariate row; Section E.6 lists the descriptors of the Poisson adapter as a second example.

**Shared network.** The shared network $\texttt{net}$ is a stack of four node-pair layers with the inputs of Equation (10), applied to every factor with the same weights. Its output is the initial embedding $E^{(0)}_n$ of Section 3.2.

### B.2 Merge

The merge applies $M = 4$ Transformer blocks to the factor sequence $E^{(0)}_{0:N}$ and sums the result over factors. Inside a block, $b_Q, b_K, b_V, b_{\mathrm{out}}, b_{\mathrm{FFN}}$ are node-pair maps, and LN is layer normalization over the $C$ features at each node or pair. Table 7 places the block side by side with a vanilla Transformer layer.

**Attention.** Block $m$ forms queries, keys, and values from the normalized input, $Q_{0:N} = b_Q(\mathrm{LN}(E^{(m-1)}_{0:N}))$ and likewise $K_{0:N}$ and $V_{0:N}$. Each is a node-pair object with $C$ features, split into $\mathsf{h} = 8$ heads of width $u = 5$; the coordinate axes are never split. Within a head, the score between factors $n$ and $\ell$ averages inner products over coordinates and over coordinate pairs,

$$S^{\mathrm{node}}_{n\ell} = \frac{1}{D}\sum_i \langle Q^{\mathrm{node}}_{n,i}, K^{\mathrm{node}}_{\ell,i}\rangle, \qquad S^{\mathrm{pair}}_{n\ell} = \frac{1}{D^2}\sum_{i,j}\langle Q^{\mathrm{pair}}_{n,ij}, K^{\mathrm{pair}}_{\ell,ij}\rangle, \tag{12}$$

and the attention weights are

$$A_{n\ell} = \mathrm{softmax}_\ell\Big(\frac{\lambda_{\mathrm{node}} S^{\mathrm{node}}_{n\ell} + \lambda_{\mathrm{pair}} S^{\mathrm{pair}}_{n\ell}}{\sqrt{u}}\Big), \qquad n, \ell = 0, \dots, N, \tag{13}$$

with learned mixing weights $\lambda_{\text{node}} + \lambda_{\text{pair}} = 1$ per block. The same weights average the node and pair values of all factors, coordinate by coordinate,

$$\widetilde{V}^{\text{node}}_{n,i} = \sum_{\ell=0}^{N} A_{n\ell} V^{\text{node}}_{\ell,i}, \qquad \widetilde{V}^{\text{pair}}_{n,ij} = \sum_{\ell=0}^{N} A_{n\ell} V^{\text{pair}}_{\ell,ij}, \tag{14}$$

and the heads are concatenated along the feature axis.

**Residual updates.** The attention output and the feed-forward update are added as residuals, with the pair array symmetrized each time,

$$\begin{aligned} \widetilde{E}_n &= \operatorname{sym}\big(E_n^{(m-1)} + b_{\text{out}}(\widetilde{V}_n)\big), \\ E_n^{(m)} &= \operatorname{sym}\big(\widetilde{E}_n + b_{\text{FFN}}(\text{LN}(\widetilde{E}_n))\big), \end{aligned} \tag{15}$$

where sym acts on the pair component only. No positional encodings are used, so the block is equivariant to the order of the factors.

**Pooling.** After the last block, $\bar{E} = \sum_{n=0}^{N} E_n^{(M)}$, symmetrized once more. Pooling has no parameters, so the merge accepts any $N$.

Table 7: Vanilla Transformer versus AFIN merge block for a sequence of length $N+1$. Both compute $A \in \mathbb{R}^{(N+1)\times(N+1)}$, but only the vanilla layer uses learned maps whose shapes involve $DC$. The table shows the single-stream notation; the AFIN implementation uses the multi-head factor-axis version described in Section B.2, and $U$ denotes the attention width, equal to $C$ in AFIN.

| | Vanilla Transformer | AFIN merge block |
|---|---|---|
| Token | $E_n^{\text{flat}} = \text{vec}(E_n) \in \mathbb{R}^{DC}$ | $E_n \in \mathbb{R}^{D\times C}$ |
| QKV | $Q_n = W_Q\,\text{LN}(E_n^{\text{flat}}),\ K_n = W_K\,\text{LN}(E_n^{\text{flat}}),$ $V_n = W_V\,\text{LN}(E_n^{\text{flat}}),\ Q_n, K_n, V_n \in \mathbb{R}^U$ | $Q_n = b_Q(\text{LN}(E_n)),\ K_n = b_K(\text{LN}(E_n)),$ $V_n = b_V(\text{LN}(E_n)),\ Q_n, K_n, V_n \in \mathbb{R}^{D\times U}$ |
| Score attention | $S_{n\ell} = U^{-1/2}\langle Q_n, K_\ell\rangle,$ $A_{n\ell} = \text{softmax}_\ell(S_{n\ell}),\ A \in \mathbb{R}^{(N+1)\times(N+1)}$ | $S_{n\ell} = (D\sqrt{U})^{-1}\sum_{i=1}^{D}\langle Q_{n,i}, K_{\ell,i}\rangle,$ $A_{n\ell} = \text{softmax}_\ell(S_{n\ell}),\ A \in \mathbb{R}^{(N+1)\times(N+1)}$ |
| Value output | $\widetilde{V}_n = \sum_{\ell=0}^{N} A_{n\ell} V_\ell \in \mathbb{R}^U,$ $\widetilde{E}_n^{\text{flat}} = E_n^{\text{flat}} + W_O \widetilde{V}_n \in \mathbb{R}^{DC}$ | $\widetilde{V}_{n,i} = \sum_{\ell=0}^{N} A_{n\ell} V_{\ell,i},\ \widetilde{V}_n \in \mathbb{R}^{D\times U},$ $\widetilde{E}_n = E_n + b_{\text{out}}(\widetilde{V}_n) \in \mathbb{R}^{D\times C}$ |
| FFN | $(E_n^{\text{flat}})^+ = \widetilde{E}_n^{\text{flat}} + \text{FFN}\left(\text{LN}(\widetilde{E}_n^{\text{flat}})\right)$ | $E_n^+ = \widetilde{E}_n + b_{\text{FFN}}\left(\text{LN}(\widetilde{E}_n)\right)$ |
| Maps | $W_Q, W_K, W_V \in \mathbb{R}^{U\times DC},\ W_O \in \mathbb{R}^{DC\times U}$ $\text{FFN} : \mathbb{R}^{DC} \to \mathbb{R}^{DC}$ | $b_Q, b_K, b_V : \mathbb{R}^{D\times C} \to \mathbb{R}^{D\times U}$ $b_{\text{out}} : \mathbb{R}^{D\times U} \to \mathbb{R}^{D\times C},\ b_{\text{FFN}} : \mathbb{R}^{D\times C} \to \mathbb{R}^{D\times C}$ |

### B.3 Decoder

Three coordinate-wise MLPs with two hidden layers of width 160 map $\bar{E}$ to the Gaussian parameters. The mean MLP takes a $5C$-dimensional input at each coordinate, namely the node features, the average of the pair row and column means, the diagonal pair features, and the global node and pair averages, and outputs $\mu_i$. The off-diagonal MLP maps each pair's features to a scalar; these are scaled by $0.1$, symmetrized, and placed in a matrix $\Omega$ with zero diagonal. The diagonal MLP maps each node's features to a scalar $r_i$. The precision is then

$$(\Lambda_\phi)_{ij} = \begin{cases} \Omega_{ij}, & i \neq j, \\ \sum_{k\neq i} |\Omega_{ik}| + \text{softplus}(r_i) + 10^{-4}, & i = j, \end{cases} \tag{16}$$

which is symmetric and strictly diagonally dominant with a positive diagonal, hence positive definite. Its Cholesky factor is used for density evaluation and sampling.

### B.4 Configuration and parameter count

The default configuration uses $C = 40$ features; coordinate-wise MLPs with two hidden layers of width 192 in the encoder and merge and of width 160 in the decoder; adapters with one hidden layer of width 64; four node-pair layers in the shared encoder; and $M = 4$ merge blocks with 8 heads. Table 8 gives the parameter count by module; none of the counts depends on $D$ or $N$.

Table 8: Module-wise parameter counts for the default AFIN configuration. Counts are independent of the latent dimension $D$ and the number of likelihood factors $N$.

| Module | # parameters |
|---|---|
| Factor-type adapters | 55,376 |
| Shared encoder (four node-pair layers) | 728,384 |
| Merge blocks ($M = 4$) | 3,643,208 |
| Gaussian decoder | 123,043 |
| Total trainable parameters | 4,550,011 |

### B.5 Computational cost

Let $K = N + 1$ be the number of factors. Every map is applied at $D$ nodes and $D^2$ pairs per factor, so the encoder and the five maps inside each merge block cost $O(KD^2(CH + H^2))$ operations and hold $O(KD^2C)$ activations per layer. Attention adds $O(K^2D^2C)$ for the scores in Equation (12) and the same again for averaging the values. The decoder costs $O(D^2(CH + H^2))$ plus $O(D^3)$ for the Cholesky factorization, and each SNIS proposal costs one Gaussian sample and one evaluation of the model's log density. The pair arrays therefore make the cost quadratic in $D$, and attention makes it quadratic in $N$; the learned parameters in Table 8 are unaffected, and Section E.5 reports measured times and memory.

## C Training Details

We train on tasks generated on the fly, so no task is ever reused. Each optimizer step uses four micro-batches of $B = 32$ tasks. Tasks in a micro-batch share $D$ and $N$ so that they can be batched; everything else is sampled independently for each task.

**Task sizes.** With probability 0.6 we draw $(D, N)$ from

$$\mathbb{P}(D, N) \propto \Big(\frac{D}{16}\Big)^{1.0}\Big(\frac{1}{N}\Big)^{0.75}, \qquad D \in \{1, \dots, 16\},\ N \in \{1, \dots, 256\}, \tag{17}$$

and otherwise uniformly from the same ranges. The bias puts more weight on high-dimensional, data-poor tasks, which are the hard ones under uniform sampling.

**Prior.** One of four configurations, chosen uniformly. The location is $\mu \sim 0.45\mathcal{N}(0, I_D)$ in all cases, and the remaining parameters are

- `gaussian prior`, diagonal covariance: $\log \sigma_i \sim \mathrm{Uniform}(-0.8, 0)$;
- `gaussian prior`, full covariance: precision $\Lambda = GG^\top/D + 0.5I_D$ with $G \sim 0.3\mathcal{N}(0, I_{D\times D})$;
- `student-t prior`: $\log \sigma_i \sim \mathrm{Uniform}(-0.7, 0)$ and $\nu \sim \mathrm{Uniform}(3, 8)$;
- `laplace prior`: $\log s_i \sim \mathrm{Uniform}(-1.0, -0.05)$.

We then draw $z$ from this prior.

**Likelihoods.** There are five likelihood forms: `gaussian likelihood` with or without covariates, `student-t likelihood` with covariates, `bernoulli-logit likelihood`, and `binomial-logit likelihood`. For $N = 1$ we pick one form uniformly. For $N > 1$, with probability 0.5 all observations share one uniformly chosen form; otherwise we draw the

number of distinct forms $k \in \{2, \dots, \min(5, N)\}$ with $\mathbb{P}(k) \propto e^{-(k-2)}$, pick $k$ forms without replacement, allocate observations to them with weights $\pi \sim \mathrm{Dirichlet}(0.5\,\mathbf{1}_k)$ and at least one observation per form, and permute the assignment. With covariates, $y_n = x_n^\top z + \varepsilon_n$ with Gaussian or Student-$t$ noise of sampled scale, $y_n \sim \mathrm{Bernoulli}(\sigma(x_n^\top z))$, or $y_n \sim \mathrm{Binomial}(n_c, \sigma(x_n^\top z))$ with $n_c \sim \mathrm{Uniform}\{2, \dots, 8\}$. Without covariates, $y_n = z + \varepsilon_n \in \mathbb{R}^D$ with isotropic Gaussian noise of sampled scale.

**Covariates.** The design matrix $X \in \mathbb{R}^{N \times D}$ has base scale $0.9/\sqrt{D}$, which keeps $Xz$ of order one. Its rows follow one of four distributions with probabilities $(0.7, 0.1, 0.1, 0.1)$: isotropic Gaussian; Gaussian with a sampled diagonal spectrum; Gaussian with a sampled spectrum and a random rotation; and the latter with row-wise multiplicative weights that give multivariate Student-$t$ tails.

**Optimization.** AdamW for $100{,}000$ steps with cosine learning-rate decay from $2 \times 10^{-4}$ and gradient accumulation over the four micro-batches; evaluation uses an exponential moving average of the weights. Training takes under 24 hours on one H100 GPU.

# D Evaluation Details

## D.1 Baselines and timing

**NUTS.** The timed NUTS is Pyro's implementation at its defaults (target acceptance $0.8$, maximum tree depth $10$, diagonal mass adaptation), initialized at the prior mean. Each point on a NUTS curve is an independent run whose time includes initialization, warmup, and sampling. We time Pyro rather than NumPyro for two reasons: AFIN and every variational baseline run in PyTorch, so all methods share one stack, and NumPyro compiles each model with JAX before sampling, a fixed per-model cost. Once compiled, NumPyro is much faster: on a 20-task check with identical settings (four chains, $1{,}000$ warmup steps, $2{,}000$ retained draws per chain, one GPU), it needs a median of $15$ s per task against Pyro's $413$ s, with the same effective sample sizes (minimum bulk ESS $4{,}326$ and $4{,}296$) and no divergences. With NumPyro the NUTS curves in Figures 2 and 3 would move left by roughly a factor of 25, which still leaves them far to the right of AFIN's forward pass ($0.012$ s) and its sub-second SNIS budgets.

**Variational baselines.** FRVI fits a full-rank Gaussian by stochastic gradient ascent on the ELBO, and IAF, MAF, and NSF VI fit flows the same way. Every curve runs seven learning rates spaced logarithmically from $10^{-6}$ to $10^{-2}$ and reports the best result at each budget. Our released FRVI parameterizes the covariance by its Cholesky factor, starts from the prior's mean and covariance, and uses Adam with 16 reparameterized samples per step and gradient clipping; the proposals in Figure 4 use learning rate $0.01$, decayed tenfold after $1{,}000$ and $5{,}000$ steps. Even with the learning rate chosen for every task and budget by the reference error, which favors FRVI, matching the mean and $\mathrm{SW}_2$ error of AFIN's direct prediction on the 96 easy and medium tasks takes about $1{,}000$ steps, or $5$ s, and the covariance error is not matched within $1{,}000$ steps.

**AFIN.** AFIN's time starts after the checkpoint is loaded and warmed up; the first call on a fresh process costs about $0.5$ s of kernel compilation, and later forward passes take 10 to 20 ms for $D \leq 32$ (Section E.5). AFIN+SNIS adds the proposal sampling and one density evaluation per proposal, and FRVI+SNIS adds the variational optimization; SNIS curves increase the number of proposals.

**Hardware and code.** Benchmark inference runs on one A100 GPU with 80 GB of memory. Code is available at `https://anonymous.4open.science/r/ICLR_AFIN-F43C/`.

## D.2 Reference posteriors

Every reference posterior comes from a long NumPyro NUTS run (target acceptance $0.8$, maximum tree depth $10$, diagonal mass matrix, chains initialized at the prior location) that never enters any timing. The benchmark references use $10^6$ draws after $10^4$ warmup steps. The Poisson and conjugate task sets use four independent chains per task with the same budget each. Table 9 reports the largest rank-normalized split-$\widehat{R}$ and the smallest bulk and tail effective sample sizes across coordinates

and tasks (Vehtari et al., 2021), together with a four-chain rerun of the full 144-task synthetic grid at $10{,}000$ warmup and $50{,}000$ retained draws per chain, and the 20-task check of the Pyro implementation used for the timed runs. No retained transition in any of these sets diverged or reached the maximum tree depth; the median leapfrog count per draw is 7, and the minimum energy Bayesian fraction of missing information (Betancourt, 2016) is $0.66$.

Table 9: MCMC diagnostics of the reference posteriors. The synthetic-grid row is a four-chain rerun of the 144 benchmark tasks with 10,000 warmup and 50,000 retained draws per chain; the Poisson and conjugate rows are the long four-chain references; the Pyro row is the implementation used for the timed runs, at 1,000 warmup and 2,000 draws per chain. ESS values are rounded to integers.

| Task set | Backend | Tasks | Max. $\widehat{R}$ | Min. bulk ESS | Min. tail ESS |
|---|---|---|---|---|---|
| Synthetic grid (rerun) | NumPyro | 144 | 1.000243 | 120,634 | 68,215 |
| Poisson extension | NumPyro | 24 | 1.0000055 | 1,964,364 | 1,567,853 |
| Conjugate check | NumPyro | 30 | 1.0000058 | 3,624,934 | 2,873,407 |
| Implementation check | Pyro | 20 | 1.0038158 | 4,296 | 2,418 |

**Check against exact posteriors.** Thirty conjugate tasks, with diagonal or full-covariance Gaussian priors and linear-Gaussian likelihoods at the three benchmark sizes plus $(D, N) = (16, 256)$ and $(32, 512)$, three seeds each, have closed-form posteriors. Against them the long references have mean error $0.00063$, relative covariance error $0.0026$, and Gaussian $\mathrm{W}_2$ $0.0023$, so reference error is negligible at the scale of the results. The same tasks also test AFIN with no reference in the loop: at the three benchmark sizes (18 tasks) the direct prediction has mean error $0.047$, relative covariance error $0.073$, and Gaussian $\mathrm{W}_2$ $0.054$, with 90% intervals covering $0.92$ of the true coordinates, and SNIS brings the errors to $0.012$, $0.045$, and $0.020$ with coverage $0.89$.

### D.3 Metrics

Let $z_q^{(1)}, \dots, z_q^{(S)}$ be samples from an approximation and $z_{\mathrm{ref}}^{(1)}, \dots, z_{\mathrm{ref}}^{(S_{\mathrm{ref}})}$ reference samples, with empirical means $\widehat{\mu}$ and covariances $\widehat{\Sigma}$. The three main metrics are

$$\mathrm{M1} = \big\|\widehat{\mu}_q - \widehat{\mu}_{\mathrm{ref}}\big\|_2, \qquad \mathrm{M2} = \big\|\widehat{\Sigma}_q - \widehat{\Sigma}_{\mathrm{ref}}\big\|_F, \qquad \mathrm{SW}_2 = \Big[\frac{1}{R}\sum_{r=1}^{R}\frac{1}{S}\sum_{s=1}^{S}\big(a_{r,(s)} - b_{r,(s)}\big)^2\Big]^{1/2}, \tag{18}$$

where $a_{r,(1)} \le \cdots \le a_{r,(S)}$ and $b_{r,(1)} \le \cdots \le b_{r,(S)}$ are the sorted projections of the two sample sets onto the $r$-th of $R = 128$ random unit directions (Bonneel et al., 2015), with equally many samples from each. For SNIS, samples $z_q^{(s)} \sim q_\phi$ receive weights $\omega_s = \tilde{p}_y(z_q^{(s)})/q_\phi(z_q^{(s)})$, where $\tilde{p}_y(z) = p(z, y_{1:N})$ is the unnormalized posterior, normalized to $\bar{\omega}_s = \omega_s / \sum_j \omega_j$; M1 and M2 then use the weighted mean and covariance, and $\mathrm{SW}_2$ the weighted one-dimensional empirical distributions.

**Coverage.** On simulated tasks the true latent vector is known, so we also check calibration: the fraction of latent coordinates whose true value lies inside the approximation's central marginal interval of nominal level $1 - \alpha$, computed from the Gaussian quantiles for the direct prediction and from weighted quantiles of the proposals for SNIS, and averaged over coordinates, tasks, and seeds. The ideal value is $1 - \alpha$; smaller values mean the posterior is overconfident.

**Weight diagnostics.** For SNIS we report four diagnostics of the weights. The Pareto-$k$ statistic is the shape of a generalized Pareto distribution fitted to the upper tail of the raw weights, as in Pareto-smoothed importance sampling (Vehtari et al., 2024); larger values mean heavier tails (the estimates themselves use the unsmoothed weights). The maximum normalized weight $\max_s \bar{\omega}_s$ is close to one when a few samples dominate. The entropy ratio $-\sum_s \bar{\omega}_s \log \bar{\omega}_s / \log S \in [0, 1]$ is close to one when the weights are diffuse. The target energy gap is the weighted average of $-\log \tilde{p}_y$ over the approximation's samples minus its average over the reference samples; it is near zero when both sets of samples sit at the same target density.

### D.4 SYNTHETIC BENCHMARK

The suite crosses the four prior configurations of Section C with the four likelihood forms that use covariates (the Gaussian likelihood without covariates is trained on but not benchmarked), giving 16 combinations, each at the three sizes in Table 10 with three seeds. AFIN receives only the factor specification and the observations.

Table 10: Synthetic evaluation difficulty levels.

| Difficulty | Latent dimension $D$ | Number of observations $N$ |
|---|---|---|
| Easy | 4 | 256 |
| Medium | 8 | 64 |
| Hard | 16 | 1 |

### D.5 UCI TASKS

Table 11 lists the 12 tasks. Each is a Bayesian regression or classification model with one prior over the coefficient vector $z$ and one likelihood factor per row; continuous covariates are standardized and whitened, and regression targets are standardized. The two heterogeneous tasks start from a regression dataset, keep half of the rows as continuous observations, and turn the other half into binary labels that say whether the response is above the dataset median. Forest Fires models $\log(1 + \text{area})$ with a Student-$t$ likelihood and Mesquite models biomass with a Gaussian likelihood; the labels use Bernoulli-logit likelihoods in both, and both observation types share one coefficient vector. The Bernoulli-logit factor is a modeling choice for these labels, not the likelihood implied by thresholding the continuous model.

Table 11: Real-world task specifications, using the type labels from Section 6.1. All priors have independent coordinates, and Gaussian and Student-t likelihoods use covariates. Rows listing two likelihood types use both within one model. Dataset sources are cited below. (Yeh, 2007; Efron et al., 2004; Gerritsma et al., 1981; Feldmesser, 1987; Quinlan, 1993; Haberman, 1976; Sejnowski and Gorman, 1988; Little, 2007; Janosi et al., 1989; Charytanowicz et al., 2010; Cortez and Morais, 2007; Gelman and Hill, 2007)

| Dataset | $D$ | $N$ | Prior type | Likelihood type(s) |
|---|---|---|---|---|
| Concrete Slump | 7 | 103 | `laplace prior` | `gaussian likelihood` |
| Diabetes | 10 | 442 | `laplace prior` | `gaussian likelihood` |
| Yacht | 6 | 308 | `gaussian prior` | `student-t likelihood` |
| Machine CPU | 6 | 209 | `laplace prior` | `gaussian likelihood` |
| Auto MPG | 7 | 392 | `laplace prior` | `gaussian likelihood` |
| Haberman | 3 | 306 | `gaussian prior` | `bernoulli-logit likelihood` |
| Sonar top-8 | 8 | 208 | `student-t prior` | `bernoulli-logit likelihood` |
| Parkinsons Voice top-8 | 8 | 195 | `gaussian prior` | `bernoulli-logit likelihood` |
| Heart | 13 | 297 | `gaussian prior` | `bernoulli-logit likelihood` |
| Seeds Binary | 7 | 210 | `gaussian prior` | `bernoulli-logit likelihood` |
| Forest Fires Hetero | 8 | 517 | `student-t prior` | `student-t likelihood`<br>`bernoulli-logit likelihood` |
| Mesquite Hetero | 7 | 46 | `gaussian prior` | `gaussian likelihood`<br>`bernoulli-logit likelihood` |

### D.6 EXTRAPOLATION

The six out-of-range configurations are $(D, N) = (24, 128)$ and $(32, 64)$ for OOD $D$; $(12, 400)$ and $(8, 512)$ for OOD $N$; and $(24, 384)$ and $(32, 400)$ for OOD $D, N$, each with three seeds. AFIN draws $5 \times 10^4$ samples, AFIN+SNIS uses $5 \times 10^4$ proposals, NUTS draws $5 \times 10^4$ samples after $10^3$ warmup steps, and FRVI is optimized for $10^4$ steps at the best of seven learning rates and then

sampled $5 \times 10^4$ times. Tables 12 to 14 give means and standard deviations over the six runs in each split, with the mean time per task.

Table 12: Extrapolation results for $SW_2$. Each out-of-distribution (OOD) split contains two configurations and three seeds. $SW_2$ is mean $\pm$ standard deviation; time is the mean per task. Best in bold.

| | OOD $D$ | | OOD $N$ | | OOD $D, N$ | |
|---|---|---|---|---|---|---|
| Method | $SW_2$ | Time (s) | $SW_2$ | Time (s) | $SW_2$ | Time (s) |
| AFIN | $0.031 \pm 0.010$ | 0.02 | $0.027 \pm 0.004$ | 0.02 | $0.040 \pm 0.020$ | 0.10 |
| AFIN+SNIS | $0.024 \pm 0.020$ | 0.24 | $\mathbf{0.0013} \pm 0.0001$ | 0.25 | $\mathbf{0.0039} \pm 0.0010$ | 0.45 |
| NUTS | $\mathbf{0.012} \pm 0.006$ | 308 | $0.0025 \pm 0.0003$ | 95 | $0.0052 \pm 0.0008$ | 157 |
| FRVI | $0.042 \pm 0.040$ | 27 | $0.0025 \pm 0.0004$ | 32 | $0.0041 \pm 0.0007$ | 100 |

Table 13: Extrapolation results for posterior mean error. Entries report mean $L_2$ error, mean $\pm$ standard deviation over six runs, with mean wall-clock time in parentheses. Best in bold.

| Split | AFIN | AFIN+SNIS | NUTS | FRVI |
|---|---|---|---|---|
| OOD $D$ | $0.134 \pm 0.044$ (0.02s) | $0.068 \pm 0.063$ (0.24s) | $\mathbf{0.0277} \pm 0.019$ (308s) | $0.0872 \pm 0.074$ (27s) |
| OOD $N$ | $0.0799 \pm 0.011$ (0.02s) | $\mathbf{0.00185} \pm 0.0006$ (0.25s) | $0.00248 \pm 0.0008$ (95s) | $0.00603 \pm 0.002$ (32s) |
| OOD $D, N$ | $0.205 \pm 0.103$ (0.10s) | $\mathbf{0.00984} \pm 0.005$ (0.45s) | $0.0110 \pm 0.004$ (157s) | $0.0143 \pm 0.003$ (100s) |

Table 14: Extrapolation results for posterior covariance error. Entries report covariance Frobenius error, mean $\pm$ standard deviation over six runs, with mean wall-clock time in parentheses. Best in bold.

| Split | AFIN | AFIN+SNIS | NUTS | FRVI |
|---|---|---|---|---|
| OOD $D$ | $0.234 \pm 0.164$ (0.02s) | $0.323 \pm 0.307$ (0.24s) | $\mathbf{0.131} \pm 0.105$ (308s) | $0.384 \pm 0.356$ (27s) |
| OOD $N$ | $0.00580 \pm 0.0023$ (0.02s) | $\mathbf{0.00115} \pm 0.0004$ (0.25s) | $0.00217 \pm 0.0009$ (95s) | $0.00292 \pm 0.0009$ (32s) |
| OOD $D, N$ | $0.0538 \pm 0.027$ (0.10s) | $\mathbf{0.0175} \pm 0.008$ (0.45s) | $0.0222 \pm 0.007$ (157s) | $0.0248 \pm 0.006$ (100s) |

# E Additional Results

## E.1 Synthetic results by size

Figures 5 to 7 split Figure 2 by size, and Figures 8 to 10 do the same for Figure 4. The picture is the same at every size: AFIN gives a strong one-pass posterior, AFIN+SNIS improves quickly with more proposals, and FRVI needs substantial optimization before it becomes a competitive proposal. The hard, data-poor setting is the most difficult for every method, and AFIN+SNIS stays competitive there with much longer NUTS and FRVI runs.

## E.2 UCI mean and covariance errors

Figures 11 and 12 report M1 and M2 for the UCI tasks, complementing the $SW_2$ curves in Figure 3. The picture is unchanged: AFIN gives a low-cost posterior in one pass, AFIN+SNIS improves it with a small sampling budget, and the iterative baselines need much more time to reach comparable accuracy.

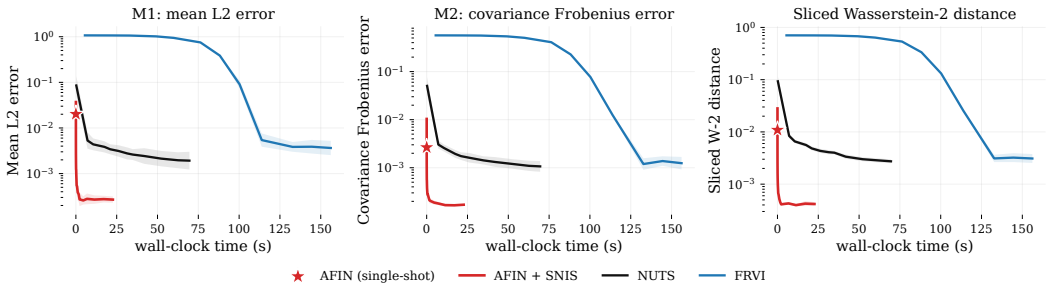



Figure 5: Synthetic posterior accuracy on the easy setting $(D = 4, N = 256)$, averaged over 16 prior–likelihood combinations and three seeds; same protocol as Figure 2.

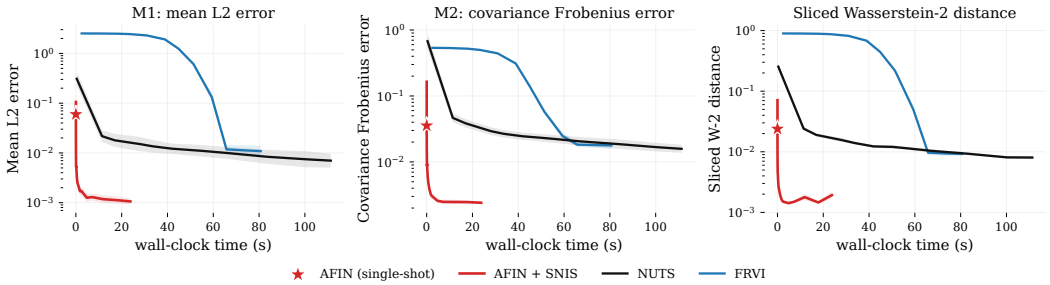



Figure 6: Synthetic posterior accuracy on the medium setting $(D = 8, N = 64)$; same protocol as Figure 2.

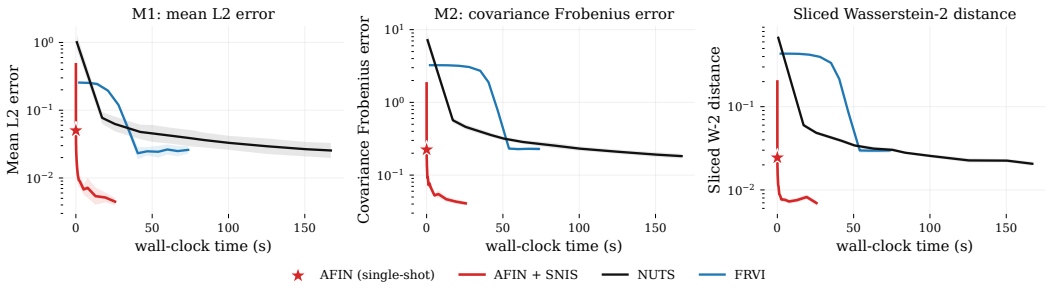



Figure 7: Synthetic posterior accuracy on the hard setting $(D = 16, N = 1)$; same protocol as Figure 2.

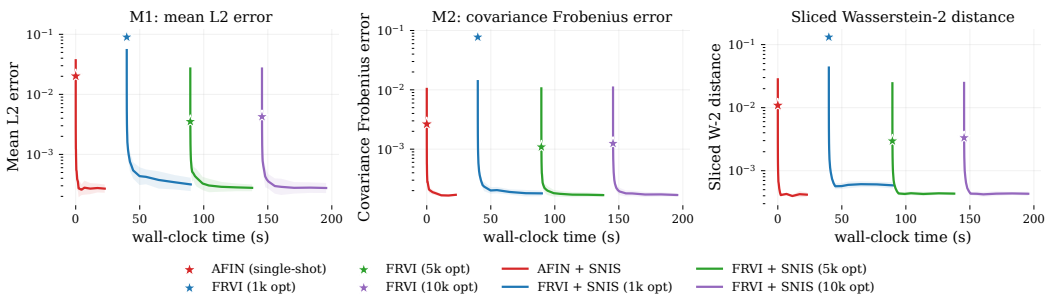



Figure 8: SNIS refinement on the easy setting $(D = 4, N = 256)$; same protocol as Figure 4.

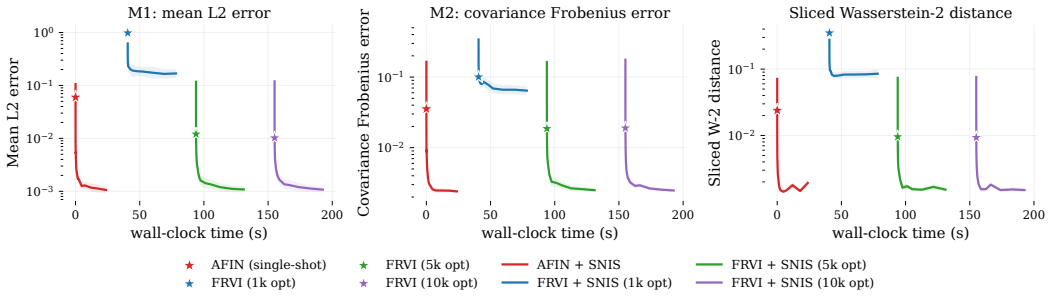



Figure 9: SNIS refinement on the medium setting $(D = 8, N = 64)$; same protocol as Figure 4.

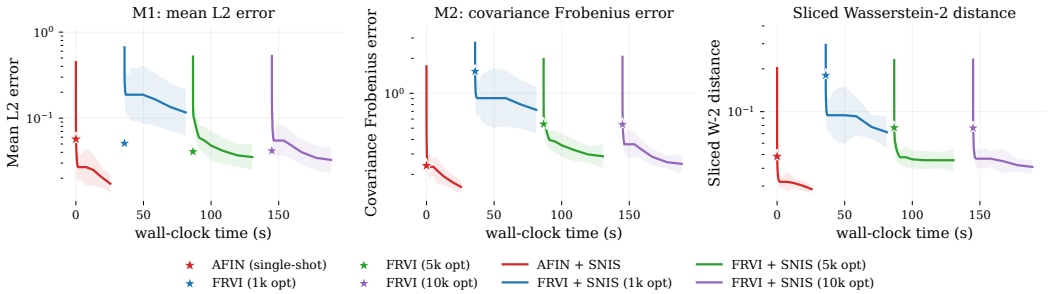



Figure 10: SNIS refinement on the hard setting $(D = 16, N = 1)$; same protocol as Figure 4.

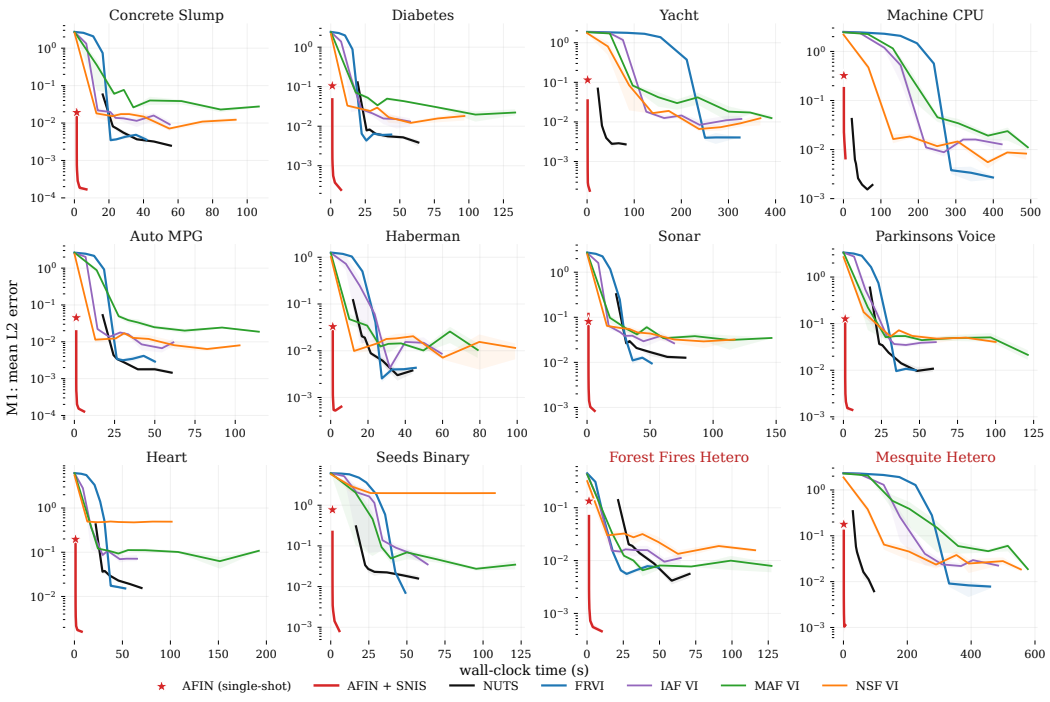



Figure 11: Posterior mean error (M1) against inference time on the 12 UCI tasks; same protocol as Figure 3.

### E.3 SNIS WEIGHT DIAGNOSTICS

Table 15 reports the diagnostics of Section D.3 on the synthetic suite. On the medium tasks, FRVI after 1k steps has a much larger maximum weight, a lower entropy ratio, and a large energy gap, while AFIN's diagnostics are close to those of FRVI after 5k or 10k steps. On the hard tasks, AFIN has the lowest maximum weight of all proposals.

Table 15: SNIS weight diagnostics on the synthetic suite (mean $\pm$ standard deviation over seeds). Lower is better for Pareto-$k$ and maximum weight, higher for entropy ratio, and the energy gap is best near zero.

| Split | Method | Pareto-$k$ | Max weight | Entropy ratio | Energy gap |
|---|---|---|---|---|---|
| Easy | AFIN+SNIS | $0.116_{\pm 0.014}$ | $7.24\times10^{-7}{}_{\pm 1.2\times10^{-7}}$ | $0.998_{\pm 0.000}$ | $-0.001_{\pm 0.000}$ |
| | FRVI+SNIS (1k) | $-0.392_{\pm 0.007}$ | $7.18\times10^{-6}{}_{\pm 2.8\times10^{-7}}$ | $0.902_{\pm 0.000}$ | $-0.002_{\pm 0.001}$ |
| | FRVI+SNIS (5k) | $0.153_{\pm 0.013}$ | $1.62\times10^{-6}{}_{\pm 5.8\times10^{-7}}$ | $1.000_{\pm 0.000}$ | $-0.002_{\pm 0.000}$ |
| | FRVI+SNIS (10k) | $0.160_{\pm 0.023}$ | $1.48\times10^{-6}{}_{\pm 7.0\times10^{-7}}$ | $1.000_{\pm 0.000}$ | $-0.002_{\pm 0.000}$ |
| Medium | AFIN+SNIS | $0.261_{\pm 0.010}$ | $2.04\times10^{-5}{}_{\pm 1.3\times10^{-5}}$ | $0.997_{\pm 0.000}$ | $-0.002_{\pm 0.001}$ |
| | FRVI+SNIS (1k) | $0.558_{\pm 0.018}$ | $8.05\times10^{-2}{}_{\pm 3.0\times10^{-2}}$ | $0.812_{\pm 0.012}$ | $0.401_{\pm 0.154}$ |
| | FRVI+SNIS (5k) | $0.268_{\pm 0.014}$ | $2.17\times10^{-5}{}_{\pm 1.2\times10^{-5}}$ | $0.998_{\pm 0.000}$ | $-0.002_{\pm 0.001}$ |
| | FRVI+SNIS (10k) | $0.266_{\pm 0.016}$ | $2.01\times10^{-5}{}_{\pm 7.7\times10^{-6}}$ | $0.998_{\pm 0.000}$ | $-0.002_{\pm 0.001}$ |
| Hard | AFIN+SNIS | $0.418_{\pm 0.014}$ | $1.93\times10^{-3}{}_{\pm 1.2\times10^{-3}}$ | $0.968_{\pm 0.001}$ | $-0.036_{\pm 0.010}$ |
| | FRVI+SNIS (1k) | $0.560_{\pm 0.016}$ | $1.52\times10^{-2}{}_{\pm 1.7\times10^{-2}}$ | $0.929_{\pm 0.013}$ | $0.031_{\pm 0.220}$ |
| | FRVI+SNIS (5k) | $0.465_{\pm 0.020}$ | $4.38\times10^{-3}{}_{\pm 4.0\times10^{-3}}$ | $0.965_{\pm 0.005}$ | $-0.079_{\pm 0.030}$ |
| | FRVI+SNIS (10k) | $0.463_{\pm 0.018}$ | $4.53\times10^{-3}{}_{\pm 3.6\times10^{-3}}$ | $0.966_{\pm 0.003}$ | $-0.073_{\pm 0.037}$ |
| Overall | AFIN+SNIS | $0.265_{\pm 0.013}$ | $6.51\times10^{-4}{}_{\pm 4.0\times10^{-4}}$ | $0.988_{\pm 0.000}$ | $-0.013_{\pm 0.004}$ |
| | FRVI+SNIS (1k) | $0.242_{\pm 0.014}$ | $3.19\times10^{-2}{}_{\pm 1.5\times10^{-2}}$ | $0.881_{\pm 0.008}$ | $0.144_{\pm 0.125}$ |
| | FRVI+SNIS (5k) | $0.295_{\pm 0.016}$ | $1.47\times10^{-3}{}_{\pm 1.4\times10^{-3}}$ | $0.988_{\pm 0.002}$ | $-0.028_{\pm 0.011}$ |
| | FRVI+SNIS (10k) | $0.296_{\pm 0.019}$ | $1.52\times10^{-3}{}_{\pm 1.2\times10^{-3}}$ | $0.988_{\pm 0.001}$ | $-0.026_{\pm 0.013}$ |

### E.4 CALIBRATION

Coverage (Section D.3) asks whether the predicted uncertainty is right, which the distance metrics do not. We measured it on 200 simulated tasks spanning all prior configurations and likelihood forms, including mixed models: 120 inside the training ranges, 40 with $N$ from 384 to $1{,}024$, and 40 with $D$ from 24 to 64. Inside the training ranges the direct prediction is close to calibrated: its nominal 50, 80, 90, and 95% intervals cover $0.47$, $0.77$, $0.875$, and $0.93$ of the true coordinates, a mean absolute deviation of $2.6$ percentage points, and SNIS with $4{,}096$ proposals leaves this unchanged ($0.871$ at the 90% level). Just above the training range, on a separate 64-task panel with $N$ between 320 and 400, 90% coverage is $0.907$. Further out it falls to $0.57$ for $N$ up to $1{,}024$ and $0.74$ for $D$ from 24 to 64, the same boundary at which accuracy degrades in Table 3. The conjugate tasks of Section D.2, with exact posteriors, agree: 90% coverage $0.92$ direct and $0.89$ after SNIS.

### E.5 FORWARD TIME AND MEMORY

Table 16 measures one warmed forward pass of the trained model on a single GPU, as the median over synchronized repetitions, together with peak GPU memory. Time is flat at about 10 ms up to $D = 16$ or $N = 64$ and then grows with the pair arrays and the attention scores, as Section B.5 predicts, reaching 67 ms and 1.1 GB at $D = 64$ and 71 ms and 1.3 GB at $N = 1{,}024$. The first call on a fresh process is slower, about $0.5$ s, because of kernel compilation.

### E.6 POISSON EXTENSION

**Tasks.** The new likelihood is $y_n \mid z \sim \mathrm{Poisson}(\exp(x_n^\top z))$, with the four pretraining prior configurations and covariates scaled by $0.5$ to keep the rates moderate; all likelihood factors in these tasks are Poisson. The held-out set has 24 tasks, two for each $D \in \{2, 4, 8, 16\}$ and $N \in \{16, 64, 256\}$. References are four-chain NumPyro runs as in Section D.2; M1 and M2 use all

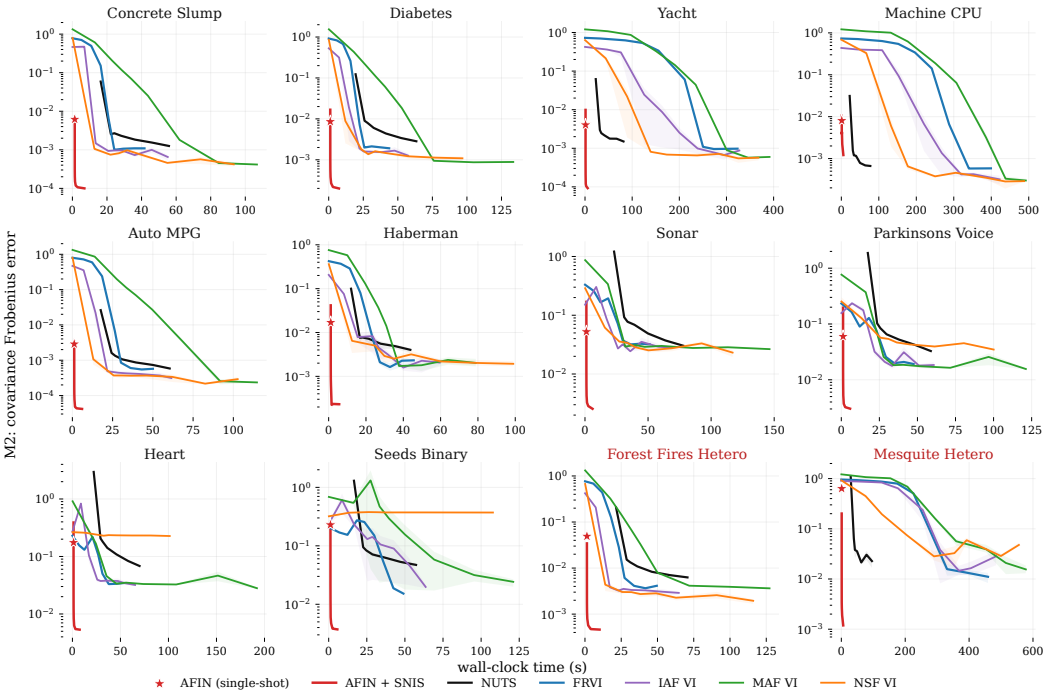



Figure 12: Posterior covariance error (M2) against inference time on the 12 UCI tasks; same protocol as Figure 3.

Table 16: Warmed forward-pass time and peak GPU memory of the trained model. Left: $D$ varies at $N = 64$; right: $N$ varies at $D = 16$.

| $D$ | Time (ms) | Memory (MB) | $N$ | Time (ms) | Memory (MB) |
|---|---|---|---|---|---|
| 4 | 9.8 | 54 | 1 | 9.8 | 38 |
| 8 | 9.8 | 64 | 16 | 9.9 | 49 |
| 16 | 9.5 | 101 | 64 | 9.1 | 101 |
| 24 | 13.2 | 187 | 256 | 17.8 | 316 |
| 32 | 17.7 | 308 | 512 | 35.3 | 618 |
| 48 | 38.9 | 650 | 1,024 | 71.4 | 1,272 |
| 64 | 66.7 | 1,130 | | | |

reference draws, and $SW_2$ uses $4{,}096$ of them against $4{,}096$ samples or proposals. These tasks are a separate set, so their absolute errors are not comparable with the benchmark curves.

**Adapter.** The Poisson adapter follows Section B.1. Its node descriptor has five entries, $y_n$, $\log(1 + y_n)$, $\log(1 + \|x_n\|_2)$, $x_{ni}$, and $y_n x_{ni}$; its pair descriptor has nine, the same three scalars, $x_{ni}$, $x_{nj}$, their product, both count–covariate products, and a diagonal indicator. With one hidden layer of width 64 and $C = 40$ output features it has $6{,}224$ parameters.

**Training.** Both modes start from the pretrained AFIN and run $30{,}000$ AdamW steps (batch size 32, four accumulated micro-batches, weight decay $10^{-4}$, 500 warmup steps, cosine decay) on tasks with $D$ uniform in $\{1, \ldots, 16\}$ and $N$ in $\{1, \ldots, 256\}$, cycling through the prior configurations and covariate generators. Adapter-only training uses learning rate $10^{-3}$ and an exponential moving average of the adapter weights; full fine-tuning uses $2 \times 10^{-4}$ and the final weights. Each mode ran three seeds, all reported. After adapter-only training the shared network is unchanged, so every original factor type behaves exactly as before. Tables 17 to 19 give $SW_2$, the per-seed results, and mean error, covariance error, and 90% coverage.

Table 17: Poisson extension: $SW_2$ and training time. Entries are means $\pm$ sample standard deviations over three training seeds; each accuracy value first averages over 24 held-out tasks. SNIS uses 4,096 proposals. Best in bold.

| | Posterior error ($SW_2$) | | Training time |
|---|---|---|---|
| Training mode | Direct | +SNIS | (GPU-h) |
| Adapter only | $0.05910 \pm 0.00198$ | $\mathbf{0.03011} \pm 0.00019$ | $\mathbf{4.946} \pm 0.015$ |
| Full fine-tuning | $\mathbf{0.04580} \pm 0.00055$ | $0.03048 \pm 0.00036$ | $8.392 \pm 0.073$ |

Table 18: Poisson extension by training seed: mean $SW_2$ over the same 24 tasks.

| | Adapter only | | Full fine-tuning | |
|---|---|---|---|---|
| Seed | Direct | +SNIS | Direct | +SNIS |
| Seed 1 | 0.06139 | 0.03029 | 0.04606 | 0.03071 |
| Seed 2 | 0.05801 | 0.03013 | 0.04518 | 0.03067 |
| Seed 3 | 0.05789 | 0.02991 | 0.04618 | 0.03007 |

Table 19: Additional Poisson results, averaged over tasks within each training seed and then reported as mean $\pm$ sample standard deviation across three seeds. Mean and covariance errors use the Euclidean and Frobenius norms. Coverage is the fraction of simulated latent coordinates within nominal 90% marginal intervals. Best in bold; for coverage, closest to the nominal 90%.

| Training | Inference | Mean error | Covariance error | 90% coverage |
|---|---|---|---|---|
| Adapter only | Direct | $0.1166 \pm 0.0039$ | $0.1565 \pm 0.0034$ | $\mathbf{0.8993} \pm 0.0030$ |
| Adapter only | +SNIS | $\mathbf{0.0319} \pm 0.0005$ | $\mathbf{0.1135} \pm 0.0035$ | $0.9306 \pm 0.0015$ |
| Full fine-tuning | Direct | $0.0785 \pm 0.0014$ | $0.1401 \pm 0.0065$ | $0.9219 \pm 0.0000$ |
| Full fine-tuning | +SNIS | $0.0331 \pm 0.0006$ | $0.1147 \pm 0.0031$ | $0.9314 \pm 0.0015$ |

### E.7 Ablations

The variants of Table 5 keep AFIN's adapters and Gaussian decoder. The node-only variants drop the pair array everywhere, so the decoder predicts the precision from node features alone. The sum variants remove the Transformer blocks and pass the sum of the encoded factors straight to the decoder, which is the conjugate template of Section 3.2 with a learned encoder and decoder. Table 20 gives the direct results at full precision, with one standard deviation of $SW_2$ across evaluation repetitions.

Table 20: Ablations on the synthetic suite, direct prediction; best in bold.

| Variant | M1 | M2 | $SW_2$ |
|---|---|---|---|
| Node and pair, attention (AFIN) | **0.0533** | **0.1047** | $\mathbf{0.0412} \pm 0.0003$ |
| Node only, attention | 0.0734 | 0.2078 | $0.0490 \pm 0.0003$ |
| Node and pair, sum | 0.2903 | 0.2484 | $0.1324 \pm 0.0009$ |
| Node only, sum | 0.3140 | 0.2758 | $0.1425 \pm 0.0008$ |

### E.8 Amortized baseline

NPE-PFN (Vetter et al., 2025) feeds prior-predictive simulations of a task to a tabular foundation model as context and reads the posterior off its in-context predictions, so it needs no task-specific training. We ran the official implementation with frozen TabPFN v2 weights (Hollmann et al., 2025) on the 144 synthetic tasks, with 1,000 or 10,000 prior-predictive simulations as context and 2,048 posterior samples per task, scored against the references of Section D.2; the 1,000-simulation context is a prefix of the 10,000. AFIN's moments are closed-form and NPE-PFN's are estimated from its samples; $SW_2$ uses 2,048 samples for both. Times are per task for inference and sampling, excluding the context simulations. Table 21 splits the results by difficulty. AFIN has the lowest error in every block; the larger context reduces NPE-PFN's errors but costs 115 s per task. With the same 2,048-proposal SNIS correction applied to both, AFIN keeps the lower mean and $SW_2$ errors and a higher effective sample size, 0.81 against 0.60 of the proposals.

Table 21: NPE-PFN against AFIN on the synthetic suite by difficulty; best in bold.

| Tasks | Method | M1 | M2 | $SW_2$ |
|---|---|---|---|---|
| All (144) | AFIN | **0.0533** | **0.1047** | **0.0415** |
| | NPE-PFN, 1,000 context simulations | 0.2417 | 0.3008 | 0.1133 |
| | NPE-PFN, 10,000 context simulations | 0.1298 | 0.2542 | 0.0761 |
| Easy (48) | AFIN | **0.0238** | **0.0039** | **0.0158** |
| | NPE-PFN, 1,000 | 0.1691 | 0.0720 | 0.1038 |
| | NPE-PFN, 10,000 | 0.0923 | 0.0421 | 0.0628 |
| Medium (48) | AFIN | **0.0644** | **0.0395** | **0.0353** |
| | NPE-PFN, 1,000 | 0.3453 | 0.1111 | 0.1281 |
| | NPE-PFN, 10,000 | 0.1848 | 0.0875 | 0.0723 |
| Hard (48) | AFIN | **0.0716** | **0.2707** | **0.0734** |
| | NPE-PFN, 1,000 | 0.2106 | 0.7193 | 0.1079 |
| | NPE-PFN, 10,000 | 0.1124 | 0.6331 | 0.0932 |

## F Extended Related Work

**Expressive posterior approximators.** Normalizing flows give tractable, invertible maps from a simple base density to flexible posteriors (Rezende and Mohamed, 2015; Papamakarios et al., 2021). Coupling and autoregressive designs such as RealNVP, IAF, MAF, and neural spline flows (Dinh et al., 2017; Kingma et al., 2016; Papamakarios et al., 2017; Durkan et al., 2019; Agrawal and Domke, 2025; Ko and Domke, 2025) serve both as variational families and as conditional density estimators in simulation-based inference (Papamakarios and Murray, 2016; Greenberg et al., 2019; Papamakarios et al., 2019; Radev et al., 2022). Diffusion and score-based models have been adapted to posterior sampling for inverse problems (Chung et al., 2023) and to simulation-based inference by learning or composing posterior scores (Geffner et al., 2023; Sharrock et al., 2024), and flow matching trains continuous flows by vector-field regression (Lipman et al., 2023; Tong et al., 2024; Albergo et al., 2025), including for scalable simulation-based inference (Wildberger et al., 2023). All of these operate within a fixed model, simulator, or task interface; they are complementary to AFIN's factor interface, which could feed a more expressive decoder.

**Amortization over model families and programs.** BayesFlow learns globally amortized posterior estimators with invertible networks and learned summaries for a fixed simulator family (Radev

et al., 2022), and JANA jointly amortizes the posterior and the likelihood, which also gives marginal likelihoods and posterior predictives (Radev et al., 2023). SA-ABI shares weights across alternative prior, likelihood, and data choices for sensitivity analysis (Elsemüller et al., 2024). Prior-fitted networks learn Bayesian prediction over supervised tasks drawn from a task prior and target predictive rather than latent posteriors (Müller et al., 2022); ACE conditions on data, latents, and priors through a general token interface (Chang et al., 2025); inference compilation trains proposals for a probabilistic program and uses them inside sequential importance sampling (Le et al., 2017); and meta-amortized VI shares inference computation across related generative models (Wu et al., 2020). None of these exposes the inference problem as a typed factor list with dimension-independent decoding, which is what lets one AFIN cover every column of Table 1 without per-task fitting.

**Message passing.** The observation behind our merge, that the useful summary of one factor depends on where the other factors put their mass, is also the basis of expectation propagation (Minka, 2001), which refines each factor's approximation against the product of the others. AFIN replaces this iterative, model-specific computation with one learned exchange of information across factors, amortized over models.