LEARN TO ADAPT DOUBLY SEQUENTIAL BAYESIAN NEURAL NETWORKS WITH COMPLEX ARCHITECTURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks used for control must adapt from a few sequentially arriving samples, often drawn from suboptimal data, and do so efficiently. Bayesian neural networks (BNNs) provide a principled framework for sample-efficient sequential adaptation with quantified uncertainty, but few techniques can use data to optimize that adaptation at scale. In this paper, we first show that adaptation can be learned from data by optimizing its backward update one layer at a time. This stands in stark contrast to existing meta-learning techniques that learn only from input–output matching and become prohibitive for large networks. The key enabler is to view training as a sequence of layerwise Bayesian posterior updates, which decomposes the problem into optimizing the update rule of each layer. Layerwise moment propagation of this kind was previously confined to feedforward networks; we extend it to computation graphs composed of attention, normalization, feed-forward, residual, and branching operations, so that transformers and other architectures can use data-driven techniques to optimize adaptation/fine tuning at scale. In parallel, we generalize BNNs to a hybrid of imitation-based and reinforcement-based learning so that BNNs learn a more optimal mapping than the data distribution. This builds on a new perspective: the uncertainty a BNN maintains can be repurposed to distinguish optimal from suboptimal samples and to weight the update toward the former. Finally, we integrate the two parts into a data-driven framework to optimize both the update rules of individual layers and the balance between imitation and reinforcement. The proposed framework allows the adaptation algorithm to use past adaptation histories to self-evolve toward improved sample efficiency, optimality, and robustness.

1 INTRODUCTION

Neural networks used for control must adapt efficiently from a few sequentially arriving samples, often collected in changing environments or generated by suboptimal policies. Bayesian neural networks (BNNs) support this setting by maintaining uncertainty over parameters and predictions and incorporating observations through sequential posterior updates (MacKay, 1992; Neal, 1996). Yet effective adaptation requires learning both how to update the network and which behaviors those updates should favor.

Meta-learning uses experience across tasks to improve adaptation. Widely used approaches optimize task-level input–output losses through network-wide learning trajectories, which can become costly as networks and adaptation horizons grow (Andrychowicz et al., 2016; Finn et al., 2017). Moreover, these losses do not directly supervise internal posterior updates. Prior work has exploited layer-specific representations or meta-learned layer-specific optimization hyperparameters (Jang et al., 2019; Baik et al., 2020), but these mechanisms are ultimately learned through downstream task losses rather than direct supervision of the internal adaptation dynamics themselves.

Our key insight is that expressing training as layerwise Bayesian posterior updates makes the adaptation rule itself amenable to layerwise learning of the internal Bayesian posterior-update operator. Doubly sequential Bayesian neural networks (DSBNNs), which perform inference sequentially over incoming data and backward through network layers (Wagner et al., 2023; Jing et al., 2025), turns a network-wide adaptation problem into a sequence of explicit layerwise posterior updates. We exploit this structure to make the backward update operator learnable from data. We first optimize these

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.

update rules from output to input, layer by layer, and subsequently refine them jointly to account for interactions across sequential updates. Once learned, adaptation requires only a single forward and backward moment pass for each new sample.

To enable this approach in complex architectures, we extend moment propagation beyond chain-structured feedforward networks. Attention, normalization, residual connections, and branching introduce dependencies that require additional propagation rules. We develop composable local moment rules and a blockwise backward recursion that accommodate these operations and their shared stochastic dependencies, making layerwise learning to adapt applicable to transformers and other complex computation graphs.

We also extend adaptation beyond reproducing suboptimal behavior. The BNN's predictive distribution represents uncertainty over candidate actions, while value information reweights this distribution toward higher-return actions. Building on statewise Kullback–Leibler (KL)-regularized policy improvement (Abdolmaleki et al., 2018), the resulting reward-tilted distribution supplies target moments for the same backward update mechanism. This combines imitation with reward-guided improvement within doubly sequential Bayesian learning. We further learn a task-dependent reward temperature from adaptation histories, allowing the balance between preserving reference behavior and pursuing improvement to vary across problems.

Together, these developments yield a framework that learns both the adaptation mechanism and its learning objective. Our contributions are:

- **Layerwise learning to adapt:** learning backward update rules from expert posterior moments, enabled by moment propagation through complex computation graphs.
- **Adaptation beyond imitation:** integrating reward-weighted output targets into uncertainty-aware, doubly sequential updates.
- **Data-driven adaptation design:** learning layer-specific gains and the imitation–reinforcement balance from experience across tasks.

After introducing the notation and preliminaries, the rest of the paper is organized as follows. Section 4 develops graph-based moment propagation and layerwise learning to adapt. Section 5 introduces reward-guided adaptation and the learned imitation–reinforcement balance. Section 6 presents the numerical simulation results.

2 Notation

Scalars, vectors, and matrices use x, $\mathbf{x}$, and $\mathbf{X}$, respectively; calligraphic symbols denote structured objects such as sets, laws, datasets, caches, bundles, and functionals. The operator $\mathrm{vec}(\mathbf{X})$ stacks columns. When a random matrix $\mathbf{X}$ appears in a covariance, we write $\mathbf{x} = \mathrm{vec}(\mathbf{X})$. We write $\mu_{\mathbf{x}} = \mathbb{E}[\mathbf{x}]$, $\mu_{\mathbf{X}} = \mathbb{E}[\mathbf{X}]$, $\Sigma_{\mathbf{x}} = \mathrm{Cov}(\mathbf{x})$, and $\Sigma_{\mathbf{x},\mathbf{y}} = \mathrm{Cov}(\mathbf{x}, \mathbf{y})$; every covariance subscript therefore names a vector random variable. Superscript $\top$ denotes transpose, and $+$ denotes moments after the current assimilation. Gaussian regressions use nonredundant random coordinates; covariance and cross-covariance blocks in a regression are understood in those coordinates.

3 Preliminaries

3.1 Chain-structured Neural Network

Existing DSBNN formulations consider feed-forward networks with input $\mathbf{x} \in \mathbb{R}^{d_x}$ and output $\mathbf{y} \in \mathbb{R}^{d_y}$. The network has L_{net} linear layers, with $d_0 = d_x$ and $d_{L_{\mathrm{net}}} = d_y$. Let $\mathbf{W}_\ell \in \mathbb{R}^{d_\ell \times (d_{\ell-1}+1)}$ be the weight matrix of layer ℓ, including its bias column. Let Ω_ω denote the parameter space, and let $\boldsymbol{\omega} = [\mathbf{w}_1^\top \cdots \mathbf{w}_{L_{\mathrm{net}}}^\top]^\top$ collect all parameters. For $\mathbf{z}_0 = \mathbf{x}$, the forward map is

$$\mathbf{w}_\ell = \mathrm{vec}(\mathbf{W}_\ell), \tag{1}$$

$$\mathbf{u}_\ell = \mathbf{W}_\ell \,\mathrm{aug}(\mathbf{z}_{\ell-1}), \tag{2}$$

$$\mathbf{z}_\ell = \varphi_\ell(\mathbf{u}_\ell), \qquad \ell < L_{\mathrm{net}}, \tag{3}$$

$$\mathbf{f}_{\boldsymbol{\omega}}(\mathbf{x}) = \mathbf{u}_{L_{\mathrm{net}}}. \tag{4}$$

Here aug appends a 1, and φ_ℓ is the activation after layer ℓ. For brevity, later derivations suppress the bias column.

3.2 Supervised Learning in Bayesian Neural Networks

Let $\mathcal{P}_{\mathrm{out}}(d\mathbf{y} \mid \mathbf{x}, \boldsymbol{\omega})$ denote the network's output kernel. For a deterministic readout it is the Dirac measure at $\mathbf{f}_{\boldsymbol{\omega}}(\mathbf{x})$; a supplied observation model or stochastic policy instead defines a nondegenerate kernel. Given a parameter distribution $p(\boldsymbol{\omega})$, the predictive law is

$$\mathcal{P}^{\mathrm{pred}}(d\mathbf{y} \mid \mathbf{x}) = \int_{\Omega_\omega} \mathcal{P}_{\mathrm{out}}(d\mathbf{y} \mid \mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \tag{5}$$

When this law has a density, we write it as $p(\mathbf{y} \mid \mathbf{x})$. For supervised data $\mathcal{D} = [(\mathbf{x}_k, \mathbf{y}_k)]_{k=1}^{N_{\mathrm{data}}}$, the objective is to approximate $p(\boldsymbol{\omega} \mid \mathcal{D})$ and hence $\mathcal{P}^{\mathrm{pred}}(d\mathbf{y} \mid \mathbf{x}, \mathcal{D})$, obtained from (5) by replacing the prior with the posterior.

3.3 Doubly Sequential Bayesian Neural Network

A doubly sequential Bayesian neural network admits sequential updates over both records and layers. For records that are conditionally independent given $\boldsymbol{\omega}$, $p(\boldsymbol{\omega} \mid \mathcal{D}_{1:k}) \propto p(\mathcal{D}_k \mid \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathcal{D}_{1:k-1})$, where $\mathcal{D}_k = (\mathbf{x}_k, \mathbf{y}_k)$ and $\mathcal{D}_{1:k} = [(\mathbf{x}_j, \mathbf{y}_j)]_{j=1}^{k}$. Under the retained layerwise factorization, the recursion for $\ell < L_{\mathrm{net}}$ uses the graph $(\mathbf{z}_{\ell-1}, \mathbf{w}_\ell) \to \mathbf{u}_\ell \to \mathbf{z}_\ell$; the output layer terminates at $\mathbf{f}_{\boldsymbol{\omega}}(\mathbf{x}) = \mathbf{u}_{L_{\mathrm{net}}}$. For record k, the forward pass retains the joint moments of $(\mathbf{w}_\ell, \mathbf{u}_\ell)$ and $(\mathbf{z}_{\ell-1}, \mathbf{u}_\ell)$ for $1 \le \ell \le L_{\mathrm{net}}$, together with $(\mathbf{u}_\ell, \mathbf{z}_\ell)$ for $1 \le \ell < L_{\mathrm{net}}$, all conditional on $(\mathcal{D}_{1:k-1}, \mathbf{x}_k)$; the backward pass updates $p(\mathbf{w}_\ell \mid \mathcal{D}_{1:k})$ from $\ell = L_{\mathrm{net}}$ down to 1.

4 Moment Propagation and Learning to Adapt for Diverse Network Architectures

In this section, we show how to learn layer-specific Bayesian update rules in complex neural networks. By extending moment propagation to attention, normalization, residual, and branching operations, we expose internal posterior updates that can be calibrated from adaptation histories while retaining single-pass adaptation at deployment. Section 4.1 introduces the local moment propagation rules required to propagate uncertainties through these operations. Section 4.2 composes them into a graph-wide forward pass, with an example in Appendex D on application to transformers. Section 1 constructs the backward pass updates and proposes how to learn their layer-specific gains from expert posterior moments, connecting the propagation machinery to learning how to adapt.

We represent a neural network as a finite directed acyclic computation graph whose nodes are evaluated in topological order. Each node receives one or more parent variables and, when applicable, stochastic parameters. The forward pass propagates first and second moments and retains the cross-covariances used by a Kalman-smoothing-style (Rauch et al., 1965) backward pass.

The propagation rules maintain a joint-Gaussian approximation. Bilinear nodes use conditional-independence closure, nonlinear nodes are moment-matched, and nonlocal cross-covariances use Gaussian regression. For a node with stacked stochastic input $\mathbf{x}_{\mathrm{in}}$ and output $\mathbf{y}_{\mathrm{out}}$, every retained external vector $\mathbf{v}_{\mathrm{ext}}$ is propagated by

$$\Sigma_{\mathbf{v}_{\mathrm{ext}}, \mathbf{y}_{\mathrm{out}}} \approx \Sigma_{\mathbf{v}_{\mathrm{ext}}, \mathbf{x}_{\mathrm{in}}} \Sigma_{\mathbf{x}_{\mathrm{in}}}^{-1} \Sigma_{\mathbf{x}_{\mathrm{in}}, \mathbf{y}_{\mathrm{out}}}. \tag{6}$$

The bilinear and additive rules below use their direct Gaussian-moment formulas instead of this regression closure.

4.1 Moment-Propagation Nodes

Each node rule supplies an output mean and covariance, the parent–output cross-covariances used by the backward pass, and a rule for propagating cross-covariances with retained stochastic sites and other external variables. We derive new forward moment-propagation rules for the correlated bilinear and LayerNorm nodes in Appendices B.1 and B.2, with implementations in Algorithms 2 and 3. The remaining nodes use existing moment approximations or standard covariance identities.

Bilinear matrix node. Let $\mathbf{X}_3 = \mathbf{X}_1\mathbf{X}_2$, where $\mathbf{X}_1$ and $\mathbf{X}_2$ share a stochastic ancestor $\mathbf{X}_0$. We approximate $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ as jointly Gaussian and impose $\mathbf{x}_1 \perp \mathbf{x}_2 \mid \mathbf{x}_0$. The resulting BILINEARMATRIXMOMENTS rule maps the input means and covariance blocks to $\mu_{\mathbf{X}_3}$, $\Sigma_{\mathbf{x}_3}$, and $\Sigma_{\mathbf{x}_\iota,\mathbf{x}_3}$ for $\iota \in \{0,1,2\}$. Equation (41) propagates retained external cross-covariances.

LayerNorm node. For one row, let

$$\begin{aligned}\mathbf{y}_{\mathrm{LN}} &= \mathbf{b}_{\mathrm{LN}} + \mathbf{g}_{\mathrm{LN}} \odot \mathbf{n}_{\mathrm{LN}},\\ \mathbf{n}_{\mathrm{LN}} &= \frac{\mathbf{C}_{\mathrm{LN}}\mathbf{x}_{\mathrm{LN}}}{\sqrt{d_{\mathrm{LN}}^{-1}\mathbf{x}_{\mathrm{LN}}^\top \mathbf{C}_{\mathrm{LN}}\mathbf{x}_{\mathrm{LN}}}},\\ \mathbf{C}_{\mathrm{LN}} &= \mathbf{I}_{d_{\mathrm{LN}}} - \frac{1}{d_{\mathrm{LN}}}\mathbf{1}_{d_{\mathrm{LN}}}\mathbf{1}_{d_{\mathrm{LN}}}^\top.\end{aligned} \tag{7}$$

With $\boldsymbol{\theta}_{\mathrm{LN}} = [\mathbf{b}_{\mathrm{LN}}^\top\ \mathbf{g}_{\mathrm{LN}}^\top]^\top$, the LAYERNORMHYBRIDMOMENTS rule approximates $(\mathbf{x}_{\mathrm{LN}}, \boldsymbol{\theta}_{\mathrm{LN}})$ as jointly Gaussian. Affine parameters with zero prior covariance remain deterministic and are excluded from smoothing sites. One-dimensional Laplace-transform quadrature gives the normalized moments; Gaussian product identities then yield the affine-output moments and retained cross-covariances. Matrix inputs are processed in a fixed row order so that cross-row dependence from shared affine parameters is retained.

Independent matrix-product node. For independent inputs, $\mathbf{Y}_{\mathrm{prod}} = \mathbf{X}_{1,\mathrm{prod}}\mathbf{X}_{2,\mathrm{prod}}$. Moment propagation follows jing2025kalman. With lower-case vectorizations,

$$\begin{aligned}&\textsc{MatrixProductMoments}(\mu_{\mathbf{X}_{1,\mathrm{prod}}}, \mu_{\mathbf{X}_{2,\mathrm{prod}}}, \Sigma_{\mathbf{x}_{1,\mathrm{prod}}}, \Sigma_{\mathbf{x}_{2,\mathrm{prod}}})\\ &\quad\longrightarrow (\mu_{\mathbf{Y}_{\mathrm{prod}}}, \Sigma_{\mathbf{y}_{\mathrm{prod}}}, \Sigma_{\mathbf{x}_{1,\mathrm{prod}},\mathbf{y}_{\mathrm{prod}}}, \Sigma_{\mathbf{x}_{2,\mathrm{prod}},\mathbf{y}_{\mathrm{prod}}}).\end{aligned} \tag{8}$$

For each retained $\mathbf{v}_{\mathrm{ext}}$, the routine also accepts $\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{1,\mathrm{prod}}}$ and $\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{2,\mathrm{prod}}}$ and applies (41) with the corresponding product-node indices and means.

Softmax node. The row-wise map $\mathbf{x}_{\mathrm{sm}} \mapsto \mathbf{y}_{\mathrm{sm}}$ uses the moment approximation of jing2025kalman, with interface $\textsc{SoftmaxMoments}(\mu_{\mathbf{x}_{\mathrm{sm}}}, \Sigma_{\mathbf{x}_{\mathrm{sm}}}) \to (\mu_{\mathbf{y}_{\mathrm{sm}}}, \Sigma_{\mathbf{y}_{\mathrm{sm}}}, \Sigma_{\mathbf{x}_{\mathrm{sm}},\mathbf{y}_{\mathrm{sm}}})$. The rule permits a fixed mask. When present, the moment map is applied to the unmasked logits in each row and the masked probabilities are restored as deterministic zeros. Without a mask, all coordinates are active. For row r and a retained external vector, Gaussian regression gives

$$\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{y}_{\mathrm{sm},r}} \approx \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{\mathrm{sm},r}}\Sigma_{\mathbf{x}_{\mathrm{sm},r}}^{-1}\Sigma_{\mathbf{x}_{\mathrm{sm},r},\mathbf{y}_{\mathrm{sm},r}}. \tag{9}$$

For distinct rows r and r', the retained cross-row block is

$$\Sigma_{\mathbf{y}_{\mathrm{sm},r},\mathbf{y}_{\mathrm{sm},r'}} \approx \Sigma_{\mathbf{x}_{\mathrm{sm},r},\mathbf{y}_{\mathrm{sm},r}}^\top\Sigma_{\mathbf{x}_{\mathrm{sm},r}}^{-1}\Sigma_{\mathbf{x}_{\mathrm{sm},r},\mathbf{x}_{\mathrm{sm},r'}}\Sigma_{\mathbf{x}_{\mathrm{sm},r'}}^{-1}\Sigma_{\mathbf{x}_{\mathrm{sm},r'},\mathbf{y}_{\mathrm{sm},r'}}. \tag{10}$$

With a mask, all terms in (9)– (10) are formed on active coordinates before deterministic masked zeros are restored.

Piecewise-linear node. The node $\mathbf{x}_{\mathrm{pw}} \to \mathbf{y}_{\mathrm{pw}}$ applies a deterministic piecewise-linear activation such as ReLU. Moment propagation follows wright2024analytic, with interface $\textsc{PiecewiseLinearMoments}(\mu_{\mathbf{x}_{\mathrm{pw}}}, \Sigma_{\mathbf{x}_{\mathrm{pw}}}) \to (\mu_{\mathbf{y}_{\mathrm{pw}}}, \Sigma_{\mathbf{y}_{\mathrm{pw}}}, \Sigma_{\mathbf{x}_{\mathrm{pw}},\mathbf{y}_{\mathrm{pw}}})$. External cross-covariances use (6) with $\mathbf{x}_{\mathrm{in}} = \mathbf{x}_{\mathrm{pw}}$.

Additive matrix node. The node $\mathbf{X}_{1,\mathrm{add}}, \mathbf{X}_{2,\mathrm{add}} \to \mathbf{Y}_{\mathrm{add}}$ has $\mathbf{Y}_{\mathrm{add}} = \mathbf{X}_{1,\mathrm{add}} + \mathbf{X}_{2,\mathrm{add}}$. Writing lower-case symbols for vectorizations, $\mu_{\mathbf{Y}_{\mathrm{add}}} = \mu_{\mathbf{X}_{1,\mathrm{add}}} + \mu_{\mathbf{X}_{2,\mathrm{add}}}$ and

$$\begin{aligned}\Sigma_{\mathbf{y}_{\mathrm{add}}} &= \Sigma_{\mathbf{x}_{1,\mathrm{add}}} + \Sigma_{\mathbf{x}_{2,\mathrm{add}}} + \Sigma_{\mathbf{x}_{1,\mathrm{add}},\mathbf{x}_{2,\mathrm{add}}} + \Sigma_{\mathbf{x}_{1,\mathrm{add}},\mathbf{x}_{2,\mathrm{add}}}^\top,\\ \Sigma_{\mathbf{x}_{1,\mathrm{add}},\mathbf{y}_{\mathrm{add}}} &= \Sigma_{\mathbf{x}_{1,\mathrm{add}}} + \Sigma_{\mathbf{x}_{1,\mathrm{add}},\mathbf{x}_{2,\mathrm{add}}},\\ \Sigma_{\mathbf{x}_{2,\mathrm{add}},\mathbf{y}_{\mathrm{add}}} &= \Sigma_{\mathbf{x}_{1,\mathrm{add}},\mathbf{x}_{2,\mathrm{add}}}^\top + \Sigma_{\mathbf{x}_{2,\mathrm{add}}}.\end{aligned} \tag{11}$$

Thus MATRIXSUMMOMENTS takes the two means, two covariances, and their cross-covariance and returns the output mean, output covariance, and the two input–output cross-covariances in that order. For any retained external vector, the exact merge rule is $\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{y}_{\mathrm{add}}} = \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{1,\mathrm{add}}} + \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{2,\mathrm{add}}}$.

Gaussian Markov cross-covariance. For a jointly Gaussian Markov chain $\mathbf{v}_{1,\mathrm{mc}} \to \mathbf{v}_{2,\mathrm{mc}} \to \mathbf{v}_{3,\mathrm{mc}}$, the closure stores $\Sigma_{\mathbf{v}_{1,\mathrm{mc}},\mathbf{v}_{3,\mathrm{mc}}} = \Sigma_{\mathbf{v}_{1,\mathrm{mc}},\mathbf{v}_{2,\mathrm{mc}}} \Sigma^{-1}_{\mathbf{v}_{2,\mathrm{mc}}} \Sigma_{\mathbf{v}_{2,\mathrm{mc}},\mathbf{v}_{3,\mathrm{mc}}}$. We denote this computation by GAUSSIANMARKOVCLOSURE$(\Sigma_{\mathbf{v}_{1,\mathrm{mc}},\mathbf{v}_{2,\mathrm{mc}}}, \Sigma_{\mathbf{v}_{2,\mathrm{mc}}}, \Sigma_{\mathbf{v}_{2,\mathrm{mc}},\mathbf{v}_{3,\mathrm{mc}}})$.

4.2 BLOCKWISE FORWARD MOMENT PROPAGATION

A finite acyclic neural-network computation graph is evaluated in topological order by composing the local rules above. A block collects nodes between two retained graph boundaries and may contain branches, merges, and shared ancestors. Let $\mathbf{r}_b \in \mathbb{R}^{d_b}$ be the vector output of block b, and let $\mathcal{I}_b$ index the stochastic parameters and latent variables assigned to that block; $\boldsymbol{\nu}_s$ denotes the variable at site $s \in \mathcal{I}_b$.

For deterministic input $\mathbf{x}_k$ and parameter moments $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$, the graph-wide forward interface is

$$\begin{aligned} &\textsc{NetworkForwardMoments}(\mathbf{x}_k, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \\ &\quad\longrightarrow (\mu^{\mathrm{pred}}_{\mathbf{y}_k}, \Sigma^{\mathrm{pred}}_{\mathbf{y}_k}, \mathcal{C}^{\mathrm{fwd}}_k), \end{aligned} \tag{12}$$

where $\mathcal{C}^{\mathrm{fwd}}_k$ stores the node moments, the block-output moments, and the retained cross-covariances indexed by graph location. In particular, for every $s \in \mathcal{I}_b$, it contains $\mu_{\boldsymbol{\nu}_s}$, $\Sigma_{\boldsymbol{\nu}_s}$, $\mu_{\mathbf{r}_b}$, $\Sigma_{\mathbf{r}_b}$, and $\Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b}$. Appendix D gives a transformer implementation of this interface.

4.3 LAYER-SPECIFIC LEARN TO ADAPT RULES

We construct the backward pass using a block decomposition with the following assumptions.

Assumption 1 (Block decomposition)**.** The computation graph admits a finite acyclic block decomposition with blocks $b = 1, \ldots, N_{\mathrm{blk}}$ indexed in topological order and a unique terminal block satisfying $\mathbf{r}_{N_{\mathrm{blk}}} = \mathbf{y}_k$. Blocks may contain branches and merges. For each block, $\mathbf{r}_b$ contains every nonredundant stochastic coordinate crossing its downstream boundary exactly once.

Assumption 2 (Site ownership and separation)**.** The sets $\{\mathcal{I}_b\}_{b=1}^{N_{\mathrm{blk}}}$ are pairwise disjoint, and every stochastic parameter or latent vector other than $\mathbf{y}_k$ has exactly one site in their union. Parameters are confined to their assigned blocks, and the retained parameter law factorizes across parameter sites. The site representing $\mathbf{r}_b$ does not belong to $\mathcal{I}_b$; for $b < N_{\mathrm{blk}}$, it is an input-boundary site of exactly one downstream block. Every directed path from a site in $\mathcal{I}_b$ to $\mathbf{y}_k$ crosses $\mathbf{r}_b$, and, under the retained joint-Gaussian approximation, $\boldsymbol{\nu}_s$ is conditionally independent of downstream evidence given $\mathbf{r}_b$ for every $s \in \mathcal{I}_b$.

For $s \in \mathcal{I}_b$, let $d_s = \dim(\boldsymbol{\nu}_s)$. The forward cache contains $\mu_{\boldsymbol{\nu}_s}$, $\Sigma_{\boldsymbol{\nu}_s}$, and $\Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b}$. The canonical block-output gain is

$$\mathbf{G}^0_s = \Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b} \Sigma^{-1}_{\mathbf{r}_b}, \qquad s \in \mathcal{I}_b. \tag{13}$$

For target posterior moments $(\mu^+_{\mathbf{r}_b}, \Sigma^+_{\mathbf{r}_b})$, define

$$\Delta\mu_b = \mu^+_{\mathbf{r}_b} - \mu_{\mathbf{r}_b}. \tag{14}$$

Each site has a learned gain modifier $\mathbf{L}_s \in \mathbb{R}^{d_s \times d_s}$. The modified gain is

$$\mathbf{G}_s = \mathbf{L}_s \mathbf{G}^0_s. \tag{15}$$

The corresponding moment transport preserves $\boldsymbol{\nu}_s - \mathbf{G}_s \mathbf{r}_b$ while replacing $\mathbf{r}_b$ by an independent draw with the target moments. It gives

$$\mu^+_{\boldsymbol{\nu}_s} = \mu_{\boldsymbol{\nu}_s} + \mathbf{G}_s \Delta\mu_b, \tag{16}$$

$$\Sigma^+_{\boldsymbol{\nu}_s} = \Sigma_{\boldsymbol{\nu}_s} - \mathbf{G}_s \Sigma_{\mathbf{r}_b,\boldsymbol{\nu}_s} - \Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b} \mathbf{G}_s^\top + \mathbf{G}_s \left(\Sigma_{\mathbf{r}_b} + \Sigma^+_{\mathbf{r}_b}\right) \mathbf{G}_s^\top. \tag{17}$$

The updated marginals of a block's input boundary sites provide the target moments for its predecessor blocks. This reverse-topological recursion gives each site one block-output update. For a block target law $\mathcal{Q}^{\mathrm{tar}}_{\mathrm{blk},b}$, (16)–(17) with $\mathbf{L}_s = \mathbf{I}_{d_s}$ give the marginal moments obtained by integrating $p(d\boldsymbol{\nu}_s \mid \mathbf{r}_b)$ against $\mathcal{Q}^{\mathrm{tar}}_{\mathrm{blk},b}$ for each $s \in \mathcal{I}_b$. Only site marginals are retained; cross-site posterior covariances are discarded. The updated parameter-site means and covariance blocks form $(\mu^+_{\boldsymbol{\omega}}, \Sigma^+_{\boldsymbol{\omega}})$ under the retained factorization.

The covariance update can be written as

$$\Sigma^{+}_{\boldsymbol{\nu}_s} = \big(\Sigma_{\boldsymbol{\nu}_s} - \mathbf{G}^0_s \Sigma_{\mathbf{r}_b} (\mathbf{G}^0_s)^\top\big) + (\mathbf{G}_s - \mathbf{G}^0_s)\Sigma_{\mathbf{r}_b}(\mathbf{G}_s - \mathbf{G}^0_s)^\top + \mathbf{G}_s \Sigma^{+}_{\mathbf{r}_b} \mathbf{G}_s^\top \succeq \mathbf{0}_{d_s \times d_s}. \tag{18}$$

Let $\mathcal{I} = \dot{\bigcup}_{b=1}^{N_{\mathrm{blk}}} \mathcal{I}_b$ and partition it into N_{grp} site groups $\mathcal{I} = \dot{\bigcup}_{g=1}^{N_{\mathrm{grp}}} \mathcal{I}_g$ indexed from input to output. Each $\mathcal{I}_b$ is contained in one group. If block a is an ancestor of block b, the group containing $\mathcal{I}_a$ has an index no larger than the group containing $\mathcal{I}_b$; blocks at the same topological level may share a group. The student–expert moment discrepancy is

$$\mathcal{L}_{\exp}\big(\mu_{\boldsymbol{\nu}_s}, \Sigma_{\boldsymbol{\nu}_s}; \mu^{\exp}_{\boldsymbol{\nu}_s}, \Sigma^{\exp}_{\boldsymbol{\nu}_s}\big) = \frac{1}{d_s} \left\| (\mathbf{D}^{\exp}_s)^{-1/2} (\mu_{\boldsymbol{\nu}_s} - \mu^{\exp}_{\boldsymbol{\nu}_s}) \right\|_2^2 + \frac{\lambda_{\exp}}{d_s^2} \left\| (\mathbf{D}^{\exp}_s)^{-1/2} (\Sigma_{\boldsymbol{\nu}_s} - \Sigma^{\exp}_{\boldsymbol{\nu}_s}) (\mathbf{D}^{\exp}_s)^{-1/2} \right\|_F^2, \tag{19}$$

$$\mathbf{D}^{\exp}_s = \mathrm{Diag}\big(\mathrm{diag}(\Sigma^{\exp}_{\boldsymbol{\nu}_s})\big), \qquad \lambda_{\exp} > 0.$$

Here diag extracts the diagonal of a matrix, and Diag forms a diagonal matrix from a vector.

We define the expert update by output replacement. At update k, TARGETLAW returns a normalized target law $\mathcal{Q}^{\mathrm{tar}}_k$ and its first two moments. For a moments-only target, $\mathcal{Q}^{\mathrm{tar}}_k$ is the Gaussian measure with those moments and a positive-definite covariance. The expert predictive law and output-replacement factor are

$$\mathcal{P}^{\mathrm{pred,exp}}_k(d\mathbf{y}) = \int \mathcal{P}_{\mathrm{out}}(d\mathbf{y} \mid \mathbf{x}_k, \boldsymbol{\omega}) p^{\exp}_{k-1}(\boldsymbol{\omega})\, d\boldsymbol{\omega}, \tag{20}$$

$$\Lambda^{\exp}_k(\boldsymbol{\omega}) = \int \frac{d\mathcal{Q}^{\mathrm{tar}}_k}{d\mathcal{P}^{\mathrm{pred,exp}}_k}(\mathbf{y}) \mathcal{P}_{\mathrm{out}}(d\mathbf{y} \mid \mathbf{x}_k, \boldsymbol{\omega}), \qquad \mathcal{Q}^{\mathrm{tar}}_k \ll \mathcal{P}^{\mathrm{pred,exp}}_k. \tag{21}$$

Here $\ll$ denotes absolute continuity. An observation-likelihood update replaces $\Lambda^{\exp}_k$ with the supplied likelihood. When a target depends on current student predictive moments, as in (28), the student and independently evolving expert receive the same realized law. That law and the expert moments are held fixed while updating the modifier matrices.

The graph-wide NETWORKBACKWARDMOMENTS procedure traverses the cached blocks in reverse topological order, applies the corresponding block routines, and passes each updated input-boundary marginal to its predecessor block. Given a requested site set $\mathcal{I}_{\mathrm{req}} \subseteq \mathcal{I}$, its interface is

$$\begin{aligned} &\textsc{NetworkBackwardMoments}(\mu^{\mathrm{tar}}_{\mathbf{y}_k}, \Sigma^{\mathrm{tar}}_{\mathbf{y}_k}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}, \mathcal{C}^{\mathrm{fwd}}_k, \{\mathbf{L}_s\}_{s\in\mathcal{I}}; \mathcal{I}_{\mathrm{req}}) \\ &\quad \longrightarrow \big(\mu^{+}_{\boldsymbol{\omega}}, \Sigma^{+}_{\boldsymbol{\omega}}, \{\mu^{+}_{\boldsymbol{\nu}_s}, \Sigma^{+}_{\boldsymbol{\nu}_s}\}_{s\in\mathcal{I}_{\mathrm{req}}}\big). \end{aligned} \tag{22}$$

For $\mathcal{I}_{\mathrm{req}} = \emptyset$, the empty third component is omitted. The procedure uses canonical gains when every $\mathbf{L}_s = \mathbf{I}_{d_s}$.

Let p_{task} denote the training task distribution. Each task object $\mathcal{T}_e$ supplies a prior and an ordered adaptation sequence. Given expert state $\mathcal{C}^{\exp}_{k-1}$, input $\mathbf{x}_k$, and $\mathcal{Q}^{\mathrm{tar}}_k$, the abstract EXPERTUPDATE applies the selected observation or replacement factor and returns $\mathcal{C}^{\exp}_k$ together with $\{(\mu^{\exp,+}_{\boldsymbol{\nu}_{s,k}}, \Sigma^{\exp,+}_{\boldsymbol{\nu}_{s,k}})\}_{s\in\mathcal{I}}$. For $\mathcal{J} \subseteq \mathcal{I}$, let $\mathcal{L}_{\mathcal{J}}(\mathcal{T}_e)$ be the record-averaged sum of (19) over $s \in \mathcal{J}$ after initializing the student and expert from the task prior and processing the records in order with the routines above.

Algorithm 1 Learning to Adapt

Require: p_{task}, $\{\mathcal{I}_g\}_{g=1}^{N_{\mathrm{grp}}}$, TARGETLAW, and EXPERTUPDATE
1: Initialize $\mathbf{L}_s \leftarrow \mathbf{I}_{d_s}$ for every $s \in \mathcal{I}$
2: **for** $g = N_{\mathrm{grp}}, N_{\mathrm{grp}} - 1, \ldots, 1$ **do**
3: Perform gradient steps on $\{\mathbf{L}_s : s \in \mathcal{I}_g\}$ using $\mathbb{E}_{\mathcal{T}_e \sim p_{\mathrm{task}}}[\mathcal{L}_{\mathcal{I}_g}(\mathcal{T}_e)]$
4: **end for**
5: Perform joint gradient steps on $\{\mathbf{L}_s\}_{s\in\mathcal{I}}$ using $\mathbb{E}_{\mathcal{T}_e \sim p_{\mathrm{task}}}[\mathcal{L}_{\mathcal{I}}(\mathcal{T}_e)]$
6: **return** $\{\mathbf{L}_s\}_{s\in\mathcal{I}}$

The output-to-input stages initialize the subsequent joint refinement; joint gradient steps account for coupling across groups induced by sequential task states. Expert moments are used during training; deployment freezes the learned modifiers and uses deterministic moment passes. Appendix E instantiates the expert with full-history sequential Monte Carlo (SMC).

5 Optimization and Learning to Optimize beyond Data Distribution

In this section, we extend Bayesian adaptation from fitting observed behavior to improving it using reward information. The BNN's predictive distribution represents uncertainty over candidate actions, while value information reweights it toward actions with higher expected return. The resulting target moments can be fed into the backward pass described in Section 4, allowing imitation and reward-guided improvement to share the same adaptation mechanism.

Section 5.1 formulates a KL-regularized policy improvement objective and characterizes its reward-tilted target distribution. Section 5.2 proposes to learn a task-dependent temperature from adaptation histories, thereby tuning the balance between preserving reference behavior and pursuing higher return.

5.1 Optimization beyond Data Distribution

Consider a Markov decision process (MDP) with state $\mathbf{s}_t$, action $\mathbf{a}_t \in \mathbb{R}^{d_a}$, transition density $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, reward $r(\mathbf{s}_t, \mathbf{a}_t)$, initial density $p(\mathbf{s}_0)$, and discount $\gamma_{\mathrm{R}} \in (0,1)$. Policies have densities with respect to Lebesgue measure on $\mathbb{R}^{d_a}$. Let the policy predictor be a block-graph network of the form described in Section 4. At time t, it receives a deterministic input $\mathbf{x}_t$ constructed from the information available through time t, including $\mathbf{s}_t$, and has action output $\mathbf{y}_t = \mathbf{a}_t$. A candidate student parameter law $p^{\mathrm{stu}}(\boldsymbol{\omega})$ induces the policy kernel

$$\mathcal{P}^{\mathrm{stu}}(d\mathbf{a} \mid \mathbf{x}) = \int_{\Omega_\omega} \mathcal{P}_{\mathrm{out}}(d\mathbf{a} \mid \mathbf{x}, \boldsymbol{\omega}) p^{\mathrm{stu}}(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \tag{23}$$

We denote its density by $\pi^{\mathrm{stu}}(\mathbf{a} \mid \mathbf{x})$. The reference policy is Markov, $\pi_{\mathrm{ref}}(\mathbf{a} \mid \mathbf{s})$. We seek high return while penalizing departure from this reference along the induced trajectory law:

$$\max_{p^{\mathrm{stu}}(\boldsymbol{\omega})} \quad \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma_{\mathrm{R}}^t \left(r(\mathbf{s}_t, \mathbf{a}_t) - \beta_{\mathrm{R}} D_{\mathrm{KL}}\big[\pi^{\mathrm{stu}}(\cdot \mid \mathbf{x}_t) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid \mathbf{s}_t)\big]\right)\right], \tag{24}$$

where $\beta_{\mathrm{R}} > 0$, D_{KL} denotes the Kullback–Leibler (KL) divergence, and the expectation is over the trajectory induced by $p(\mathbf{s}_0)$, the dynamics, the deterministic construction of $\mathbf{x}_t$, and π^{stu}.

We use the statewise target

$$q^{\mathrm{tar}}_{\beta_{\mathrm{R}}}(\mathbf{a} \mid \mathbf{s}) \propto \pi_{\mathrm{ref}}(\mathbf{a} \mid \mathbf{s}) \exp\left(\frac{\mathsf{Adv}^{\pi_{\mathrm{ref}}}(\mathbf{s}, \mathbf{a})}{\beta_{\mathrm{R}}}\right), \tag{25}$$
$$\mathsf{Adv}^{\pi_{\mathrm{ref}}}(\mathbf{s}, \mathbf{a}) = \mathsf{Q}^{\pi_{\mathrm{ref}}}(\mathbf{s}, \mathbf{a}) - \mathsf{V}^{\pi_{\mathrm{ref}}}(\mathbf{s}).$$

At fixed $\mathbf{s}$, the state-value term cancels from the normalized density. The tilt preserves the support of π_{ref} and therefore cannot introduce actions to which the reference policy assigns zero density. The task supplies either an evaluable reference-policy density or samples with normalized importance weights, together with pointwise estimates of $\mathsf{Q}^{\pi_{\mathrm{ref}}}$, obtained analytically or from logged data.

Theorem 1 (Statewise exponential tilt)**.** *Fix* $\mathbf{s}$ *and* $\beta_{\mathrm{R}} > 0$*. Suppose* $0 < Z_{\beta_{\mathrm{R}}}(\mathbf{s}) < \infty$ *and the advantage is integrable under* $q^{\mathrm{tar}}_{\beta_{\mathrm{R}}}$*. Then, up to* $\pi_{\mathrm{ref}}(\cdot \mid \mathbf{s})$*-null sets,* $q^{\mathrm{tar}}_{\beta_{\mathrm{R}}}$ *is the unique finite-objective maximizer of*

$$\max_{q} \quad \mathbb{E}_{\mathbf{a}\sim q}\left[\mathsf{Adv}^{\pi_{\mathrm{ref}}}(\mathbf{s}, \mathbf{a})\right] - \beta_{\mathrm{R}} D_{\mathrm{KL}}[q \,\|\, \pi_{\mathrm{ref}}(\cdot \mid \mathbf{s})] \tag{26}$$

over probability densities q *that vanish wherever* $\pi_{\mathrm{ref}}(\cdot \mid \mathbf{s})$ *vanishes.*

The result follows by writing the objective as $\beta_{\mathrm{R}} \log Z_{\beta_{\mathrm{R}}}(\mathbf{s}) - \beta_{\mathrm{R}} D_{\mathrm{KL}}[q \| q^{\mathrm{tar}}_{\beta_{\mathrm{R}}}]$. This variational form underlies several KL-regularized policy-improvement methods (Peters et al., 2010; Peng et al., 2019; Frans et al., 2025). Equation (25) gives a one-step improvement target. We provide an implementation in Algorithm 11 in Appendix F.

5.2 Learning to Optimize beyond Data Distribution

The temperature β_{R} controls the reference-policy tradeoff. We learn the positive task-conditioned rule

$$\beta_{\mathrm{R}} = \mathrm{Temp}(\boldsymbol{\phi}; \boldsymbol{\xi}), \qquad \mathrm{Temp}(\boldsymbol{\phi}; \boldsymbol{\xi}) > 0, \tag{27}$$

where $\boldsymbol{\phi}$ is the task descriptor and $\boldsymbol{\xi}$ the parameter vector for the learning process.

Let $\mathcal{T}_e \sim p_{\mathrm{task}}$. A task specifies the MDP, reference policy, value estimates, prior moments $(\mu^{\mathrm{prior}}_{\boldsymbol{\omega},e}, \Sigma^{\mathrm{prior}}_{\boldsymbol{\omega},e})$, ordered training records

$$\mathcal{D}_{\mathrm{tr},e} = [(\mathbf{x}_{e,k}, \mathbf{s}_{e,k}, \mathbf{a}_{\log,e,k})]_{k=1}^{N_{\mathrm{tr},e}},$$

and validation input–state records

$$\mathcal{D}_{\mathrm{val},e} = [(\mathbf{x}_{e,k}, \mathbf{s}_{e,k})]_{k=N_{\mathrm{tr},e}+1}^{N_{\mathrm{tr},e}+N_{\mathrm{val},e}}.$$

Here $\mathbf{x}_{e,k}$ is the policy-network input associated with $\mathbf{s}_{e,k}$, and $\mathbf{a}_{\log,e,k}$ is the logged action. Adaptation targets the reward-tilted action distribution. Logged actions can also enter $\boldsymbol{\phi}_e$ or the estimate of the reference policy. The descriptor $\boldsymbol{\phi}_e$ is computed from $\mathcal{T}_e$ and $\mathcal{D}_{\mathrm{tr},e}$. We suppress the task index on the reference policy and value functions.

At update k, NetworkForwardMoments returns $(\mu^{\mathrm{pred}}_{\mathbf{a}_{e,k}}, \Sigma^{\mathrm{pred}}_{\mathbf{a}_{e,k}}, C^{\mathrm{fwd}}_{e,k})$ from $(\mathbf{x}_{e,k}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$. We tilt the resulting Gaussian approximation to the predictive action law:

$$\widetilde{\mathcal{Q}}^{\mathrm{tar}}_{\beta_{\mathrm{R}},e,k}(d\mathbf{a} \mid \mathbf{x}_{e,k}, \mathbf{s}_{e,k}) \propto \exp\left(\frac{\mathrm{Q}^{\pi_{\mathrm{ref}}}(\mathbf{s}_{e,k}, \mathbf{a})}{\beta_{\mathrm{R}}}\right) \mathcal{N}\left(d\mathbf{a}; \mu^{\mathrm{pred}}_{\mathbf{a}_{e,k}}, \Sigma^{\mathrm{pred}}_{\mathbf{a}_{e,k}}\right). \tag{28}$$

A differentiable integration rule computes the normalized law's first two moments. They form the target for NetworkBackwardMoments, and the updated parameter moments initialize the next record. The student moments reset to $(\mu^{\mathrm{prior}}_{\boldsymbol{\omega},e}, \Sigma^{\mathrm{prior}}_{\boldsymbol{\omega},e})$ for each task. Let $\mu^{\mathrm{pred}}_{\mathbf{a}}(\mathbf{x} \mid \mathcal{T}_e, \beta_{\mathrm{R}}, \mathcal{D}_{\mathrm{tr},e})$ be the predictive mean after the ordered training updates. We optimize

$$\min_{\boldsymbol{\xi}} \quad \mathbb{E}_{\mathcal{T}_e \sim p_{\mathrm{task}}}\left[-\frac{1}{N_{\mathrm{val},e}} \sum_{(\mathbf{x}_{e,k}, \mathbf{s}_{e,k}) \in \mathcal{D}_{\mathrm{val},e}} \mathrm{Q}^{\pi_{\mathrm{ref}}}\left(\mathbf{s}_{e,k}, \mu^{\mathrm{pred}}_{\mathbf{a}}(\mathbf{x}_{e,k} \mid \mathcal{T}_e, \mathrm{Temp}(\boldsymbol{\phi}_e; \boldsymbol{\xi}), \mathcal{D}_{\mathrm{tr},e})\right)\right]. \tag{29}$$

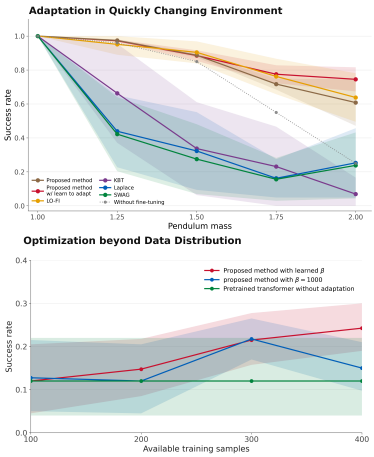
We provide an implementation in Algorithm 12 in Appendix F, and an example task descriptor in Appendix F.1.

6 Numerical Simulations

The experiments examine the two deployed components of the framework. The first evaluates sequential decision-transformer adaptation enabled by the moment-propagation machinery in Section 4 and its transformer instantiation in Appendix D; the second evaluates the reward-guided target distribution in Section 5. The first compares canonical backward gains with expert-trained final-action-layer gain modification, motivated by Algorithm 1. The second compares fixed and task-conditioned temperatures. End-to-end evaluation of the expert-guided calibration in Algorithm 1 and the task-conditioned temperature in Section 5.2 is left for future work.

6.1 Sequential Transformer Adaptation under Environment Drift

We consider a decision transformer (Chen et al., 2021) pretrained at pendulum mass 1 that receives one logged adaptation window per update as the mass increases by 0.01 to 2. We compare canonical Bayesian adaptation with learned final-action-layer gain modification, alongside LO-FI (Chang et al., 2023), KBT (Jing et al., 2025), Laplace, SWAG (Onal et al., 2024), and no fine-tuning. Figure 1a reports upright-maintenance success across ten adaptation streams, using shared random initial states. Controllers remain fixed during evaluation episodes. Detailed settings are given in Appendix G.



(a) Sequential adaptation under gradual pendulum-mass drift: the proposed method updates after every sample, with or without an expert-trained modifier for its final action-layer gain; LO-FI updates sequentially, the Kalman Bayesian transformer adapts only its final module, Laplace and SWAG refit on batches containing data from different environments, and the model without fine-tuning retains its pretrained weights.

(b) Reward-guided adaptation from suboptimal data. We compare a meta-learned temperature with a fixed temperature for the shifted distribution in (25), using the pretrained transformer without adaptation as a baseline.

Figure 1: Decision-transformer adaptation results. Where across-seed variability is present, shaded regions denote 95% confidence intervals for the mean over ten random seeds.

6.2 Reward-Guided Adaptation from Suboptimal Data

We next fix the pendulum mass at 2 while retaining a decision transformer pretrained at mass 1. The available adaptation data were generated at mass 1 and are therefore suboptimal for the target environment. We additionally provide grid-based action-value estimates for upright occupancy under optimal continuation. We compare three methods across training-set sizes: the proposed method with meta-learned action tilting, the same method with fixed normalized temperature $\beta = 1000$, and the pretrained transformer without adaptation. Action tilting is based on the shifted target in (25). Figure 1b reports the resulting success rates; detailed settings are given in Appendix G.

7 Conclusion

We developed a data-driven framework for learning how Bayesian neural networks adapt from limited, sequentially arriving, and potentially suboptimal data. By expressing adaptation as layerwise posterior updates, the framework learns backward update rules from expert posterior moments, providing supervision beyond input–output matching. Extending moment propagation to attention, normalization, feedforward, residual, and branching operations makes this approach applicable to transformers and other complex architectures while retaining one forward and one backward moment pass per sample at deployment.

We further integrated reward-guided output targets into the same Bayesian update mechanism, using predictive uncertainty together with value information to improve upon observed behavior. Learning both layer-specific gains and a task-dependent imitation–reinforcement balance allows the adaptation algorithm to improve through experience across tasks. Numerical simulations demonstrate the benefits of these components for sample efficiency, control performance, and robustness under environmental changes and suboptimal data.

AI Usage Statement

We used large language models to assist with drafting and polishing the manuscript, finding and organizing related work, and coding of the numerical simulations.

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1ANxQW0b.

Hayk Amirkhanian, Markus Walker, Uwe D. Hanebeck, and Marco F. Huber. Bridging Bayesian inference and neural network training: Equivalence of KBNN and statistical linearization. In *Proceedings of the 28th International Conference on Information Fusion (FUSION 2025)*, Rio de Janeiro, Brazil, July 2025. doi: 10.23919/FUSION65864.2025.11124135.

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. Meta-learning with adaptive hyperparameters. *Advances in neural information processing systems*, 33: 20755–20765, 2020.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622. PMLR, 2015. URL https://proceedings.mlr.press/v37/blundell15.html.

Peter G. Chang, Gerardo Durán-Martín, Alex Shestopaloff, Matt Jones, and Kevin Patrick Murphy. Low-rank extended Kalman filtering for online learning of neural networks from streaming data. In *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pp. 1025–1071. PMLR, 2023. URL https://proceedings.mlr.press/v232/chang23a.html.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems*, 34:15084–15097, 2021.

Tianqi Chen, Emily B. Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1683–1691. PMLR, 2014.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31, pp. 4759–4770, 2018.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux – effortless Bayesian deep learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20089–20103. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a7c9585703d275249f30a088cebba0ad-Abstract.html.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006. doi: 10.1111/j.1467-9868.2006.00553.x.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

Xinjie Fan, Shujian Zhang, Bo Chen, and Mingyuan Zhou. Bayesian attention modules. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/bcff3f632fd16ff099a49c2f0932b47a-Abstract.html.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1126–1135, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator, 2025. URL https://arxiv.org/abs/2505.23458.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059. PMLR, 2016. URL https://proceedings.mlr.press/v48/gal16.html.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 8(5–6):359–483, 2015. doi: 10.1561/2200000049.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, volume 24, pp. 2348–2356. Curran Associates, Inc., 2011.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, volume 31, pp. 5302–5311, 2018.

Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(9):5149–5169, 2022.

Yunhun Jang, Hankook Lee, Sung Ju Hwang, and Jinwoo Shin. Learning what and where to transfer. In *International conference on machine learning*, pp. 3030–3039. PMLR, 2019.

Haoming Jing, Oren Wright, José M. F. Moura, and Yorie Nakahira. Kalman Bayesian transformer, 2025. URL https://arxiv.org/abs/2509.10695.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6402–6413. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper/7219-simple-and-scalable-predictive-uncertainty-estimation-using-deep-ensembles.

David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992. doi: 10.1162/neco.1992.4.3.448.

Wesley J. Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, volume 32, pp. 13153–13164. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/9472-a-simple-baseline-for-bayesian-uncertainty-in-deep-learning.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets, 2020. URL https://arxiv.org/abs/2006.09359.

Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, New York, 1996. doi: 10.1007/978-1-4612-0745-0.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Emre Onal, Klemens Flöge, Emma Caldwell, Arsen Sheverdin, and Vincent Fortuin. Gaussian stochastic weight averaging for Bayesian low-rank adaptation of large language models. In *AABI 2024 Symposium*, 2024. URL https://arxiv.org/abs/2405.03425.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, volume 29, pp. 4026–4034, 2016.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pp. 1607–1612. AAAI Press, 2010. doi: 10.1609/aaai.v24i1.7727.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 5331–5340, 2019.

Herbert E Rauch, Fong Tung, and Charlotte T Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965.

Sachin Ravi and Alex Beatson. Amortized Bayesian meta-learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rkgpy3C5tX.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable Laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

Karthik Abinav Sankararaman, Sinong Wang, and Han Fang. BayesFormer: Transformer with uncertainty estimation, 2022. URL https://arxiv.org/abs/2206.00826.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1842–1850, 2016.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 4077–4087, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper/7181-attention-is-all-you-need.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, volume 29, pp. 3630–3638, 2016.

Philipp Wagner, Xinyang Wu, and Marco F. Huber. Kalman Bayesian neural networks for closed-form online learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10069–10077, 2023. doi: 10.1609/aaai.v37i8.26200. URL https://ojs.aaai.org/index.php/AAAI/article/view/26200.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. Learning to reinforcement learn. *CoRR*, abs/1611.05763, 2016.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 681–688, 2011. URL https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, volume 31, pp. 7332–7342, 2018.

A Related Work

Bayesian neural networks and approximate posterior inference. Bayesian neural networks (BNNs) replace point estimates of network weights with posterior distributions. Posterior prediction represents epistemic uncertainty and incorporates prior information, which is useful with limited or noisy data and under nonstationarity or distribution shift. Early work established weight-space priors, posterior averaging, and uncertainty-aware prediction (MacKay, 1992; Neal, 1996). The intractability of exact inference in modern networks has motivated a broad range of approximations.

Variational inference methods approximate the posterior with a tractable parametric family and optimize an evidence lower bound or variational free-energy objective (Graves, 2011; Blundell et al., 2015). Sampling-based approaches draw approximate posterior samples using Markov chain Monte Carlo or stochastic-gradient MCMC methods (Neal, 1996; Welling & Teh, 2011; Chen et al., 2014). Dropout-based methods reinterpret dropout training as approximate Bayesian inference and estimate uncertainty through stochastic test-time forward passes (Gal & Ghahramani, 2016). Laplace approximations fit a local Gaussian posterior around a maximum a posteriori solution using curvature information, with modern variants introducing scalable diagonal, Kronecker-factored, last-layer, or subnetwork approximations (MacKay, 1992; Ritter et al., 2018; Daxberger et al., 2021). Other scalable approximations include SWA-Gaussian (SWAG), which fits a Gaussian distribution to stochastic-gradient trajectories near convergence, and deep ensembles, which approximate uncertainty by aggregating independently trained networks (Maddox et al., 2019; Lakshminarayanan et al., 2017; Osband et al., 2016).

Most of these methods are not designed for online adaptation: variational and sampling methods require iterative updates, dropout and ensembles require repeated evaluation, and Laplace and SWAG are commonly post hoc. Kalman-style updates instead revise posterior moments sequentially without gradient-based retraining.

Kalman-smoothing-based neural network learning. Kalman Bayesian neural networks (KBNNs) exploit a network's layerwise Markov structure by formulating adaptation as a smoothing problem along its computational pathway. A forward pass propagates Gaussian moments through the network, while a backward smoothing pass uses the observed output to update posterior moments of preceding latent activations and weights. This combination enables sequential Gaussian moment approximations without conventional gradient-based retraining (Wagner et al., 2023).

Existing KBNNs mainly address chain-structured feed-forward architectures, use analytically specified gains, and optimize supervised likelihood objectives. Kalman Bayesian transformer variants typically adapt a final subnetwork while freezing the remaining architecture (Jing et al., 2025). Relative to existing KBNN extensions (Amirkhanian et al., 2025), our framework combines graph-wide moments, learned matrix-valued gain modifiers, and reward-guided targets.

Bayesian learning in transformers. Transformer Bayesian inference must propagate uncertainty through residual branches, normalization, masked softmax attention, and aggregation. BayesFormer extends dropout-based variational inference to transformer components and evaluates it on language modeling, classification, long-sequence modeling, translation, and active learning (Sankararaman et al., 2022). Bayesian Attention Modules learn stochastic attention distributions with data-dependent priors for graph, vision-language, and language models (Fan et al., 2020). These methods place distributions over selected modules, whereas ours targets branch-aware, graph-wide sequential Gaussian updates.

Reward-weighted learning, KL-regularized policy optimization, and Bayesian reinforcement learning. Supervised BNN and KBNN objectives fit the observed distribution. In control, that distribution may come from suboptimal demonstrations, imperfect controllers, or behavior policies, so direct fitting can preserve low-quality behavior.

A related line of work in reinforcement learning formulates policy improvement through KL-regularized or reward-weighted objectives. Relative entropy policy search constrains policy updates by a KL divergence from a reference distribution while favoring high-reward samples (Peters et al., 2010). Maximum a posteriori policy optimization and advantage-weighted regression similarly connect policy improvement with weighted maximum-likelihood updates, where samples with higher

estimated advantage or reward receive greater weight (Abdolmaleki et al., 2018; Peng et al., 2019). Advantage-weighted actor-critic combines offline data with online reinforcement learning while controlling deviation from the data distribution (Nair et al., 2020). Related ideas appear in guided generative modeling and diffusion-based optimization, where reward or value information biases a model toward more desirable outputs (Frans et al., 2025).

Bayesian reinforcement learning maintains posterior uncertainty over dynamics, rewards, value functions, or policies to support exploration and robust control (Ghavamzadeh et al., 2015; Osband et al., 2016; Chua et al., 2018). Here, a statewise KL-regularized policy-improvement target serves as output evidence for the sequential Gaussian update, allowing the target to depart from logged behavior.

Meta-learning and learned adaptation rules. Meta-learning aims to learn mechanisms that enable fast adaptation across tasks (Hospedales et al., 2022). In supervised learning, gradient-based meta-learning methods such as Model-Agnostic Meta-Learning and first-order variants learn initial parameters that can be rapidly adapted to new tasks through a small number of gradient steps (Finn et al., 2017; Nichol et al., 2018). Metric-based methods, including Matching Networks and Prototypical Networks, learn representation spaces in which new examples can be classified by similarity to stored or task-specific prototypes (Vinyals et al., 2016; Snell et al., 2017). Memory-augmented and black-box meta-learning methods learn recurrent update mechanisms or learned optimizers that internalize adaptation dynamics (Santoro et al., 2016; Andrychowicz et al., 2016; Ravi & Larochelle, 2017).

Meta-learning has also been applied to reinforcement learning, where agents must adapt online to changing tasks or environments. Recurrent meta-RL methods use experience-dependent hidden states to support adaptation, while gradient-based and latent-context methods learn policies or representations that can be efficiently updated from new trajectories (Wang et al., 2016; Duan et al., 2016; Finn et al., 2017; Gupta et al., 2018; Rakelly et al., 2019). Bayesian and probabilistic meta-learning methods further extend this perspective by learning distributions over parameters, priors, or amortized inference procedures across tasks (Finn et al., 2018; Yoon et al., 2018; Ravi & Beatson, 2019).

These approaches meta-learn initializations, priors, embeddings, recurrent adaptation mechanisms, or gradient-based update rules. Our method instead calibrates sitewise matrix-valued gain modifiers against expert posterior moments and separately meta-learns the reward-tilt temperature.

B Detailed Moment-Propagation Derivations

For a matrix $\mathbf{C}$, $\mathbf{C}[:, i]$ denotes its ith column.

B.1 Bilinear Matrix Node

Consider the directed structure $\mathbf{X}_0 \to \mathbf{X}_1 \to \mathbf{X}_3$ and $\mathbf{X}_0 \to \mathbf{X}_2 \to \mathbf{X}_3$, where $\mathbf{X}_3 = \mathbf{X}_1\mathbf{X}_2$. Let $\mathbf{X}_0 \in \mathbb{R}^{d_{0,\mathrm{row}} \times d_{0,\mathrm{col}}}$, $\mathbf{X}_1 \in \mathbb{R}^{d_{\mathrm{row}} \times d_{\mathrm{mid}}}$, $\mathbf{X}_2 \in \mathbb{R}^{d_{\mathrm{mid}} \times d_{\mathrm{col}}}$, and $\mathbf{X}_3 \in \mathbb{R}^{d_{\mathrm{row}} \times d_{\mathrm{col}}}$, with $\mathbf{x}_\iota = \mathrm{vec}(\mathbf{X}_\iota)$ for $\iota \in \{0, 1, 2, 3\}$. The approximation assumes that $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ is jointly Gaussian with $\mathbf{x}_1 \perp \mathbf{x}_2 \mid \mathbf{x}_0$. For jointly Gaussian variables, conditional independence is equivalent to zero conditional covariance, so

$$\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2 \mid \mathbf{x}_0) = \Sigma_{\mathbf{x}_1,\mathbf{x}_2} - \Sigma_{\mathbf{x}_0,\mathbf{x}_1}^{\top} \Sigma_{\mathbf{x}_0}^{-1} \Sigma_{\mathbf{x}_0,\mathbf{x}_2} = \mathbf{0}_{(d_{\mathrm{row}} d_{\mathrm{mid}}) \times (d_{\mathrm{mid}} d_{\mathrm{col}})}, \tag{30}$$

which gives

$$\Sigma_{\mathbf{x}_1,\mathbf{x}_2} = \Sigma_{\mathbf{x}_0,\mathbf{x}_1}^{\top} \Sigma_{\mathbf{x}_0}^{-1} \Sigma_{\mathbf{x}_0,\mathbf{x}_2}. \tag{31}$$

Define the covariance-entry map $\kappa_{\iota,\iota'}$ for $\iota, \iota' \in \{0, 1, 2, 3\}$ by

$$\kappa_{\iota,\iota'}\big((i, j), (i', j')\big) = \mathrm{Cov}\left(\mathbf{X}_\iota[i, j], \mathbf{X}_{\iota'}[i', j']\right). \tag{32}$$

If $d_{\iota,\mathrm{row}}$ is the row count of $\mathbf{X}_\iota$, then column-major vectorization gives

$$\mathrm{ind}_\iota(i, j) = i + (j - 1) d_{\iota,\mathrm{row}}, \tag{33}$$

and therefore

$$\kappa_{\iota,\iota'}\big((i,j),(i',j')\big) = \Sigma_{\mathbf{x}_\iota,\mathbf{x}_{\iota'}}\left[\mathrm{ind}_\iota(i,j),\mathrm{ind}_{\iota'}(i',j')\right]. \tag{34}$$

In the sums below, ν and ν' index the shared matrix dimension.

The mean of $\mathbf{X}_3$ is computed entrywise as

$$\begin{aligned}\mu_{\mathbf{X}_3}[i,j] &= \mathbb{E}[\mathbf{X}_3[i,j]] \\ &= \sum_{\nu=1}^{d_{\mathrm{mid}}} \big(\mu_{\mathbf{X}_1}[i,\nu]\mu_{\mathbf{X}_2}[\nu,j] + \kappa_{1,2}\big((i,\nu),(\nu,j)\big)\big).\end{aligned} \tag{35}$$

Equivalently,

$$\mu_{\mathbf{X}_3} = \mu_{\mathbf{X}_1}\mu_{\mathbf{X}_2} + \mathbf{C}_{\mathrm{bil}}, \qquad \mathbf{C}_{\mathrm{bil}}[i,j] = \sum_{\nu=1}^{d_{\mathrm{mid}}} \kappa_{1,2}\big((i,\nu),(\nu,j)\big). \tag{36}$$

Expanding each product about its means and using the vanishing centered third-order moments of a jointly Gaussian vector gives the cross-covariance between $\mathbf{X}_0$ and $\mathbf{X}_3$:

$$\kappa_{0,3}\big((i,j),(i',j')\big) = \sum_{\nu=1}^{d_{\mathrm{mid}}} \left[\kappa_{0,1}\big((i,j),(i',\nu)\big)\mu_{\mathbf{X}_2}[\nu,j'] + \kappa_{0,2}\big((i,j),(\nu,j')\big)\mu_{\mathbf{X}_1}[i',\nu]\right]. \tag{37}$$

Similarly, the cross-covariance between $\mathbf{X}_1$ and $\mathbf{X}_3$ is

$$\kappa_{1,3}\big((i,j),(i',j')\big) = \sum_{\nu=1}^{d_{\mathrm{mid}}} \left[\kappa_{1,1}\big((i,j),(i',\nu)\big)\mu_{\mathbf{X}_2}[\nu,j'] + \kappa_{1,2}\big((i,j),(\nu,j')\big)\mu_{\mathbf{X}_1}[i',\nu]\right], \tag{38}$$

and the cross-covariance between $\mathbf{X}_2$ and $\mathbf{X}_3$ is

$$\kappa_{2,3}\big((i,j),(i',j')\big) = \sum_{\nu=1}^{d_{\mathrm{mid}}} \left[\kappa_{2,1}\big((i,j),(i',\nu)\big)\mu_{\mathbf{X}_2}[\nu,j'] + \kappa_{2,2}\big((i,j),(\nu,j')\big)\mu_{\mathbf{X}_1}[i',\nu]\right]. \tag{39}$$

The corresponding vectorized covariance matrices are obtained by applying the column-major index map:

$$\Sigma_{\mathbf{x}_\iota,\mathbf{x}_3}\left[\mathrm{ind}_\iota(i,j),\mathrm{ind}_3(i',j')\right] = \kappa_{\iota,3}\big((i,j),(i',j')\big), \qquad \iota \in \{0,1,2\}. \tag{40}$$

The same Gaussian product identity propagates a retained external vector $\mathbf{v}_{\mathrm{ext}}$ that is jointly Gaussian with $(\mathbf{x}_1,\mathbf{x}_2)$:

$$\begin{aligned}\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_3}[:,\mathrm{ind}_3(i,j)] = \sum_{\nu=1}^{d_{\mathrm{mid}}} \Big(&\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_1}[:,\mathrm{ind}_1(i,\nu)]\mu_{\mathbf{X}_2}[\nu,j] \\ &+ \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_2}[:,\mathrm{ind}_2(\nu,j)]\mu_{\mathbf{X}_1}[i,\nu]\Big).\end{aligned} \tag{41}$$

To compute $\Sigma_{\mathbf{x}_3}$, use Isserlis' theorem for fourth-order moments of jointly Gaussian variables. Since

$$\mathbf{X}_3[i,j] = \sum_{\nu=1}^{d_{\mathrm{mid}}} \mathbf{X}_1[i,\nu]\mathbf{X}_2[\nu,j], \tag{42}$$

we have

$$\mathrm{Cov}\left(\mathbf{X}_3[i,j],\mathbf{X}_3[i',j']\right) = \sum_{\nu=1}^{d_{\mathrm{mid}}}\sum_{\nu'=1}^{d_{\mathrm{mid}}} \mathrm{Cov}\left(\mathbf{X}_1[i,\nu]\mathbf{X}_2[\nu,j],\mathbf{X}_1[i',\nu']\mathbf{X}_2[\nu',j']\right). \tag{43}$$

For jointly Gaussian scalar random variables x,y,u,v,

$$\begin{aligned}\mathrm{Cov}(xy,uv) = &\,\mathbb{E}[x]\mathbb{E}[u]\,\mathrm{Cov}(y,v) + \mathbb{E}[x]\mathbb{E}[v]\,\mathrm{Cov}(y,u) \\ &+ \mathbb{E}[y]\mathbb{E}[u]\,\mathrm{Cov}(x,v) + \mathbb{E}[y]\mathbb{E}[v]\,\mathrm{Cov}(x,u) \\ &+ \mathrm{Cov}(x,u)\,\mathrm{Cov}(y,v) + \mathrm{Cov}(x,v)\,\mathrm{Cov}(y,u).\end{aligned} \tag{44}$$

Applying (44) gives

$$\begin{aligned}\kappa_{3,3}\big((i,j),(i',j')\big) = \sum_{\nu=1}^{d_{\mathrm{mid}}}\sum_{\nu'=1}^{d_{\mathrm{mid}}}\Big[&\mu_{\mathbf{X}_1}[i,\nu]\mu_{\mathbf{X}_1}[i',\nu']\kappa_{2,2}\big((\nu,j),(\nu',j')\big)\\ &+\mu_{\mathbf{X}_1}[i,\nu]\mu_{\mathbf{X}_2}[\nu',j']\kappa_{2,1}\big((\nu,j),(i',\nu')\big)\\ &+\mu_{\mathbf{X}_2}[\nu,j]\mu_{\mathbf{X}_1}[i',\nu']\kappa_{1,2}\big((i,\nu),(\nu',j')\big)\\ &+\mu_{\mathbf{X}_2}[\nu,j]\mu_{\mathbf{X}_2}[\nu',j']\kappa_{1,1}\big((i,\nu),(i',\nu')\big)\\ &+\kappa_{1,1}\big((i,\nu),(i',\nu')\big)\kappa_{2,2}\big((\nu,j),(\nu',j')\big)\\ &+\kappa_{1,2}\big((i,\nu),(\nu',j')\big)\kappa_{2,1}\big((\nu,j),(i',\nu')\big)\Big].\end{aligned} \tag{45}$$

Finally, the covariance matrix of $\mathbf{x}_3$ is obtained by stacking $\kappa_{3,3}$ according to the same column-major convention:

$$\Sigma_{\mathbf{x}_3}\left[\mathrm{ind}_3(i,j),\mathrm{ind}_3(i',j')\right] = \kappa_{3,3}\big((i,j),(i',j')\big). \tag{46}$$

Algorithm 2 Moment Propagation Through a Bilinear Matrix Node

1: **procedure** BILINEARMATRIXMOMENTS($\mu_{\mathbf{X}_0},\mu_{\mathbf{X}_1},\mu_{\mathbf{X}_2},\Sigma_{\mathbf{x}_0},\Sigma_{\mathbf{x}_1},\Sigma_{\mathbf{x}_2},\Sigma_{\mathbf{x}_0,\mathbf{x}_1},\Sigma_{\mathbf{x}_0,\mathbf{x}_2}$)
2: Compute $\Sigma_{\mathbf{x}_1,\mathbf{x}_2}$ using (31)
3: Construct the covariance-entry maps $\kappa_{\iota,\iota'}$ using the column-major index map
4: Compute $\mu_{\mathbf{X}_3}$ using (35)
5: Compute $\Sigma_{\mathbf{x}_0,\mathbf{x}_3}$, $\Sigma_{\mathbf{x}_1,\mathbf{x}_3}$, and $\Sigma_{\mathbf{x}_2,\mathbf{x}_3}$ using (37)–(40)
6: Compute $\Sigma_{\mathbf{x}_3}$ using (43)–(46)
7: **return** $\mu_{\mathbf{X}_3},\Sigma_{\mathbf{x}_3},\Sigma_{\mathbf{x}_0,\mathbf{x}_3},\Sigma_{\mathbf{x}_1,\mathbf{x}_3},\Sigma_{\mathbf{x}_2,\mathbf{x}_3}$
8: **end procedure**

B.2 LAYERNORM NODE

For one row, let $\mathbf{x}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}},\mathbf{b}_{\mathrm{LN}},\mathbf{g}_{\mathrm{LN}}\in\mathbb{R}^{d_{\mathrm{LN}}}$. The node is

$$y_{\mathrm{LN},i} = b_{\mathrm{LN},i} + g_{\mathrm{LN},i}n_{\mathrm{LN},i}, \qquad \mathbf{n}_{\mathrm{LN}} = \frac{\mathbf{C}_{\mathrm{LN}}\mathbf{x}_{\mathrm{LN}}}{\sqrt{d_{\mathrm{LN}}^{-1}\mathbf{x}_{\mathrm{LN}}^\top\mathbf{C}_{\mathrm{LN}}\mathbf{x}_{\mathrm{LN}}}}, \qquad \boldsymbol{\theta}_{\mathrm{LN}} = \begin{bmatrix}\mathbf{b}_{\mathrm{LN}}\\ \mathbf{g}_{\mathrm{LN}}\end{bmatrix}, \tag{47}$$

with

$$\mathbf{C}_{\mathrm{LN}} = \mathbf{I}_{d_{\mathrm{LN}}} - \frac{1}{d_{\mathrm{LN}}}\mathbf{1}_{d_{\mathrm{LN}}}\mathbf{1}_{d_{\mathrm{LN}}}^\top. \tag{48}$$

The input moments are $\mu_{\mathbf{x}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{x}_{\mathrm{LN}}}$, $\mu_{\boldsymbol{\theta}_{\mathrm{LN}}}$, $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}}$, and $\Sigma_{\mathbf{x}_{\mathrm{LN}},\boldsymbol{\theta}_{\mathrm{LN}}}$. Under a joint-Gaussian approximation to $(\mathbf{x}_{\mathrm{LN}},\boldsymbol{\theta}_{\mathrm{LN}})$, these input moments determine approximations to $\mu_{\mathbf{y}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{y}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{x}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$, and $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$. The deterministic-affine case is obtained by taking $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}} = \mathbf{0}_{2d_{\mathrm{LN}}\times 2d_{\mathrm{LN}}}$ and $\Sigma_{\mathbf{x}_{\mathrm{LN}},\boldsymbol{\theta}_{\mathrm{LN}}} = \mathbf{0}_{d_{\mathrm{LN}}\times 2d_{\mathrm{LN}}}$.

Define the feature-projected Gaussian vector

$$\begin{aligned}\mathbf{c}_{\mathrm{LN}} &= \mathbf{C}_{\mathrm{LN}}\mathbf{x}_{\mathrm{LN}}, \qquad &\mathbf{c}_{\mathrm{LN}} &\sim \mathcal{N}(\mu_{\mathbf{c}_{\mathrm{LN}}},\Sigma_{\mathbf{c}_{\mathrm{LN}}}),\\ \mu_{\mathbf{c}_{\mathrm{LN}}} &= \mathbf{C}_{\mathrm{LN}}\mu_{\mathbf{x}_{\mathrm{LN}}}, \qquad &\Sigma_{\mathbf{c}_{\mathrm{LN}}} &= \mathbf{C}_{\mathrm{LN}}\Sigma_{\mathbf{x}_{\mathrm{LN}}}\mathbf{C}_{\mathrm{LN}}.\end{aligned} \tag{49}$$

Let $\mathbf{B}_{\mathrm{LN}}\in\mathbb{R}^{d_{\mathrm{LN}}\times(d_{\mathrm{LN}}-1)}$ be an orthonormal contrast matrix and define $\mathbf{z}_{\mathrm{LN}} = \mathbf{B}_{\mathrm{LN}}^\top\mathbf{x}_{\mathrm{LN}}$, with

$$\begin{aligned}\mathbf{B}_{\mathrm{LN}}^\top\mathbf{B}_{\mathrm{LN}} &= \mathbf{I}_{d_{\mathrm{LN}}-1}, \qquad &\mathbf{B}_{\mathrm{LN}}\mathbf{B}_{\mathrm{LN}}^\top &= \mathbf{C}_{\mathrm{LN}},\\ \Sigma_{\mathbf{z}_{\mathrm{LN}}} &= \mathbf{B}_{\mathrm{LN}}^\top\Sigma_{\mathbf{x}_{\mathrm{LN}}}\mathbf{B}_{\mathrm{LN}}.\end{aligned} \tag{50}$$

For a deterministic input row, $\mathbf{n}_{\mathrm{LN}}$ is evaluated directly and its covariance and all cross-covariances are zero. For a stochastic input, let

$$\varrho_{\mathrm{LN}}(\mathbf{c}_{\mathrm{LN}}) = \frac{1}{d_{\mathrm{LN}}}\mathbf{c}_{\mathrm{LN}}^\top\mathbf{c}_{\mathrm{LN}}. \tag{51}$$

Then $\mathbf{n}_{\mathrm{LN}} = \mathbf{c}_{\mathrm{LN}}/\sqrt{\varrho_{\mathrm{LN}}(\mathbf{c}_{\mathrm{LN}})}$.

The required moments follow from the Laplace-transform identities

$$\lambda_{\mathrm{LT}}^{-1/2} = \frac{1}{\sqrt{\pi}} \int_0^\infty \zeta^{-1/2} \exp(-\lambda_{\mathrm{LT}}\zeta)\, d\zeta, \qquad \lambda_{\mathrm{LT}}^{-1} = \int_0^\infty \exp(-\lambda_{\mathrm{LT}}\zeta)\, d\zeta, \qquad \lambda_{\mathrm{LT}} > 0. \tag{52}$$

For $\zeta \geq 0$, define

$$\begin{aligned}
\mathbf{\Gamma}_{\mathrm{LN}}(\zeta) &= \mathbf{I}_{d_{\mathrm{LN}}} + \frac{2\zeta}{d_{\mathrm{LN}}} \Sigma_{\mathbf{c}_{\mathrm{LN}}}, \\
\widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) &= \mathbf{\Gamma}_{\mathrm{LN}}(\zeta)^{-1} \mu_{\mathbf{c}_{\mathrm{LN}}}, \\
\widetilde{\Sigma}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) &= \Sigma_{\mathbf{c}_{\mathrm{LN}}} \mathbf{\Gamma}_{\mathrm{LN}}(\zeta)^{-1}, \\
\chi_{\mathrm{LN}}(\zeta) &= \det(\mathbf{\Gamma}_{\mathrm{LN}}(\zeta))^{-1/2} \exp\left(-\frac{\zeta}{d_{\mathrm{LN}}} \mu_{\mathbf{c}_{\mathrm{LN}}}^\top \mathbf{\Gamma}_{\mathrm{LN}}(\zeta)^{-1} \mu_{\mathbf{c}_{\mathrm{LN}}} \right).
\end{aligned} \tag{53}$$

Gaussian integration gives

$$\mathbb{E}\left[\exp(-\zeta \varrho_{\mathrm{LN}}(\mathbf{c}_{\mathrm{LN}}))\right] = \chi_{\mathrm{LN}}(\zeta), \tag{54}$$

$$\mathbb{E}\left[\mathbf{c}_{\mathrm{LN}} \exp(-\zeta \varrho_{\mathrm{LN}}(\mathbf{c}_{\mathrm{LN}}))\right] = \chi_{\mathrm{LN}}(\zeta) \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta), \tag{55}$$

$$\mathbb{E}\left[\mathbf{c}_{\mathrm{LN}} \mathbf{c}_{\mathrm{LN}}^\top \exp(-\zeta \varrho_{\mathrm{LN}}(\mathbf{c}_{\mathrm{LN}}))\right] = \chi_{\mathrm{LN}}(\zeta) \left(\widetilde{\Sigma}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) + \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta)^\top \right). \tag{56}$$

Therefore,

$$\mu_{\mathbf{n}_{\mathrm{LN}}} = \frac{1}{\sqrt{\pi}} \int_0^\infty \zeta^{-1/2} \chi_{\mathrm{LN}}(\zeta) \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta)\, d\zeta, \tag{57}$$

$$\mathbf{\Xi}_{\mathbf{c}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \frac{1}{\sqrt{\pi}} \int_0^\infty \zeta^{-1/2} \chi_{\mathrm{LN}}(\zeta) \left(\widetilde{\Sigma}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) + \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta)^\top \right) d\zeta, \tag{58}$$

$$\Sigma_{\mathbf{c}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \mathbf{\Xi}_{\mathbf{c}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} - \mu_{\mathbf{c}_{\mathrm{LN}}} \mu_{\mathbf{n}_{\mathrm{LN}}}^\top, \tag{59}$$

$$\Sigma_{\mathbf{z}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \mathbf{B}_{\mathrm{LN}}^\top \Sigma_{\mathbf{c}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}, \tag{60}$$

$$\Sigma_{\mathbf{x}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \Sigma_{\mathbf{x}_{\mathrm{LN}}} \mathbf{B}_{\mathrm{LN}} \Sigma_{\mathbf{z}_{\mathrm{LN}}}^{-1} \Sigma_{\mathbf{z}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}, \tag{61}$$

$$\mathbf{\Xi}_{\mathbf{n}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \int_0^\infty \chi_{\mathrm{LN}}(\zeta) \left(\widetilde{\Sigma}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) + \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta) \widetilde{\mu}_{\mathbf{c}_{\mathrm{LN}}}(\zeta)^\top \right) d\zeta, \tag{62}$$

$$\Sigma_{\mathbf{n}_{\mathrm{LN}}} = \mathbf{\Xi}_{\mathbf{n}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} - \mu_{\mathbf{n}_{\mathrm{LN}}} \mu_{\mathbf{n}_{\mathrm{LN}}}^\top. \tag{63}$$

The cross-covariance between the trainable affine parameters and $\mathbf{n}_{\mathrm{LN}}$ follows by Gaussian regression through $\mathbf{z}_{\mathrm{LN}}$:

$$\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}} = \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{x}_{\mathrm{LN}}} \mathbf{B}_{\mathrm{LN}} \Sigma_{\mathbf{z}_{\mathrm{LN}}}^{-1} \Sigma_{\mathbf{z}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}. \tag{64}$$

The quadrature above gives the moments of $\mathbf{n}_{\mathrm{LN}}$ under the Gaussian approximation for $\mathbf{x}_{\mathrm{LN}}$. The affine transformation then contains products $g_{\mathrm{LN},i} n_{\mathrm{LN},i}$, for which we impose a second Gaussian closure on $(\mathbf{b}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}})$. Gaussian product-moment identities give the output mean

$$\mu_{\mathbf{y}_{\mathrm{LN}}}[i] = \mu_{\mathbf{b}_{\mathrm{LN}}}[i] + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \mu_{\mathbf{n}_{\mathrm{LN}}}[i] + \Sigma_{\mathbf{g}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[i, i]. \tag{65}$$

For coordinates i and j, the output covariance is

$$\begin{aligned}
\Sigma_{\mathbf{y}_{\mathrm{LN}}}[i, j] \approx\ & \Sigma_{\mathbf{b}_{\mathrm{LN}}}[i, j] + \mu_{\mathbf{g}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{b}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[i, j] + \mu_{\mathbf{n}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{b}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}}[i, j] \\
& + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \Sigma_{\mathbf{n}_{\mathrm{LN}}, \mathbf{b}_{\mathrm{LN}}}[i, j] + \mu_{\mathbf{n}_{\mathrm{LN}}}[i] \Sigma_{\mathbf{g}_{\mathrm{LN}}, \mathbf{b}_{\mathrm{LN}}}[i, j] + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \mu_{\mathbf{g}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{n}_{\mathrm{LN}}}[i, j] \\
& + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \mu_{\mathbf{n}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{n}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}}[i, j] + \mu_{\mathbf{n}_{\mathrm{LN}}}[i] \mu_{\mathbf{g}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{g}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[i, j] \\
& + \mu_{\mathbf{n}_{\mathrm{LN}}}[i] \mu_{\mathbf{n}_{\mathrm{LN}}}[j] \Sigma_{\mathbf{g}_{\mathrm{LN}}}[i, j] + \Sigma_{\mathbf{g}_{\mathrm{LN}}}[i, j] \Sigma_{\mathbf{n}_{\mathrm{LN}}}[i, j] \\
& + \Sigma_{\mathbf{g}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[i, j] \Sigma_{\mathbf{n}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}}[i, j].
\end{aligned} \tag{66}$$

The input and affine-parameter cross-covariances are

$$\Sigma_{\mathbf{x}_{\mathrm{LN}}, \mathbf{y}_{\mathrm{LN}}}[:, i] \approx \Sigma_{\mathbf{x}_{\mathrm{LN}}, \mathbf{b}_{\mathrm{LN}}}[:, i] + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \Sigma_{\mathbf{x}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[:, i] + \mu_{\mathbf{n}_{\mathrm{LN}}}[i] \Sigma_{\mathbf{x}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}}[:, i], \tag{67}$$

$$\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{y}_{\mathrm{LN}}}[:, i] \approx \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{b}_{\mathrm{LN}}}[:, i] + \mu_{\mathbf{g}_{\mathrm{LN}}}[i] \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{n}_{\mathrm{LN}}}[:, i] + \mu_{\mathbf{n}_{\mathrm{LN}}}[i] \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}, \mathbf{g}_{\mathrm{LN}}}[:, i]. \tag{68}$$

The covariance blocks involving $\mathbf{b}_{\mathrm{LN}}$ and $\mathbf{g}_{\mathrm{LN}}$ are extracted from $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}}$ and $\Sigma_{\mathbf{x}_{\mathrm{LN}},\boldsymbol{\theta}_{\mathrm{LN}}}$.

For a retained external vector $\mathbf{v}_{\mathrm{ext}}$ with known $\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{\mathrm{LN}}}$ and $\Sigma_{\mathbf{v}_{\mathrm{ext}},\boldsymbol{\theta}_{\mathrm{LN}}}$, the same Gaussian closure gives

$$\begin{aligned}\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{y}_{\mathrm{LN}}}[:,i] \approx{}& \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{b}_{\mathrm{LN}}}[:,i] + \mu_{\mathbf{g}_{\mathrm{LN}}}[i]\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{n}_{\mathrm{LN}}}[:,i] \\ &+ \mu_{\mathbf{n}_{\mathrm{LN}}}[i]\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{g}_{\mathrm{LN}}}[:,i], \\ \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{n}_{\mathrm{LN}}} ={}& \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{\mathrm{LN}}}\mathbf{B}_{\mathrm{LN}}\Sigma_{\mathbf{z}_{\mathrm{LN}}}^{-1}\Sigma_{\mathbf{z}_{\mathrm{LN}},\mathbf{n}_{\mathrm{LN}}}. \end{aligned} \tag{69}$$

Let $\mathcal{V}_{\mathrm{LN}}^{\mathrm{ext}}$ index these external vectors and let

$$\mathcal{E}_{\mathrm{LN}} = \{(\mathbf{v}_{\mathrm{ext}}, \Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{x}_{\mathrm{LN}}}, \Sigma_{\mathbf{v}_{\mathrm{ext}},\boldsymbol{\theta}_{\mathrm{LN}}}) : \mathbf{v}_{\mathrm{ext}} \in \mathcal{V}_{\mathrm{LN}}^{\mathrm{ext}}\}. \tag{70}$$

For a matrix input, rows are processed in a fixed order. When row r is processed, $\mathcal{E}_{\mathrm{LN}}$ contains every unprocessed input row $\mathbf{x}_{\mathrm{LN},r'}$ for $r' > r$. When row r' is processed, each completed output $\mathbf{y}_{\mathrm{LN},r}$ is included as an external vector. Equation (69) then constructs all $\Sigma_{\mathbf{y}_{\mathrm{LN},r},\mathbf{y}_{\mathrm{LN},r'}}$ blocks, including dependence induced by the shared affine parameters.

The block-output smoother uses $\Sigma_{\mathbf{x}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$ for upstream activations and $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$ for affine parameters.

Algorithm 3 Propagation Through a LayerNorm Node with Trainable Affine Parameters

procedure LAYERNORMHYBRIDMOMENTS($\mu_{\mathbf{x}_{\mathrm{LN}}}, \Sigma_{\mathbf{x}_{\mathrm{LN}}}, \mu_{\boldsymbol{\theta}_{\mathrm{LN}}}, \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}}}, \Sigma_{\mathbf{x}_{\mathrm{LN}},\boldsymbol{\theta}_{\mathrm{LN}}}, \mathcal{E}_{\mathrm{LN}}$)
  Construct $\mathbf{C}_{\mathrm{LN}}$ using (48) and extract the affine-parameter moment blocks
  if $\mathbf{x}_{\mathrm{LN}}$ is deterministic **then**
    Evaluate $\mathbf{n}_{\mathrm{LN}}$ at $\mu_{\mathbf{x}_{\mathrm{LN}}}$ and set $\Sigma_{\mathbf{n}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{x}_{\mathrm{LN}},\mathbf{n}_{\mathrm{LN}}}$, and $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{n}_{\mathrm{LN}}}$ to zero
    Set every external–normalized cross-covariance to zero
  else
    Compute $\mu_{\mathbf{c}_{\mathrm{LN}}}$ and $\Sigma_{\mathbf{c}_{\mathrm{LN}}}$ using (49)
    Construct $\mathbf{B}_{\mathrm{LN}}$ and $\Sigma_{\mathbf{z}_{\mathrm{LN}}}$ using (50)
    Construct the tilted Gaussian quantities using (53)
    Evaluate the normalized moments using (63)
    Compute $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{n}_{\mathrm{LN}}}$ using (64)
    Compute every external–normalized cross-covariance using the second relation in (69)
  end if
  Compute $\mu_{\mathbf{y}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{y}_{\mathrm{LN}}}$, $\Sigma_{\mathbf{x}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$, and $\Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}$ using (65)–(68)
  Compute every external–output cross-covariance using the first relation in (69)
  return $\mu_{\mathbf{y}_{\mathrm{LN}}}, \Sigma_{\mathbf{y}_{\mathrm{LN}}}, \Sigma_{\mathbf{x}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}, \Sigma_{\boldsymbol{\theta}_{\mathrm{LN}},\mathbf{y}_{\mathrm{LN}}}, \{\Sigma_{\mathbf{v}_{\mathrm{ext}},\mathbf{y}_{\mathrm{LN}}}\}_{\mathbf{v}_{\mathrm{ext}} \in \mathcal{V}_{\mathrm{LN}}^{\mathrm{ext}}}$
end procedure

C Overview of Kalman Bayesian Neural Networks

Kalman Bayesian neural networks (Wagner et al., 2023) approximate retained random vectors as jointly Gaussian and propagate their first two moments. This appendix records the chain-structured primitive used inside the graph routines. At update k, the forward pass retains, for each layer ℓ,

$$\begin{aligned} \mu_{\mathbf{u}_\ell} &= \mathbb{E}[\mathbf{u}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k], & \Sigma_{\mathbf{u}_\ell} &= \mathrm{Cov}(\mathbf{u}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k), \\ \Sigma_{\mathbf{w}_\ell,\mathbf{u}_\ell} &= \mathrm{Cov}(\mathbf{w}_\ell, \mathbf{u}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k), & \Sigma_{\mathbf{z}_{\ell-1},\mathbf{u}_\ell} &= \mathrm{Cov}(\mathbf{z}_{\ell-1}, \mathbf{u}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k), \\ \mu_{\mathbf{z}_\ell} &= \mathbb{E}[\mathbf{z}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k], & \Sigma_{\mathbf{z}_\ell} &= \mathrm{Cov}(\mathbf{z}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k), \\ \Sigma_{\mathbf{u}_\ell,\mathbf{z}_\ell} &= \mathrm{Cov}(\mathbf{u}_\ell, \mathbf{z}_\ell \mid \mathcal{D}_{1:k-1}, \mathbf{x}_k). \end{aligned} \tag{71}$$

Here $\mathbf{z}_0 = \mathbf{x}_k$ and activation moments are needed only for $\ell < L_{\mathrm{net}}$.

The canonical backward primitive operates on jointly Gaussian upstream and downstream vectors $\boldsymbol{\nu}$ and $\boldsymbol{\upsilon}$. Define

$$\mathbf{G}_{\boldsymbol{\nu}\leftarrow\boldsymbol{\upsilon}} = \Sigma_{\boldsymbol{\nu},\boldsymbol{\upsilon}}\Sigma_{\boldsymbol{\upsilon}}^{-1}, \tag{72}$$

$$\Delta\mu_{\boldsymbol{\upsilon}} = \mu_{\boldsymbol{\upsilon}}^{+} - \mu_{\boldsymbol{\upsilon}}, \qquad \Delta\Sigma_{\boldsymbol{\upsilon}} = \Sigma_{\boldsymbol{\upsilon}}^{+} - \Sigma_{\boldsymbol{\upsilon}}. \tag{73}$$

The upstream update is

$$\mu_{\boldsymbol{\nu}}^{+} = \mu_{\boldsymbol{\nu}} + \mathbf{G}_{\boldsymbol{\nu}\leftarrow\boldsymbol{\upsilon}}\Delta\mu_{\boldsymbol{\upsilon}}, \tag{74}$$

$$\Sigma_{\boldsymbol{\nu}}^{+} = \Sigma_{\boldsymbol{\nu}} + \mathbf{G}_{\boldsymbol{\nu}\leftarrow\boldsymbol{\upsilon}}\Delta\Sigma_{\boldsymbol{\upsilon}}\mathbf{G}_{\boldsymbol{\nu}\leftarrow\boldsymbol{\upsilon}}^{\top}. \tag{75}$$

Algorithm 4 Canonical Gaussian Backward Update

procedure GAUSSIANBACKWARDUPDATE($\mu_{\boldsymbol{\nu}}, \Sigma_{\boldsymbol{\nu}}, \Sigma_{\boldsymbol{\nu},\boldsymbol{\upsilon}}, \mu_{\boldsymbol{\upsilon}}, \Sigma_{\boldsymbol{\upsilon}}, \mu_{\boldsymbol{\upsilon}}^{+}, \Sigma_{\boldsymbol{\upsilon}}^{+}$)
  Compute $\mathbf{G}_{\boldsymbol{\nu}\leftarrow\boldsymbol{\upsilon}}$ using (72)
  Compute $\Delta\mu_{\boldsymbol{\upsilon}}$ and $\Delta\Sigma_{\boldsymbol{\upsilon}}$ using (73)
  Apply (74)–(75)
  return $\mu_{\boldsymbol{\nu}}^{+}, \Sigma_{\boldsymbol{\nu}}^{+}$
end procedure

For layer ℓ, the weight and previous-node updates apply for $1 \le \ell \le L_{\mathrm{net}}$; the preactivation-from-activation update applies only for $1 \le \ell < L_{\mathrm{net}}$. The three specializations are

$$\begin{aligned}(\mu_{\mathbf{w}_\ell}^{+}, \Sigma_{\mathbf{w}_\ell}^{+}) \leftarrow{}& \textsc{GaussianBackwardUpdate}\\ &\big(\mu_{\mathbf{w}_\ell}, \Sigma_{\mathbf{w}_\ell}, \Sigma_{\mathbf{w}_\ell,\mathbf{u}_\ell},\\ &\quad \mu_{\mathbf{u}_\ell}, \Sigma_{\mathbf{u}_\ell}, \mu_{\mathbf{u}_\ell}^{+}, \Sigma_{\mathbf{u}_\ell}^{+}\big),\end{aligned} \tag{76}$$

$$\begin{aligned}(\mu_{\mathbf{u}_\ell}^{+}, \Sigma_{\mathbf{u}_\ell}^{+}) \leftarrow{}& \textsc{GaussianBackwardUpdate}\\ &\big(\mu_{\mathbf{u}_\ell}, \Sigma_{\mathbf{u}_\ell}, \Sigma_{\mathbf{u}_\ell,\mathbf{z}_\ell},\\ &\quad \mu_{\mathbf{z}_\ell}, \Sigma_{\mathbf{z}_\ell}, \mu_{\mathbf{z}_\ell}^{+}, \Sigma_{\mathbf{z}_\ell}^{+}\big), \qquad \ell < L_{\mathrm{net}},\end{aligned} \tag{77}$$

$$\begin{aligned}(\mu_{\mathbf{z}_{\ell-1}}^{+}, \Sigma_{\mathbf{z}_{\ell-1}}^{+}) \leftarrow{}& \textsc{GaussianBackwardUpdate}\\ &\big(\mu_{\mathbf{z}_{\ell-1}}, \Sigma_{\mathbf{z}_{\ell-1}}, \Sigma_{\mathbf{z}_{\ell-1},\mathbf{u}_\ell},\\ &\quad \mu_{\mathbf{u}_\ell}, \Sigma_{\mathbf{u}_\ell}, \mu_{\mathbf{u}_\ell}^{+}, \Sigma_{\mathbf{u}_\ell}^{+}\big).\end{aligned} \tag{78}$$

The target supplies $(\mu_{\mathbf{u}_{L_{\mathrm{net}}}}^{+}, \Sigma_{\mathbf{u}_{L_{\mathrm{net}}}}^{+})$.

The forward pass stores the quantities required by the backward pass in a single cache. The parameter mean follows the ordering in Section 3, $\mu_{\boldsymbol{\omega}} = [\mu_{\mathbf{w}_1}^{\top} \ \cdots \ \mu_{\mathbf{w}_{L_{\mathrm{net}}}}^{\top}]^{\top}$. Under the maintained cross-layer factorization, $\Sigma_{\boldsymbol{\omega}} = \mathrm{blockdiag}(\Sigma_{\mathbf{w}_1}, \ldots, \Sigma_{\mathbf{w}_{L_{\mathrm{net}}}})$.

Algorithm 5 Chain-Structured Forward Moments

procedure MLPFORWARDMOMENTS($\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}$)
  Set $\mu_{\mathbf{z}_0} \leftarrow \mu_{\mathbf{x}}$ and $\Sigma_{\mathbf{z}_0} \leftarrow \Sigma_{\mathbf{x}}$
  Initialize cache $\mathcal{C}^{\mathrm{MLP}}$ with $(\mu_{\mathbf{z}_0}, \Sigma_{\mathbf{z}_0})$ and indexed layer fields
  for $\ell = 1, 2, \ldots, L_{\mathrm{net}}$ **do**
    Compute and cache $\mu_{\mathbf{u}_\ell}$, $\Sigma_{\mathbf{u}_\ell}$, $\Sigma_{\mathbf{w}_\ell,\mathbf{u}_\ell}$, and $\Sigma_{\mathbf{z}_{\ell-1},\mathbf{u}_\ell}$
    if $\ell < L_{\mathrm{net}}$ **then**
      Compute and cache $\mu_{\mathbf{z}_\ell}$, $\Sigma_{\mathbf{z}_\ell}$, and $\Sigma_{\mathbf{u}_\ell,\mathbf{z}_\ell}$
    end if
  end for
  return $\mu_{\mathbf{y}} = \mu_{\mathbf{u}_{L_{\mathrm{net}}}}$, $\Sigma_{\mathbf{y}} = \Sigma_{\mathbf{u}_{L_{\mathrm{net}}}}$, $\mathcal{C}^{\mathrm{MLP}}$
end procedure

Algorithm 6 Chain-Structured Backward Moments

procedure MLPBACKWARDMOMENTS($\mu_{\mathbf{y}}^{\mathrm{tar}}, \Sigma_{\mathbf{y}}^{\mathrm{tar}}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}, \mathcal{C}^{\mathrm{MLP}}$)
  Set $(\mu^{+}_{\mathbf{u}_{L_{\mathrm{net}}}}, \Sigma^{+}_{\mathbf{u}_{L_{\mathrm{net}}}}) \leftarrow (\mu_{\mathbf{y}}^{\mathrm{tar}}, \Sigma_{\mathbf{y}}^{\mathrm{tar}})$
  for $\ell = L_{\mathrm{net}}, L_{\mathrm{net}} - 1, \ldots, 1$ **do**
    Update $(\mu^{+}_{\mathbf{w}_\ell}, \Sigma^{+}_{\mathbf{w}_\ell})$ and $(\mu^{+}_{\mathbf{z}_{\ell-1}}, \Sigma^{+}_{\mathbf{z}_{\ell-1}})$ with GAUSSIANBACKWARDUPDATE
    if $\ell > 1$ **then**
      Update $(\mu^{+}_{\mathbf{u}_{\ell-1}}, \Sigma^{+}_{\mathbf{u}_{\ell-1}})$ from $(\mu^{+}_{\mathbf{z}_{\ell-1}}, \Sigma^{+}_{\mathbf{z}_{\ell-1}})$
    end if
  end for
  Assemble $\mu^{+}_{\boldsymbol{\omega}}$ and $\Sigma^{+}_{\boldsymbol{\omega}}$ in the parameter order above
  return $\mu^{+}_{\boldsymbol{\omega}}, \Sigma^{+}_{\boldsymbol{\omega}}$
end procedure

Algorithm 7 Sequential Kalman Bayesian Neural Network

Require: $\mathcal{D}_{\mathrm{mom}} = [(\mathbf{x}_k, \mu_{\mathbf{y}_k}^{\mathrm{tar}}, \Sigma_{\mathbf{y}_k}^{\mathrm{tar}})]_{k=1}^{N_{\mathrm{data}}}$ and prior moments $(\mu_{\boldsymbol{\omega}}^{\mathrm{prior}}, \Sigma_{\boldsymbol{\omega}}^{\mathrm{prior}})$
$(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \leftarrow (\mu_{\boldsymbol{\omega}}^{\mathrm{prior}}, \Sigma_{\boldsymbol{\omega}}^{\mathrm{prior}})$
for $k = 1, 2, \ldots, N_{\mathrm{data}}$ **do**
  Set $\mu_{\mathbf{x}_k} \leftarrow \mathbf{x}_k$ and $\Sigma_{\mathbf{x}_k} \leftarrow \mathbf{0}_{d_x \times d_x}$
  $(\mu_{\mathbf{y}_k}^{\mathrm{pred}}, \Sigma_{\mathbf{y}_k}^{\mathrm{pred}}, \mathcal{C}_k^{\mathrm{MLP}}) \leftarrow$ MLPFORWARDMOMENTS$(\mu_{\mathbf{x}_k}, \Sigma_{\mathbf{x}_k}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$
  $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \leftarrow$ MLPBACKWARDMOMENTS$(\mu_{\mathbf{y}_k}^{\mathrm{tar}}, \Sigma_{\mathbf{y}_k}^{\mathrm{tar}}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}, \mathcal{C}_k^{\mathrm{MLP}})$
end for
return $\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}$

D TRANSFORMER COMPUTATION-GRAPH EXAMPLE

A pre-normalization transformer instantiates the forward and backward block constructions in Sections 4.2 and 4.3. Its attention and feed-forward residual subblocks provide the block outputs, and their internal branches are evaluated with the node rules in Section 4.1.

D.1 FORWARD MOMENT PROPAGATION THROUGH MULTI-HEAD ATTENTION

We use scaled dot-product multi-head self-attention (Vaswani et al., 2017). For attention residual subblock b, let n_{tok} be the token count, n_{head} the number of heads, d_{model} the model width, d_{key} the key/query width, and d_{value} the per-head value width. The attention variables and head-concatenation output have dimensions

$$\begin{aligned}
\mathbf{X}_{\mathrm{att},b}, \mathbf{U}_{\mathrm{att},b}, \mathbf{Z}_{\mathrm{att},b}, \mathbf{R}_b &\in \mathbb{R}^{n_{\mathrm{tok}} \times d_{\mathrm{model}}}, \\
\mathbf{Q}_{\mathrm{att},b,h}, \mathbf{K}_{\mathrm{att},b,h} &\in \mathbb{R}^{n_{\mathrm{tok}} \times d_{\mathrm{key}}}, \\
\mathbf{V}_{\mathrm{att},b,h}, \mathbf{H}_{\mathrm{att},b,h} &\in \mathbb{R}^{n_{\mathrm{tok}} \times d_{\mathrm{value}}}, \\
\mathbf{S}_{\mathrm{att},b,h}, \mathbf{A}_{\mathrm{att},b,h} &\in \mathbb{R}^{n_{\mathrm{tok}} \times n_{\mathrm{tok}}}, \\
\mathbf{M}_{\mathrm{att},b} &\in \{0, -\infty\}^{n_{\mathrm{tok}} \times n_{\mathrm{tok}}}, \\
\mathbf{H}_{\mathrm{cat},b} &\in \mathbb{R}^{n_{\mathrm{tok}} \times (n_{\mathrm{head}} d_{\mathrm{value}})}.
\end{aligned} \tag{79}$$

The projection matrices have fixed shapes

$$\begin{aligned}
\mathbf{W}_{Q,b,h}, \mathbf{W}_{K,b,h} &\in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{key}}}, \qquad & \mathbf{W}_{V,b,h} &\in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{value}}}, \\
\mathbf{W}_{O,b} &\in \mathbb{R}^{(n_{\mathrm{head}} d_{\mathrm{value}}) \times d_{\mathrm{model}}}.
\end{aligned} \tag{80}$$

We write $\mathbf{w}_{O,b} = \mathrm{vec}(\mathbf{W}_{O,b})$ and vectorize the Q, K, and V projection weights analogously. Here $\mathbf{M}_{\mathrm{att},b}$ is a fixed additive mask: zero entries are active and $-\infty$ entries are masked, with the latter assigned exactly zero softmax probability. Every mask row has at least one active entry. Attention applies the LayerNorm rule with $d_{\mathrm{LN}} = d_{\mathrm{model}}$ independently to every token row, with

$\mathbf{b}_{\mathrm{LN,att},b}, \mathbf{g}_{\mathrm{LN,att},b} \in \mathbb{R}^{d_{\mathrm{model}}}$. For head h,

$$\begin{aligned}
\mathbf{U}_{\mathrm{att},b} &= \mathrm{LayerNorm}(\mathbf{X}_{\mathrm{att},b}; \mathbf{b}_{\mathrm{LN,att},b}, \mathbf{g}_{\mathrm{LN,att},b}) \,, \\
\mathbf{Q}_{\mathrm{att},b,h} &= \mathbf{U}_{\mathrm{att},b}\mathbf{W}_{Q,b,h}, \\
\mathbf{K}_{\mathrm{att},b,h} &= \mathbf{U}_{\mathrm{att},b}\mathbf{W}_{K,b,h}, \\
\mathbf{V}_{\mathrm{att},b,h} &= \mathbf{U}_{\mathrm{att},b}\mathbf{W}_{V,b,h}, \\
\mathbf{S}_{\mathrm{att},b,h} &= d_{\mathrm{key}}^{-1/2}\mathbf{Q}_{\mathrm{att},b,h}\mathbf{K}_{\mathrm{att},b,h}^{\top}, \\
\mathbf{A}_{\mathrm{att},b,h} &= \mathrm{softmax}(\mathbf{S}_{\mathrm{att},b,h} + \mathbf{M}_{\mathrm{att},b}) \,, \\
\mathbf{H}_{\mathrm{att},b,h} &= \mathbf{A}_{\mathrm{att},b,h}\mathbf{V}_{\mathrm{att},b,h}, \\
\mathbf{H}_{\mathrm{cat},b} &= [\mathbf{H}_{\mathrm{att},b,1} \ \cdots \ \mathbf{H}_{\mathrm{att},b,n_{\mathrm{head}}}], \\
\mathbf{Z}_{\mathrm{att},b} &= \mathbf{H}_{\mathrm{cat},b}\mathbf{W}_{O,b}, \\
\mathbf{R}_b &= \mathbf{X}_{\mathrm{att},b} + \mathbf{Z}_{\mathrm{att},b}.
\end{aligned} \tag{81}$$

The affine parameter vector is ordered as shift followed by scale, $\boldsymbol{\theta}_{\mathrm{LN,att},b} = [\mathbf{b}_{\mathrm{LN,att},b}^{\top} \, \mathbf{g}_{\mathrm{LN,att},b}^{\top}]^{\top}$. The bundle $\mathcal{B}_{\mathrm{att},b}$ contains parameter moments and their retained cross-covariances; $\mathcal{B}_{\mathrm{att},b,h}$ is its restriction to head h. The fixed mask is excluded from this stochastic bundle. After each record, the assumed-density approximation restores independence across weight blocks and between each projection-weight block and its input activation. The forward pass then retains the cross-covariances induced within the record; these are the blocks consumed by the product rules above. Let $\mathcal{I}_b$ contain the stochastic subblock inputs and parameter vectors assigned to block b, together with every retained latent vector, and write $\boldsymbol{\nu}_s$ for the vector at site $s \in \mathcal{I}_b$. The bundle $\mathcal{E}_{\mathrm{att},b}$ stores, for every currently available $s \in \mathcal{I}_b$, the cross-covariances between $\boldsymbol{\nu}_s$ and the inputs of the current node. It is initialized from the LayerNorm outputs and $\mathcal{B}_{\mathrm{att},b}$, with zero blocks supplied by the restored factorization, and extended as latent sites are created. Let $\mathcal{I}_{b,h}^{\mathrm{loc}} \subseteq \mathcal{I}_b$ denote the sites available when head h is entered, let $\mathcal{I}_{b,h}^{\mathrm{out}}$ add the sites created within that head, and let $\mathcal{I}_b^{\mathrm{head}}$ contain the sites available after all heads have been processed. The fixed block specification is $\mathcal{F}_{\mathrm{att},b} = (\mathbf{M}_{\mathrm{att},b}, n_{\mathrm{head}}, d_{\mathrm{key}})$.

Algorithm 8 composes the head-level routine in Section D.2.

A complete transformer forward pass composes these local routines residual subblock by residual subblock. A transformer layer contributes an attention subblock and a feed-forward subblock in sequence. Feed-forward subblocks use matrix-product and activation nodes, residual paths use additive nodes, and normalization uses LAYERNORMHYBRIDMOMENTS. The final readout is the terminal block, with $\mathbf{r}_{N_{\mathrm{blk}}} = \mathbf{y}_k$. We denote this composition by TRANSFORMERFORWARDMOMENTS. Its first argument is a deterministic network input, written $\mathbf{x}_k$ for a generic record or $\mathbf{c}_k$ for a decision-transformer context; the procedure sets its covariance to the dimensioned zero matrix and extracts the required node and subblock parameter bundles from $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$, including $\mathcal{B}_{\mathrm{att},b}$ for an attention subblock. Its interface is

$$\begin{aligned}
&\textsc{TransformerForwardMoments}(\mathbf{x}_k, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \\
&\quad \longrightarrow (\mu_{\mathbf{y}_k}^{\mathrm{pred}}, \Sigma_{\mathbf{y}_k}^{\mathrm{pred}}, \mathcal{C}_k^{\mathrm{fwd}}),
\end{aligned} \tag{82}$$

where $\mathcal{C}_k^{\mathrm{fwd}}$ is the graph-wide cache. Feed-forward activations use the piecewise-linear moment operator defined above. TRANSFORMERFORWARDMOMENTS is the transformer implementation of NETWORKFORWARDMOMENTS in (12).

Algorithm 8 Moment Propagation Through Multi-Head Attention Subblock b

procedure ATTENTIONBLOCKMOMENTS($b, \mu_{\mathbf{X}_{\mathrm{att},b}}, \Sigma_{\mathbf{x}_{\mathrm{att},b}}, \mathcal{B}_{\mathrm{att},b}, \mathcal{F}_{\mathrm{att},b}$)
  Apply LAYERNORMHYBRIDMOMENTS to rows in order, mapping each row of $\mathbf{X}_{\mathrm{att},b}$ to $\mathbf{x}_{\mathrm{LN}}$
    using unprocessed input rows and completed output rows as $\mathcal{E}_{\mathrm{LN}}$; supply $\boldsymbol{\theta}_{\mathrm{LN,att},b}$ through the native affine-parameter argument
  Assemble $\mu_{\mathbf{U}_{\mathrm{att},b}}$, $\Sigma_{\mathbf{u}_{\mathrm{att},b}}$, and all retained input, affine-parameter, and cross-row covariance blocks; initialize $\mathcal{E}_{\mathrm{att},b}$
  for $h = 1, 2, \ldots, n_{\mathrm{head}}$ **do**
    $(\mu_{\mathbf{H}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{h}_{\mathrm{att},b,h}}, \{\Sigma_{\boldsymbol{\nu}_s,\mathbf{h}_{\mathrm{att},b,h}}\}_{s\in\mathcal{I}^{\mathrm{out}}_{b,h}}, \mathcal{C}_{\mathrm{att},b,h}) \leftarrow$
    ATTENTIONHEADMOMENTS($b, h, \mu_{\mathbf{U}_{\mathrm{att},b}}, \Sigma_{\mathbf{u}_{\mathrm{att},b}}, \mathcal{B}_{\mathrm{att},b,h}, \mathcal{E}_{\mathrm{att},b}, \mathbf{M}_{\mathrm{att},b}, d_{\mathrm{key}}$)
    Extend $\mathcal{E}_{\mathrm{att},b}$ with every newly created latent site and its retained covariance with $\mathbf{u}_{\mathrm{att},b}$
  end for
  for each still-missing pair $(s, h) \in \mathcal{I}^{\mathrm{head}}_b \times \{1, \ldots, n_{\mathrm{head}}\}$ **do**
    $\Sigma_{\boldsymbol{\nu}_s,\mathbf{h}_{\mathrm{att},b,h}} \leftarrow \Sigma_{\boldsymbol{\nu}_s,\mathbf{u}_{\mathrm{att},b}} \Sigma^{-1}_{\mathbf{u}_{\mathrm{att},b}} \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{h}_{\mathrm{att},b,h}}$
  end for
  for each missing head pair $h < h'$ with $h, h' \in \{1, \ldots, n_{\mathrm{head}}\}$ **do**
    $\Sigma_{\mathbf{h}_{\mathrm{att},b,h},\mathbf{h}_{\mathrm{att},b,h'}} \leftarrow$
    GAUSSIANMARKOVCLOSURE($\Sigma^{\top}_{\mathbf{u}_{\mathrm{att},b},\mathbf{h}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{h}_{\mathrm{att},b,h'}}$)
  end for
  Concatenate the head means and site–head cross-covariances; assemble $\Sigma_{\mathbf{h}_{\mathrm{cat},b}}$ and $\Sigma_{\boldsymbol{\nu}_s,\mathbf{h}_{\mathrm{cat},b}}$ for $s \in \mathcal{I}^{\mathrm{head}}_b$
  $(\mu_{\mathbf{Z}_{\mathrm{att},b}}, \Sigma_{\mathbf{z}_{\mathrm{att},b}}, \Sigma_{\mathbf{h}_{\mathrm{cat},b},\mathbf{z}_{\mathrm{att},b}}, \Sigma_{\mathbf{w}_{O,b},\mathbf{z}_{\mathrm{att},b}}) \leftarrow$
    MATRIXPRODUCTMOMENTS($\mu_{\mathbf{H}_{\mathrm{cat},b}}, \mu_{\mathbf{W}_{O,b}}, \Sigma_{\mathbf{h}_{\mathrm{cat},b}}, \Sigma_{\mathbf{w}_{O,b}}$)
  Propagate every retained site covariance to $\mathbf{z}_{\mathrm{att},b}$ using the matrix-product external rule
  $(\mu_{\mathbf{R}_b}, \Sigma_{\mathbf{r}_b}, \Sigma_{\mathbf{x}_{\mathrm{att},b},\mathbf{r}_b}, \Sigma_{\mathbf{z}_{\mathrm{att},b},\mathbf{r}_b}) \leftarrow$
    MATRIXSUMMOMENTS($\mu_{\mathbf{X}_{\mathrm{att},b}}, \mu_{\mathbf{Z}_{\mathrm{att},b}}, \Sigma_{\mathbf{x}_{\mathrm{att},b}}, \Sigma_{\mathbf{z}_{\mathrm{att},b}}, \Sigma_{\mathbf{x}_{\mathrm{att},b},\mathbf{z}_{\mathrm{att},b}}$)
  Retain $\Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b}$ for every available stochastic site s using the exact additive merge rule
  Set $\mu_{\mathbf{r}_b} \leftarrow \mathrm{vec}(\mu_{\mathbf{R}_b})$
  Let $\mathcal{C}_{\mathrm{att},b}$ collect the indexed intermediate moments, cross-covariances, parameter moments, and head caches, including $\mu_{\mathbf{r}_b}$
  return $\mu_{\mathbf{R}_b}, \Sigma_{\mathbf{r}_b}, \mathcal{C}_{\mathrm{att},b}$
end procedure

D.2 Forward Moment Propagation through a Single Attention Head

Algorithm 8 calls the following head-level routine. Define the auxiliary matrices $\mathbf{T}_{\mathrm{att},b,h} = \mathbf{K}_{\mathrm{att},b,h}^{\top}$ and $\widetilde{\mathbf{S}}_{\mathrm{att},b,h} = \mathbf{Q}_{\mathrm{att},b,h}\mathbf{T}_{\mathrm{att},b,h}$; hence $\mathbf{S}_{\mathrm{att},b,h} = d_{\mathrm{key}}^{-1/2}\widetilde{\mathbf{S}}_{\mathrm{att},b,h}$. The moments of $\mathbf{T}_{\mathrm{att},b,h}$ are obtained by the fixed transpose permutation. Each newly created smoothing site is added to the working external bundle with its native input cross-covariances before the next node is processed. Thus the routine retains direct site–head-output covariances for $\mathcal{I}_{b,h}^{\mathrm{out}}$; all retained head moments are stored in $\mathcal{C}_{\mathrm{att},b,h}$.

Algorithm 9 Forward Moment Propagation Through Attention Head (b, h)

procedure AttentionHeadMoments($b, h, \mu_{\mathbf{U}_{\mathrm{att},b}}, \Sigma_{\mathbf{u}_{\mathrm{att},b}}, \mathcal{B}_{\mathrm{att},b,h}, \mathcal{E}_{\mathrm{att},b}, \mathbf{M}_{\mathrm{att},b}, d_{\mathrm{key}}$)
- Extract the moments of $\mathbf{W}_{Q,b,h}$, $\mathbf{W}_{K,b,h}$, and $\mathbf{W}_{V,b,h}$ from $\mathcal{B}_{\mathrm{att},b,h}$
- Initialize the working external bundle with $\mathbf{u}_{\mathrm{att},b}$ and the sites in $\mathcal{I}_{b,h}^{\mathrm{loc}}$ represented by $\mathcal{E}_{\mathrm{att},b}$
- Obtain the moments of $\mathbf{Q}_{\mathrm{att},b,h}$, $\mathbf{K}_{\mathrm{att},b,h}$, and $\mathbf{V}_{\mathrm{att},b,h}$ with MatrixProductMoments
- Propagate every retained-site cross-covariance to $\mathbf{Q}_{\mathrm{att},b,h}$, $\mathbf{K}_{\mathrm{att},b,h}$, and $\mathbf{V}_{\mathrm{att},b,h}$ with (41); add these sites to the working bundle
- Set $\mathbf{T}_{\mathrm{att},b,h} \leftarrow \mathbf{K}_{\mathrm{att},b,h}^{\top}$ and permute its mean, covariance, and retained cross-covariances
- $(\mu_{\widetilde{\mathbf{S}}_{\mathrm{att},b,h}}, \Sigma_{\widetilde{\mathbf{s}}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\widetilde{\mathbf{s}}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{q}_{\mathrm{att},b,h},\widetilde{\mathbf{s}}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{t}_{\mathrm{att},b,h},\widetilde{\mathbf{s}}_{\mathrm{att},b,h}}) \leftarrow$ BilinearMatrixMoments$(\mu_{\mathbf{U}_{\mathrm{att},b}}, \mu_{\mathbf{Q}_{\mathrm{att},b,h}}, \mu_{\mathbf{T}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b}}, \Sigma_{\mathbf{q}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{t}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{q}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{t}_{\mathrm{att},b,h}})$
- Propagate every retained-site cross-covariance to $\widetilde{\mathbf{S}}_{\mathrm{att},b,h}$ with (41); add the score sites to the working bundle
- Scale the mean and every retained input–output cross-covariance by $d_{\mathrm{key}}^{-1/2}$ and the covariance by d_{key}^{-1} to obtain the moments of $\mathbf{S}_{\mathrm{att},b,h}$
- Select the unmasked coordinates as $\mathbf{s}_{\mathrm{att},b,h}^{\mathrm{free}}$ using $\mathbf{M}_{\mathrm{att},b}$
- Apply SoftmaxMoments row-wise and assemble cross-row blocks with (10)
- Propagate every retained-site cross-covariance through the active logits with (9); add the attention-probability site to the working bundle
- Restore masked probabilities, covariance rows and columns, and external cross-covariance columns as deterministic zeros to assemble $\mu_{\mathbf{A}_{\mathrm{att},b,h}}$, $\Sigma_{\mathbf{a}_{\mathrm{att},b,h}}$, and $\Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{a}_{\mathrm{att},b,h}}$
- $(\mu_{\mathbf{H}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{h}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{h}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{a}_{\mathrm{att},b,h},\mathbf{h}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{v}_{\mathrm{att},b,h},\mathbf{h}_{\mathrm{att},b,h}}) \leftarrow$ BilinearMatrixMoments$(\mu_{\mathbf{U}_{\mathrm{att},b}}, \mu_{\mathbf{A}_{\mathrm{att},b,h}}, \mu_{\mathbf{V}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b}}, \Sigma_{\mathbf{a}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{v}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{a}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{u}_{\mathrm{att},b},\mathbf{v}_{\mathrm{att},b,h}})$
- Propagate every retained-site cross-covariance to $\mathbf{H}_{\mathrm{att},b,h}$ with (41)
- Let $\mathcal{C}_{\mathrm{att},b,h}$ collect every indexed intermediate moment and retained cross-covariance above
- Set $\mathcal{I}_{b,h}^{\mathrm{out}}$ to the incoming and newly created head sites
- **return** $\mu_{\mathbf{H}_{\mathrm{att},b,h}}, \Sigma_{\mathbf{h}_{\mathrm{att},b,h}}, \{\Sigma_{\boldsymbol{\nu}_s,\mathbf{h}_{\mathrm{att},b,h}}\}_{s\in\mathcal{I}_{b,h}^{\mathrm{out}}}, \mathcal{C}_{\mathrm{att},b,h}$

end procedure

D.3 Subblock-Output Backward Pass through Multi-Head Attention

Let $\mathcal{I}_b$ be the smoothing sites in attention subblock b. Algorithm 10 specializes the matrix-modified subblock-output update in (15)– (17). Because every retained cross-covariance is with the same vector $\mathbf{r}_b$, each marginal incorporates every path represented by the joint approximation in a single update. Setting $\mathbf{L}_s = \mathbf{I}_{d_s}$ for every $s \in \mathcal{I}_b$ gives the canonical marginal Gaussian smoother. The graph-wide TransformerBackwardMoments procedure first applies the block update to the terminal readout, then applies the corresponding attention and feed-forward routines in reverse residual-subblock order. TransformerBackwardMoments updates and assembles the parameter moments and returns the site marginals requested in $\mathcal{I}_{\mathrm{req}}$. It passes each stochastic subblock-input marginal to its predecessor block; the deterministic network input is not updated. It is the transformer implementation of NetworkBackwardMoments in (22).

Algorithm 10 Subblock-Output Marginal Smoothing for Pre-Normalization Attention

procedure ATTENTIONBLOCKBACKWARDMOMENTS($b, \mu^+_{\mathbf{r}_b}, \Sigma^+_{\mathbf{r}_b}, \mathcal{C}_{\mathrm{att},b}, \{\mathbf{L}_s\}_{s\in\mathcal{I}_b}$)
  Extract $\mu_{\mathbf{r}_b}, \Sigma_{\mathbf{r}_b}$, and $\{\mu_{\boldsymbol{\nu}_s}, \Sigma_{\boldsymbol{\nu}_s}, \Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b} : s \in \mathcal{I}_b\}$ from $\mathcal{C}_{\mathrm{att},b}$
  Compute $\Delta\mu_b$ using (14)
  for $s \in \mathcal{I}_b$ **do**
    $\mathbf{G}_s \leftarrow \mathbf{L}_s\Sigma_{\boldsymbol{\nu}_s,\mathbf{r}_b}\Sigma^{-1}_{\mathbf{r}_b}$
    Compute $\mu^+_{\boldsymbol{\nu}_s}$ and $\Sigma^+_{\boldsymbol{\nu}_s}$ using (16)–(17)
  end for
  return $\{(\mu^+_{\boldsymbol{\nu}_s}, \Sigma^+_{\boldsymbol{\nu}_s})\}_{s\in\mathcal{I}_b}$
end procedure

E A FULL-HISTORY SEQUENTIAL MONTE CARLO EXPERT

We obtain the expert moments required by Algorithm 1 using persistent full-history SMC (Del Moral et al., 2006). In replacement mode, it evaluates the predictive law $\mathcal{P}^{\mathrm{pred,exp}}_k$ and factor Λ^{exp}_k from (20)– (21) for the realized target $\mathcal{Q}^{\mathrm{tar}}_k$. In observation mode, Λ^{exp}_k is instead a supplied likelihood. Starting from the same task prior as the student, the expert posterior is

$$p^{\mathrm{exp}}_k(\boldsymbol{\omega}) \propto p^{\mathrm{exp}}_0(\boldsymbol{\omega}) \prod_{j=1}^{k} \Lambda^{\mathrm{exp}}_j(\boldsymbol{\omega}). \tag{83}$$

The expert and student share the task prior, $p^{\mathrm{exp}}_0(\boldsymbol{\omega}) = p^{\mathrm{stu}}_0(\boldsymbol{\omega})$, and then evolve independently using the same ordered inputs, realized targets, and update factors. Observation likelihoods give the ordinary posterior; output replacement gives the generalized posterior defined by (21). The particle population persists across ordered records and is reinitialized when a new task is drawn.

Suppose update k produces N_{part} equal-weight particles approximating p^{exp}_k. Propagating the same population through the network at $\mathbf{x}_k$ produces site vectors $\{\boldsymbol{\nu}^{(i_{\mathrm{part}})}_{s,k}\}^{N_{\mathrm{part}}}_{i_{\mathrm{part}}=1}$. Their expert moments are

$$\begin{aligned}
\mu^{\mathrm{exp},+}_{\boldsymbol{\nu}_{s,k}} &= \frac{1}{N_{\mathrm{part}}} \sum_{i_{\mathrm{part}}=1}^{N_{\mathrm{part}}} \boldsymbol{\nu}^{(i_{\mathrm{part}})}_{s,k}, \\
\Sigma^{\mathrm{exp},+}_{\boldsymbol{\nu}_{s,k}} &= \frac{1}{N_{\mathrm{part}}} \sum_{i_{\mathrm{part}}=1}^{N_{\mathrm{part}}} \left(\boldsymbol{\nu}^{(i_{\mathrm{part}})}_{s,k} - \mu^{\mathrm{exp},+}_{\boldsymbol{\nu}_{s,k}}\right)\left(\boldsymbol{\nu}^{(i_{\mathrm{part}})}_{s,k} - \mu^{\mathrm{exp},+}_{\boldsymbol{\nu}_{s,k}}\right)^{\top}.
\end{aligned} \tag{84}$$

A shared particle population preserves correspondence across branches and merges. Exact inference, MCMC, or another sufficiently accurate method may supply the same expert moments.

F ALGORITHMS FOR REWARD-GUIDED ADAPTATION

Algorithm 11 implements the proposed method from Section 5.1. Algorithm 12 implements the proposed method form Section 5.2.

Algorithm 11 Optimization beyond Data Distribution

Require: $\mathcal{D}_{\mathrm{RG}} = [(\mathbf{x}_k, \mathbf{s}_k)]_{k=1}^{N_{\mathrm{RG}}}$, prior moments $(\mu_{\boldsymbol{\omega}}^{\mathrm{prior}}, \Sigma_{\boldsymbol{\omega}}^{\mathrm{prior}})$, $\beta_{\mathrm{R}} > 0$, π_{ref}, $\mathsf{Q}^{\pi_{\mathrm{ref}}}$, and a target-moment integration rule

$(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \leftarrow (\mu_{\boldsymbol{\omega}}^{\mathrm{prior}}, \Sigma_{\boldsymbol{\omega}}^{\mathrm{prior}})$
for $k = 1, 2, \ldots, N_{\mathrm{RG}}$ **do**
  $(\mu_{\mathbf{a}_k}^{\mathrm{pred}}, \Sigma_{\mathbf{a}_k}^{\mathrm{pred}}, \mathcal{C}_k^{\mathrm{fwd}}) \leftarrow$ NetworkForwardMoments$(\mathbf{x}_k, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$
  Compute $(\mu_{\mathbf{a}_k}^{\mathrm{tar}}, \Sigma_{\mathbf{a}_k}^{\mathrm{tar}})$ from (25) at $\mathbf{s}_k$
  $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \leftarrow$ NetworkBackwardMoments$(\mu_{\mathbf{a}_k}^{\mathrm{tar}}, \Sigma_{\mathbf{a}_k}^{\mathrm{tar}}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}, \mathcal{C}_k^{\mathrm{fwd}}, \{\mathbf{L}_s = \mathbf{I}_{d_s}\}_{s\in\mathcal{I}}; \emptyset)$
end for
return $\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}$

Algorithm 12 Learning to Optimize beyond Data Distribution

Require: p_{task} whose tasks supply prior moments, training and validation records, reference-policy access or samples for ϕ, value estimates, and a differentiable target-moment integration rule

Initialize $\boldsymbol{\xi}$
for each meta-training step **do**
  Sample $\mathcal{T}_e \sim p_{\mathrm{task}}$ with $\mathcal{D}_{\mathrm{tr},e}$ and $\mathcal{D}_{\mathrm{val},e}$
  Compute $\boldsymbol{\phi}_e$ from $\mathcal{T}_e$ and $\mathcal{D}_{\mathrm{tr},e}$; set $\beta_{\mathrm{R}} \leftarrow \mathrm{Temp}(\boldsymbol{\phi}_e; \boldsymbol{\xi})$
  Reset $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$ to $(\mu_{\boldsymbol{\omega},e}^{\mathrm{prior}}, \Sigma_{\boldsymbol{\omega},e}^{\mathrm{prior}})$
  for $k = 1, 2, \ldots, N_{\mathrm{tr},e}$, in record order **do**
    $(\mu_{\mathbf{a}_{e,k}}^{\mathrm{pred}}, \Sigma_{\mathbf{a}_{e,k}}^{\mathrm{pred}}, \mathcal{C}_{e,k}^{\mathrm{fwd}}) \leftarrow$ NetworkForwardMoments$(\mathbf{x}_{e,k}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$
    Compute $(\mu_{\mathbf{a}_{e,k}}^{\mathrm{tar}}, \Sigma_{\mathbf{a}_{e,k}}^{\mathrm{tar}})$ as the first two moments of (28)
    $(\mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}) \leftarrow$ NetworkBackwardMoments$(\mu_{\mathbf{a}_{e,k}}^{\mathrm{tar}}, \Sigma_{\mathbf{a}_{e,k}}^{\mathrm{tar}}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}}, \mathcal{C}_{e,k}^{\mathrm{fwd}}, \{\mathbf{L}_s = \mathbf{I}_{d_s}\}_{s\in\mathcal{I}}; \emptyset)$
  end for
  $\mathcal{L}_{\mathrm{val}} \leftarrow 0$
  for $(\mathbf{x}_{e,k}, \mathbf{s}_{e,k}) \in \mathcal{D}_{\mathrm{val},e}$ **do**
    Obtain $\mu_{\mathbf{a}_{e,k}}^{\mathrm{pred}}$ from NetworkForwardMoments$(\mathbf{x}_{e,k}, \mu_{\boldsymbol{\omega}}, \Sigma_{\boldsymbol{\omega}})$
    $\mathcal{L}_{\mathrm{val}} \leftarrow \mathcal{L}_{\mathrm{val}} - N_{\mathrm{val},e}^{-1} \mathsf{Q}^{\pi_{\mathrm{ref}}}(\mathbf{s}_{e,k}, \mu_{\mathbf{a}_{e,k}}^{\mathrm{pred}})$
  end for
  Perform a gradient step on $\boldsymbol{\xi}$ using $\mathcal{L}_{\mathrm{val}}$, differentiating through the ordered moment updates
end for
return $\boldsymbol{\xi}$

F.1 Example Task Descriptor for Temperature Meta-Learning

Consider an auxiliary discounted linear–quadratic–Gaussian (LQG) task to define ϕ. Each environment has state-transition matrix $\mathbf{A}_{\mathrm{dyn}}$, control matrix $\mathbf{B}_{\mathrm{ctl}}$, state-cost $\mathbf{Q}_{\mathrm{state}}$, action-cost $\mathbf{R}_{\mathrm{act}}$, discount factor γ_{R}, and process-noise covariance $\mathbf{I}_2$. A sampled stabilizing gain $\mathbf{K}_{\mathrm{ref}}$ defines the state-only reference policy $\pi_{\mathrm{ref}}(\mathbf{a} \mid \mathbf{s}) = \mathcal{N}(\mathbf{K}_{\mathrm{ref}}\mathbf{s}, \mathbf{I}_2)$. We draw N_{tr} training states and N_{val} validation states from $\mathrm{Uniform}([-2, 2]^2)$. Each training state contributes one reference action $\mathbf{a}_{\mathrm{ref},k}$ to $\boldsymbol{\phi}$; inner adaptation uses the states themselves.

All features use the training split. Define $\delta_k^{\mathrm{adv}} = \mathsf{Adv}^{\pi_{\mathrm{ref}}}(\mathbf{s}_k, \mathbf{a}_{\mathrm{ref},k})$. Let $\mathrm{IQR}_{\mathrm{adv}}$ and σ_{adv} be the interquartile range and population standard deviation of $\{\delta_k^{\mathrm{adv}}\}_{k=1}^{N_{\mathrm{tr}}}$. Let $\mathsf{V}_{\mathrm{ref},k} = \mathsf{V}^{\pi_{\mathrm{ref}}}(\mathbf{s}_k)$ have mean $\overline{\mathsf{V}}_{\mathrm{ref}}$ and population standard deviation $\sigma_{\mathsf{V},\mathrm{ref}}$. Define weights

$$\varpi_k = \exp\left(\frac{\delta_k^{\mathrm{adv}}}{\mathrm{IQR}_{\mathrm{adv}}}\right). \tag{85}$$

The six-dimensional descriptor is

$$\boldsymbol{\phi} = \begin{bmatrix} \log(\mathrm{IQR}_{\mathrm{adv}}) \\ \log(\sigma_{\mathrm{adv}}) \\ N_{\mathrm{tr}}^{-1} \sum_{k=1}^{N_{\mathrm{tr}}} \mathbb{I}\{\delta_k^{\mathrm{adv}} > 0\} \\ \left(\sum_{k=1}^{N_{\mathrm{tr}}} \varpi_k\right)^2 / \left(N_{\mathrm{tr}} \sum_{k=1}^{N_{\mathrm{tr}}} \varpi_k^2\right) \\ \mathrm{sign}(\overline{\mathsf{V}}_{\mathrm{ref}}) \log(1 + |\overline{\mathsf{V}}_{\mathrm{ref}}|) \\ \log\bigl(1 + \sigma_{\mathsf{V},\mathrm{ref}} / |\overline{\mathsf{V}}_{\mathrm{ref}}|\bigr) \end{bmatrix}. \tag{86}$$

Here $\mathbb{I}\{\cdot\}$ denotes the indicator. It records advantage scale and spread, the positive-advantage fraction, normalized effective sample size at $\beta_{\mathrm{R}} = \mathrm{IQR}_{\mathrm{adv}}$, and value level and heterogeneity.

G Detailed Simulation Settings

Transformer architecture. Both experiments use a decision transformer with one attention layer, two heads, context length 4, hidden dimension 8, and feed-forward dimension 32. The pretrained model and adaptation data are obtained from the supplied MPC-generated data. Inputs and actions use the pretrained model's normalization.

Sequential adaptation under environment drift. For Figure 1a, the transformer is pretrained at pendulum mass 1 and updated sequentially as the mass increases by 0.01 per sample. Laplace LoRA and SWAG fine-tune on the 25 most recent samples. Each curve aggregates ten random seeds. Where variability is nonzero, shaded regions show 95% confidence intervals for the mean, computed using the two-sided Student-t critical value with nine degrees of freedom.

Reward-guided adaptation. For Figure 1b, the pretrained transformer and adaptation data correspond to mass 1, while evaluation uses mass 2. Pendulum length is 1, gravity is 10, and the simulation time step is 0.05. Each episode contains 200 steps, with torque limited to $[-2, 2]$. We compare the proposed method with meta-learned action tilting, the same method with fixed action tilting, and the pretrained transformer without adaptation. The transformer is reset to the same prior before each adaptation stream and sample budget. The temperature network remains frozen during evaluation.

Temperature and action targets. We report the normalized temperature $\bar{\beta} = \beta_{\mathrm{R}}/S_A$, where S_A is the interquartile range of the training-record advantages, with a numerical floor. The fixed-temperature comparison uses $\bar{\beta} = 1000$. Thus the coefficient multiplying action values in the exponential tilt is $\alpha = 1/(\bar{\beta} S_A)$.

Action values are approximated by finite-horizon grid dynamic programming under the mass-2 dynamics, maximizing smooth upright occupancy during the final 50 steps. The grid has 256 angle bins, 129 angular-velocity bins, and 41 candidate torques. These values use optimal continuation and are not exact values of the pretrained behavior policy.

Bounded numerical quadrature reweights the current Gaussian action prediction by $\exp(\alpha Q)$. The target mean uses a conservative step

$$\mu_a^{\mathrm{tar}} = \mu_a^{\mathrm{pred}} + 0.05 \tanh\left(\frac{\mu_a^{\mathrm{tilt}} - \mu_a^{\mathrm{pred}}}{0.05}\right),$$

followed by clipping to the supported normalized-action range.

Temperature meta-training and selection. The temperature network is trained on 32 adaptation tasks for 20 epochs, with 200 sequential adaptation updates per task. Its objective is a smooth sustained-upright rollout score. Meta-gradients are estimated by centered finite differences in $\log \alpha$, using perturbations of ± 0.15 and common episode starts for the two perturbed policies. Each training estimate uses 20 episode starts. This rollout-based objective differs from the held-out action-value objective in Section 5.2.

Checkpoint selection uses eight validation adaptation streams and 50 shared episode starts. Selection prioritizes exact success rate, then final-window upright fraction, and then return; the selected checkpoint is meta-update 448. Only one independent meta-training run is used.

Sample budgets and success metric. Adaptation is evaluated at budgets of 100, 200, 300, and 400 ordered records. Each budget starts from the common pretrained prior.

An episode succeeds if the absolute wrapped pendulum angle is below 0.2 radians for at least 40 of the final 50 steps. Validation uses episode seeds 900000–900049. Test evaluation uses twelve separate adaptation streams and episode seeds 910000–910099. Within each split, all methods and budgets share the evaluation starts.

Aggregation and uncertainty. Success is averaged across adaptation streams and episode starts. For Figure 1b, pointwise 95% bootstrap intervals are computed from 5000 resamples of the episode-start indices. Each resampled index retains the results from all adaptation streams, preserving their shared-start dependence. The interval endpoints are the 2.5th and 97.5th percentiles.

These intervals are conditional on the fixed adaptation streams and trained temperature network; they do not quantify variability across independent meta-training runs. The unchanged-policy and learned-temperature results are reused from the corresponding saved evaluations, while the fixed-temperature arm is newly evaluated. The test cases have been inspected during experiment development, so these curves characterize the recorded setting rather than constitute a fresh, untouched confirmation experiment.