Generative Bayesian Hyperparameter Tuning

Hedibert Lopes
Insper

Nick Polson
Chicago Booth
University of Chicago

Vadim Sokolov*
Department of Systems Engineering
and Operations Research
George Mason University

First Version: 20 January 2025
This Version: April 8, 2026

Abstract

Choosing regularization hyper-parameters is a core task in statistical and machine-learning practice. Standard approaches include cross-validation, which is computationally expensive because the model must be retrained for every candidate value, and marginal-likelihood methods, which provide analytic tuning criteria but are limited to specific model classes. We develop a generative approach to hyper-parameter tuning that combines two ideas: the weighted Bayesian bootstrap (WBB), which converts fast optimization into approximate posterior draws by randomly reweighting the training objective, and neural-network-based amortization, which learns a transport map (generator) from hyper-parameters and bootstrap weights to the corresponding fitted model. Once trained, the generator produces fitted parameters for any new hyper-parameter value via a single forward pass, replacing the expensive retraining loop. We show how this viewpoint connects to classical generalized cross-validation and the generative multi-purpose sampler, and we note how auxiliary-variable representations can extend the same template to non-Gaussian losses such as quantile regression.

Keywords: Amortized inference; Bayesian bootstrap; Cross-validation; Hyperparameter optimization; Transport map; Weighted *M*-estimation.

*Hedibert Lopes is Professor of Statistics and Econometrics at Insper: hlopes@insper.edu.br. Nick Polson is Professor of Econometrics and Statistics at Chicago Booth, University of Chicago: ngp@chicagobooth.edu. Vadim Sokolov is Associate Professor at Volgenau School of Engineering, George Mason University: vsokolov@gmu.edu.

1 Introduction

Hyper-parameters such as regularization strengths, robustness or quantile levels, and architectural or optimization settings have an outsized impact on predictive performance and calibration. Several families of methods exist for choosing these hyper-parameters, each with characteristic trade-offs.

Cross-validation (CV) is the most widely used approach: the data are split into training and validation folds, the model is retrained for each candidate hyper-parameter value, and the value that minimizes validation error is selected [Stone, 1974]. CV is general-purpose but computationally expensive, since the full training procedure must be repeated for every candidate value and every fold.

Marginal-likelihood and empirical Bayes methods avoid repeated retraining by deriving analytic tuning criteria [Berger, 1985]. Generalized cross-validation (GCV) is a classical example: for linear smoothers such as ridge regression, the leave-one-out prediction error can be estimated from a single fit using the trace of the hat matrix [Golub et al., 1979]. These methods are fast but restricted to model classes where the optimizer and the tuning criterion admit closed-form expressions [Golub et al., 1979].

Bayesian approaches place a prior on the hyper-parameter and compute or approximate the posterior, providing a principled framework for model comparison. In practice, however, posterior computation typically requires MCMC over the joint space of parameters and hyper-parameters, which is expensive when the model is large or when the conditional distributions mix slowly [Gamerman and Lopes, 2006]. Variational approximations offer an alternative but require careful design of the variational family [Blei et al., 2017]. A concrete example is estimating the degrees-of-freedom parameter ν of a Student-t distribution: the profile likelihood is flat in ν, improper priors lead to improper posteriors, and MCMC mixes slowly because the marginal likelihood $p(y \mid \nu)$ is nonlinear in ν [Fonseca et al., 2008].

This paper develops a generative approach to hyper-parameter tuning that combines two ideas [Nareklishvili et al., 2024, Polson and Sokolov, 2025b, Nareklishvili et al., 2023b]. First, the weighted Bayesian bootstrap (WBB) of Newton et al. [2021] converts fast optimization into approximate posterior draws by randomly reweighting the training objective — each draw of the random weights produces a new fitted model, and the distribution over these fitted models approximates the Bayesian posterior for a fixed hyper-parameter value. Second, a parametric map (generator) learns the mapping from hyper-parameters and bootstrap weights to the corresponding fitted model, so that evaluating the fit at new hyper-parameter values or new bootstrap draws requires only a forward pass through the generator rather than retraining from scratch. The generator can be a simple linear map (for classical models) or a neural network (for deep learning), and is related in spirit to hyper-networks [Ha et al., 2017, Bhadra et al., 2024] but grounded in the WBB's posterior-approximation properties rather than in variational objectives.

Viewed as a hyper-parameter search strategy, the learned generator can rapidly evaluate predictive criteria over hyper-parameter grids, serving as a drop-in replacement for expensive CV loops and complementing Bayesian optimization [Snoek et al., 2012, Bergstra and Bengio, 2012, Baker et al., 2022] and gradient-based or bilevel approaches

[Maclaurin et al., 2015, Franceschi et al., 2018]. Viewed as approximate Bayesian inference, the same generator yields fast sampling of parameter draws across hyper-parameter settings by resampling the WBB weights at test time. We illustrate both uses across three settings: ridge regression (where GCV provides a benchmark), Student-t degree-of-freedom estimation (where MCMC is the standard but expensive alternative), and MNIST classification (where the generator amortizes neural-network tuning).

The work closest to ours is the Generative Multi-purpose Sampler (GMS) of Shin et al. [2023, 2024], which constructs a generator that outputs solutions of weighted penalized M-estimators across bootstrap and cross-validation weight patterns and across tuning parameters. A central contribution of GMS is its training objective: instead of regressing onto precomputed optimizers (which would require many expensive inner solves), the generator is trained by minimizing the integrated weighted objective value directly, so that generator training and M-estimation co-evolve via a single optimization loop. GMS also demonstrates how this enables computationally intensive procedures (iterated bootstrap, bootstrapped CV, solution paths) at scale.

Our contribution is to place these ideas, together with GCV and the WBB, into a single paradigm suitable for modern models trained by stochastic gradient descent (SGD): a randomized weighted inner objective, an outer tuning criterion defined as an expectation over the induced randomness, and an optional transport-map generator to amortize repeated optimization. In this view, GCV is the deterministic linear-smoother special case where both the optimizer map and the outer risk surrogate are analytic, while GMS is the weighted M-estimation special case where the generator is trained by criterion minimization.

Our contribution is threefold. First, we develop a unified three-component template — inner weighted objective, outer tuning criterion, and optional transport-map generator — that recovers GCV and GMS as special cases and extends to SGD-trained models. Second, we demonstrate the template on estimating the degrees-of-freedom parameter ν of a Student-t distribution [Fonseca et al., 2008], a setting where the profile likelihood is flat and MCMC mixes slowly; the generator recovers the profile MLE's ν selection and simultaneously provides WBB uncertainty summaries for the location and scale parameters. Third, we illustrate criterion-based generator training and amortized tuning in controlled ridge-regression and MNIST experiments.

The remainder of the article is organized as follows. Section 2 develops the generative Bayesian hyper-parameter tuning framework, including motivation (Section 2.1), the method (Section 2.2), an illustration (Section 2.3), and extensions (Section 2.4). Section 3 presents empirical illustrations: ridge regression (Section 3.1), Student-t degree-of-freedom estimation (Section 3.2), and MNIST classification (Section 3.3). Section 4 concludes with a discussion. Code to reproduce all experiments is available at `https://github.com/VadimSokolov/Generative-Bayesian-Hyperparameter-Tuning`.

2 Generative Bayesian Hyperparameter Tuning

This section develops the generative Bayesian hyper-parameter tuning framework. Section 2.1 establishes the regularization–MAP duality that motivates the approach. Sec-

tion 2.2 presents the method: the WBB-weighted inner objective, the transport-map generator, and the outer tuning criterion. Section 2.3 gives an illustration on weighted ridge regression comparing supervised and criterion-based generator training. Section 2.4 discusses extensions and connections to data augmentation and quantile regression.

2.1 Motivation and setup

A Bayesian approach to hyper-parameter tuning is conceptually straightforward but computationally difficult (often requiring MCMC). A central link is the duality between regularization and maximum a posteriori (MAP) inference: we treat λ as a hyper-parameter with prior density $\lambda \sim p(\lambda)$.

Many problems in machine learning involve an output y_i with x_i a high-dimensional input. Let $\ell(y_i, f_\theta(x_i))$ denote a per-observation data-fit term (e.g. negative log-likelihood or a surrogate loss), ϕ a regularizer, and $\theta = (\theta_1, \ldots, \theta_p)$ the model parameters. A generic regularized objective takes the form

$$\mathcal{L}(\theta; \lambda) = \sum_{i=1}^{n} \ell\big(y_i, f_\theta(x_i)\big) + \lambda \sum_{j=1}^{p} \phi(\theta_j).$$

Throughout, we treat the data $\{(x_i, y_i)\}_{i=1}^n$ as fixed and study how the optimizer varies with hyper-parameters. We write $\ell(y_i \mid \theta)$ as shorthand for $\ell(y_i, f_\theta(x_i))$ when x_i is implicit. Hyper-parameters can enter either through the regularization term (e.g. λ) or through auxiliary/nuisance parameters in the data-fit term (write ℓ_η for a loss indexed by η).

The regularized estimator is the mode of

$$p(y \mid \theta) \propto \exp\left\{ -\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)) \right\}, \qquad p(\theta \mid \lambda) \propto \exp\{-\lambda\phi(\theta)\},$$
$$p(\theta \mid y, \lambda) \propto \exp\left\{ -\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)) - \lambda\phi(\theta) \right\}.$$

When ℓ is a negative log-likelihood this is standard MAP inference; for a generic surrogate loss the same construction defines a Gibbs (generalized Bayes) posterior [Bissiri et al., 2016, Datta et al., 2025].

2.2 The method

We now state the method, which subsumes (i) classical analytic tuning rules such as GCV for linear smoothers [Golub et al., 1979] and (ii) modern generator-based weighted M-estimation as in GMS [Shin et al., 2023, 2024]. Three components interact: a randomized inner training objective whose weights make the procedure Bayesian, a transport-map generator that amortizes repeated optimization, and an outer tuning criterion that selects hyper-parameters.

Let $h = (\lambda, \eta)$ denote a vector of hyper-parameters: λ is a regularization strength and η collects any additional hyper-parameters that index the data-fit term (e.g. robustness/quantile parameters, temperature parameters, augmentation strengths, etc.). The key ingredient that converts a single optimization into approximate posterior inference is a random weight vector $\omega \sim \pi(\omega)$. The idea originates with the weighted Bayesian bootstrap (WBB) of Newton et al. [2021], whose weighted objective is

$$\mathcal{L}_\omega(\theta, \lambda, y) = \sum_{i=1}^{n} \omega_i \, \ell(y_i \mid \theta) + \lambda \omega_0 \phi(\theta), \qquad \phi(\theta) = \sum_{j=1}^{p} \phi_j(\theta_j),$$

where $\omega_i = \ln(1/u_i)$ with $u_i \overset{\text{iid}}{\sim} U(0,1)$ are random per-observation weights and $\omega_0 = \ln(1/u_0)$ with $u_0 \sim U(0,1)$, independent of $\{\omega_i\}$, weights the regularizer. This induces a map $\omega \mapsto \theta^*_\omega = \arg\min_\theta \mathcal{L}_\omega$, and the conjecture of Newton et al. [2021] is that the induced distribution of θ^*_ω (with data fixed) approximates the Bayesian posterior $p(\theta \mid y, \lambda)$.

We adopt a slightly modified WBB-style objective with deterministic penalty weight $\omega_0 \equiv 1$ and a $1/n$ factor in the data-fit term (a convenience for SGD-based training). In Newton et al.'s full construction, dividing by ω_0 shows that the optimizer depends on ω only through the ratios $\tilde{\omega}_i = \omega_i/\omega_0$; fixing $\omega_0 \equiv 1$ is therefore a simplification, not an equivalence, that keeps the penalty term deterministic while the data-fit weights ω_i provide the randomization. Common choices for $\pi(\omega)$ include the WBB exponential weights above, the Bayesian bootstrap $v \sim \text{Dirichlet}(\iota_n)$, $\omega = nv$ [Rubin, 1981] (where ι_n denotes the n-vector of ones), and the frequentist bootstrap $\omega \sim \text{Multinomial}(n, \iota_n/n)$. The deterministic case $\omega \equiv \mathbf{1}$ is also included as a special case. From a broader perspective, randomized/tempered objectives connect naturally to generalized Bayes and power posterior ideas [Bissiri et al., 2016, Friel and Pettitt, 2008]. The weighted objective is

$$L(\theta; \omega, h) \;=\; \frac{1}{n} \sum_{i=1}^{n} \omega_i \, \ell_\eta(y_i \mid \theta) \;+\; \lambda \, \phi(\theta), \qquad \hat{\theta}(\omega, h) \in \arg\min_\theta L(\theta; \omega, h). \tag{1}$$

In neural networks, $\hat{\theta}(\omega, h)$ is computed approximately by SGD on reweighted mini-batches; in convex problems it may be available in closed form.

To avoid re-solving the inner optimization for every candidate hyper-parameter value, we introduce a parametric transport map (generator) g_φ intended to approximate the optimizer:

$$g_\varphi(\omega, h) \approx \hat{\theta}(\omega, h).$$

This map can be trained in two ways: (A) by supervised regression to precomputed optimizers $\hat{\theta}(\omega^{(b)}, h^{(b)})$, or (B) by direct minimization of the criterion value (GMS-style) by plugging g_φ into $L(\cdot; \omega, h)$ and optimizing φ by backpropagation. The generator admits a natural generative Bayesian interpretation [Polson and Sokolov, 2025b, Nareklishvili et al., 2023a]: rather than sampling from a posterior via MCMC, one samples weights $\omega \sim \pi(\omega)$ and pushes them through $g_\varphi(\omega, h)$; the distribution of the output approximates the posterior over θ for fixed h, to the extent that g_φ accurately approximates $\hat{\theta}$ and under the WBB posterior-approximation property. This "generative WBB" viewpoint simultaneously supports hyper-parameter tuning and uncertainty quantification:

the same trained generator that enables fast evaluation of tuning criteria also produces approximate posterior draws by resampling ω. In the SGD setting, the outer criterion $\mathcal{C}$ is typically instantiated as a proper scoring rule $\mathcal{R}$ on a held-out validation set $\mathcal{D}_{\text{val}}$ (e.g. negative log-likelihood), promoting both accuracy and calibration [Polson et al., 2024].

The outer goal is to choose h to optimize a criterion $\mathcal{C}$ that measures out-of-sample quality. A general template is

$$\min_h \; J(h) \;=\; \mathbb{E}_{\omega\sim\pi(\omega)}\Big[\mathcal{C}\big(\hat{\theta}(\omega,h),h\big)\Big] \;\approx\; \frac{1}{M}\sum_{m=1}^{M}\mathcal{C}\big(g_{\hat{\varphi}}(\omega^{(m)},h),h\big), \tag{2}$$

where $\mathcal{C}$ might be a validation negative log-likelihood, a proper scoring rule, a risk estimate, or a proxy for marginal-likelihood objectives. The approximation on the right is the computational payoff: once $g_{\hat{\varphi}}$ is trained, we can evaluate (and often differentiate) $J(h)$ without repeatedly re-running SGD for each h. A common simplification sets $M = 1$ with unit weights $\omega \equiv \mathbf{1}$, reducing the outer criterion to $\mathcal{C}(g_{\hat{\varphi}}(\mathbf{1},h),h)$; this amounts to tuning on the point estimate and is the variant used in our Student-t and MNIST experiments. The full Monte Carlo average over ω is used when predictive averaging or Bayesian model comparison is desired. Uncertainty summaries follow immediately: after tuning $(\hat{\lambda},\hat{\eta})$, we draw $\omega^{(m)} \sim \pi(\omega)$, form $\theta^{(m)} = g_{\hat{\varphi}}(\omega^{(m)},\hat{\lambda},\hat{\eta})$, and approximate the posterior predictive by Monte Carlo averaging,

$$p(y \mid x,\mathcal{D},\hat{\lambda},\hat{\eta}) \;\approx\; \frac{1}{M}\sum_{m=1}^{M} p(y \mid x,\theta^{(m)}).$$

This yields predictive means, variances, and quantiles, and plays a role similar to MC-dropout [Gal and Ghahramani, 2016] and deep Gaussian processes [Schultz and Sokolov, 2022], but is aligned with the "random weights + optimization" template used throughout this paper.

The method recovers two important special cases. In ridge regression (or any linear smoother), the optimizer is explicit and (1) is deterministic: take $\omega \equiv \mathbf{1}$ and η empty, with squared-error loss and quadratic penalty. Then $\hat{\theta}(h)$ is available in closed form and the fitted values are linear in y via a hat matrix $A(\lambda)$. Choosing $\mathcal{C}$ in (2) as the generalized cross-validation surrogate yields the classical GCV rule $\hat{\lambda} \in \arg\min_\lambda V(\lambda)$ [Golub et al., 1979]; no learned generator is needed. In weighted M-estimation, (1) matches the GMS setting of Shin et al. [2023, 2024]: ω is random (bootstrap/Dirichlet-type weights), ℓ_η is a loss indexed by η, and λ tunes regularization. GMS emphasizes the criterion-based training route (mode B), training g_φ by minimizing the weighted objective value directly rather than matching precomputed optimizers. The outer criterion (2) can then be used to tune h for prediction or for approximate Bayesian sampling.

Algorithm 1 in Appendix A summarizes the full procedure.

2.3 An illustration

We illustrate the method on a weighted ridge regression example, closely following Shin et al. [2023], to make explicit why criterion-based generator training can be more efficient than matching precomputed optimizers.

Consider a linear model with fixed design matrix $X \in \mathbb{R}^{n \times p}$ and response $y \in \mathbb{R}^n$. Let $\omega \in \mathbb{R}^n_+$ be weights and $W = \mathrm{diag}(\omega)$. The weighted ridge objective is

$$L(\theta; \omega, \lambda) \;=\; \frac{1}{n}\|W^{1/2}(y - X\theta)\|_2^2 + \lambda\|\theta\|_2^2,$$

with closed-form optimizer

$$\hat{\theta}(\omega, \lambda) \;=\; \Big(X^\top W X + n\lambda I\Big)^{-1} X^\top W y.$$

Thus, in this toy setting the "true" map $(\omega, \lambda) \;\mapsto\; \hat{\theta}(\omega, \lambda)$ exists explicitly, and bootstrap/CV-type procedures correspond to repeatedly evaluating this map for many draws of ω and many values of λ.

Let $g_\varphi(\omega, \lambda)$ be a parametric map intended to approximate $\hat{\theta}(\omega, \lambda)$. In the toy setting we use a linear generator $g_\varphi(\omega, \lambda) = Az(\omega, \lambda)$ where z are features of (ω, λ); in deep-learning settings g_φ would be a neural network. A classical supervised approach generates a training set $\{(\omega^{(b)}, \lambda^{(b)}, \hat{\theta}^{(b)})\}_{b=1}^B$ by repeatedly computing $\hat{\theta}^{(b)} = \hat{\theta}(\omega^{(b)}, \lambda^{(b)})$, then fits φ by regression (e.g. squared loss). This is accurate when B is very large, but becomes expensive when computing $\hat{\theta}^{(b)}$ is costly.

The GMS approach instead trains the generator by minimizing the integrated objective value, without ever computing $\hat{\theta}(\omega, \lambda)$ labels:

$$\min_\varphi \;\mathbb{E}_{(\omega,\lambda)\sim P}\Big[L\big(g_\varphi(\omega, \lambda); \omega, \lambda\big)\Big], \tag{3}$$

where P is a user-chosen distribution over weights and tuning parameters. In practice, the expectation is approximated by Monte Carlo samples inside SGD: on each iteration one draws fresh (ω, λ) and backpropagates through $L(g_\varphi(\omega, \lambda); \omega, \lambda)$.

Supervised training controls $\|g_\varphi(\omega, \lambda) - \hat{\theta}(\omega, \lambda)\|$ only on the sampled training set, and can overfit if B is limited. By contrast, the criterion-based loss (3) supplies a task-relevant signal at essentially unlimited "training inputs" (ω, λ) because drawing new (ω, λ) is cheap; the optimization of φ and the minimization of the statistical objective co-evolve in a single loop [Shin et al., 2023].

One way to formalize this intuition is to evaluate a population mismatch measure such as an integrated prediction loss,

$$\mathrm{IPL}(g_\varphi) \;=\; \mathbb{E}_{(\omega,\lambda)\sim P}\Big[\|g_\varphi(\omega, \lambda) - \hat{\theta}(\omega, \lambda)\|_2^2\Big],$$

which measures accuracy of the generator averaged over the weight/tuning distribution of interest. Even in settings where L is convex and has an explicit optimizer, optimizing (3) can reduce $\mathrm{IPL}(g_\varphi)$ efficiently because it provides gradients everywhere in (ω, λ)-space, whereas supervised fitting relies on a finite (and expensive) set of optimizer labels.

To make this contrast concrete, we implement the weighted ridge setup in R (script: `code/toy/toy_gms_ridge.R`) with $n = 200$ observations, $p = 20$ covariates drawn from an AR(1) design with autocorrelation $\rho = 0.3$, true coefficients $\theta_0 = (1, \ldots, 1, 0, \ldots, 0)^\top$ (5

nonzero), and noise standard deviation $\sigma = 1$. Observations are partitioned into $S = 25$ blocks, and random weights are drawn as $\omega \sim S \cdot \text{Dirichlet}(\mathbf{1}_S)$ (constant within blocks); λ is drawn log-uniformly on $[10^{-3}, 1]$. We compare two linear-generator training modes: (A) supervised fitting that uses B optimizer labels $\hat{\theta}(\omega^{(b)}, \lambda^{(b)})$ and (B) criterion-based fitting that uses M Monte Carlo draws (ω, λ) without computing any optimizer labels (mimicking the GMS advantage in problems where $\hat{\theta}$ requires an expensive solve). All results are averaged over 12 replications with $M_{\text{test}} = 300$ evaluation draws. Figure 1 compares the two training modes by IPL at equal training-time budgets; Table 1 gives representative IPL values at selected training budgets.

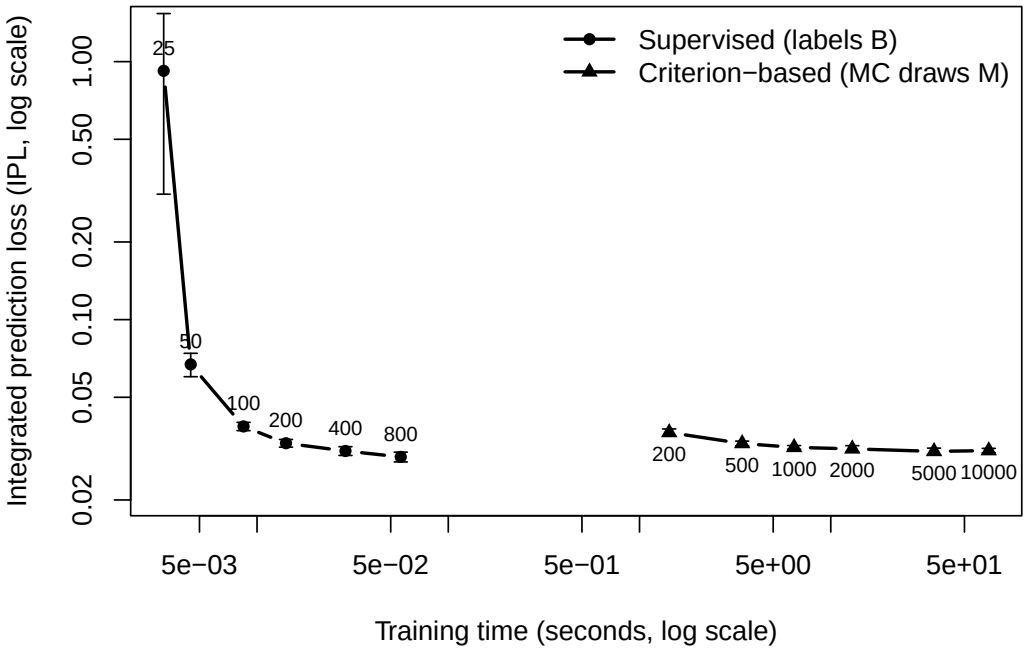

Figure 1: Toy weighted-ridge example. Integrated prediction loss (IPL, log scale) versus training time (log scale) for supervised match-to-optimizer training (circles, labeled by number of optimizer solves B) and criterion-based (GMS-style) training (triangles, labeled by number of Monte Carlo draws M). Both methods converge to similar IPL; the criterion-based method achieves this without computing any optimizer labels, though at higher wall-clock cost in this cheap-optimizer setting (each ridge solve is closed-form). The criterion-based advantage grows when each optimizer solve is expensive.

Table 1: Toy weighted-ridge example: integrated prediction loss (IPL) for a linear generator trained either by supervised matching to optimizer labels (B optimizer solves) or by criterion-based (GMS-style) training (M Monte Carlo draws). Values are means over 12 replications with Monte Carlo standard errors in parentheses.

	Training budget		
Method	B=50	B=200	B=800
Supervised match-to-optimizer	0.067 (0.003)	0.033 (0.001)	0.029 (0.001)
	M=500	M=2,000	M=10,000
Criterion-based (GMS-style)	0.033 (0.000)	0.032 (0.000)	0.031 (0.000)

2.4 Extensions and connections

The template developed above connects to several existing ideas and extends naturally to non-standard losses.

The idea of jointly updating parameters and hyper-parameters within a single iterative scheme has precedent in data augmentation methods. Polson and Scott [2011] show how ECME-style updates [Liu and Rubin, 1994] can jointly estimate parameters β and the regularization parameter for support vector machines. Writing ξ for their regularization strength (denoted ν in their paper; we use ξ here to avoid conflict with the degrees-of-freedom parameter ν used elsewhere in our paper), with L^α penalty and prior shape parameters a, b, their update for a model with k nonzero coefficients and prior scales σ_j is

$$(\xi^{-\alpha})^{(g+1)} = \frac{b + \sum_{j=1}^{k} |\beta_j^{(g+1)}/\sigma_j|^\alpha}{a - 1 + k/\alpha}.$$

In our framework, such closed-form hyper-parameter updates correspond to solving the outer problem (2) analytically when $\mathcal{C}$ has a conjugate form. More broadly, the GMS implementation of Shin et al. [2024], including an R package and PyTorch code, aligns directly with our inner problem (1).

Another useful extension is the "quantile trick" for turning quantile-regression (check) loss into a weighted least-squares form via a variance–mean Gaussian envelope [Polson and Scott, 2016]. Here u denotes an auxiliary scale variable in the envelope representation (distinct from the uniform draws used for WBB weights). For $q \in (0, 1)$ and residual r, define the (asymmetric) absolute/hinge loss

$$\rho_q(r) = |r| + (2q - 1)r,$$

which equals twice the standard check loss $\rho_q^{\text{std}}(r) = r\,(q - \mathbf{1}_{r<0})$; the factor of two does not affect the optimizer. Polson and Scott [2016] show that ρ_q admits a quadratic envelope in an auxiliary variable $u > 0$: the check loss equals $\inf_{u>0}$ of a quadratic form in r plus a function of u alone, and the conditional mode of u given r is

$$\hat{u}(r) = \text{sgn}(r)/r = 1/|r| \quad (r \neq 0);$$

in practice one regularizes via $\hat{u}_\varepsilon(r) = 1/\max(|r|, \varepsilon)$ for small $\varepsilon > 0$. This yields per-observation auxiliary weights $\tilde{w}_i = \hat{u}(r_i)$ and a shifted working response $z_i = y_i - (1 -$

$2q)/\tilde{w}_i$, so that the inner update becomes a weighted least-squares problem (plus the usual regularizer). This connects to our main framework: once a difficult loss can be expressed through auxiliary variables as a weighted least-squares inner step, it fits the same computational template as WBB. Moreover, the quantile level q plays the role of a tunable hyper-parameter; in the generator view one can treat (ω, q, λ) as inputs and learn an amortized map $(\omega, q, \lambda) \mapsto \hat{\theta}$, enabling rapid tuning over many q (and λ) values. We leave this extension to future work.

3 Empirical illustrations

This section presents three empirical illustrations. Section 3.1 considers ridge regression, where GCV and CV provide exact benchmarks for the amortized generator. Section 3.2 applies the template to Student-t degree-of-freedom estimation, a non-Gaussian setting where the profile likelihood is flat and MCMC is expensive. Section 3.3 demonstrates amortized tuning for a neural network on MNIST classification.

3.1 Ridge regression

As discussed in Section 2.2, ridge regression is the special case where all three components of our template are available in closed form. With hat matrix $A(\lambda) = X(X^\top X + n\lambda I)^{-1}X^\top$, the GCV criterion $V(\lambda) = n^{-1}\|y - A(\lambda)y\|^2/(1 - n^{-1}\mathrm{tr}\{A(\lambda)\})^2$ provides an analytic outer criterion [Golub et al., 1979]. This setting connects to modern discussions of double descent phenomena [Belkin et al., 2019, Polson and Sokolov, 2025a] and makes an ideal testbed for comparing the amortized generator against exact methods.

We simulate a ridge regression problem with $n = 300$ observations, $p = 80$ covariates drawn from an AR(1) design with autocorrelation $\rho = 0.2$, true coefficients $\theta_0 = (1, \ldots, 1, 0, \ldots, 0)^\top$ (8 nonzero), and noise standard deviation $\sigma = 1$. An independent test set of $n_{\text{te}} = 300$ is generated from the same distribution. We compare (i) GCV-selected λ, (ii) standard 5-fold CV, and (iii) an amortized "generator proxy" trained to map λ to coefficients and thereby approximate the CV curve without repeated refits across λ. To avoid data leakage, the generator is trained separately for each fold using only that fold's training data. The generator proxy is a linear map from $(1, \log\lambda, (\log\lambda)^2)$ to θ, deliberately simple to isolate the amortization idea.

Figure 2 shows the estimated CV curves; Table 2 reports the selected λ and test MSE. The amortized proxy selects a comparable regularization parameter with a modest increase in test MSE, illustrating the trade-off between amortization speed and approximation accuracy due to the limited capacity of the quadratic generator. The corresponding code is in `code/experiments/ridge_tuning_demo.R`.

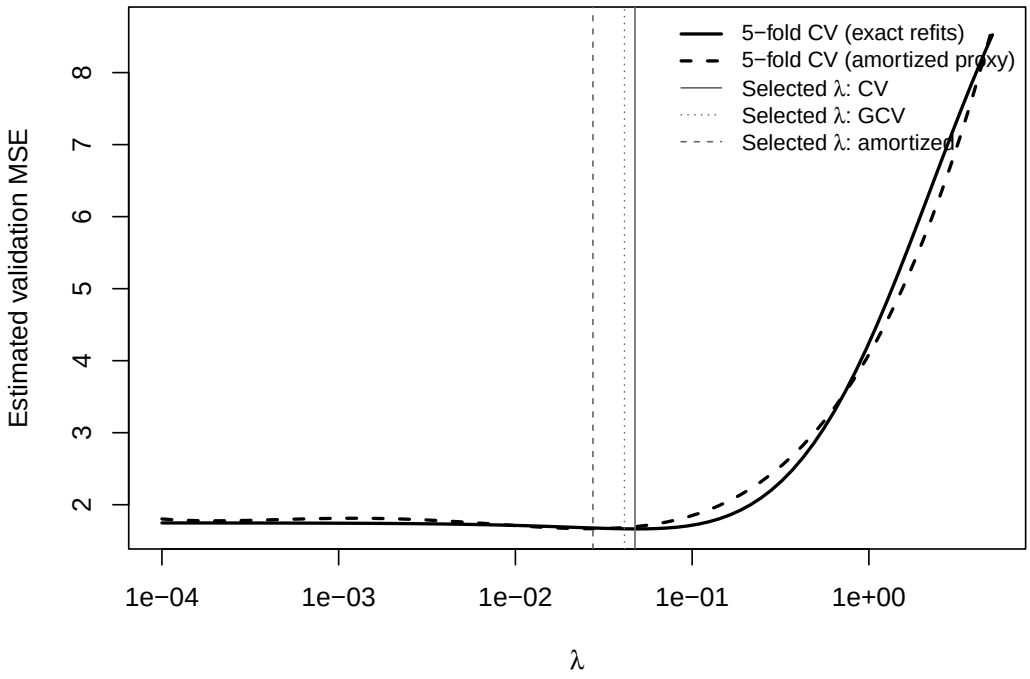

Figure 2: Ridge tuning illustration. Solid curve: 5-fold CV requiring repeated refits for each λ. Dashed curve: amortized generator proxy. Vertical lines mark the selected λ for each method: solid (CV), dotted (GCV), dashed (amortized proxy).

Table 2: Ridge tuning illustration (simulated data). We compare generalized cross-validation (GCV), K-fold cross-validation (CV), and a simple amortized generator proxy that approximates the CV curve without repeated refits across λ. Test MSE is computed on an independent hold-out test set.

Method	Selected λ	Test MSE
GCV	0.041	1.463
5-fold CV	0.047	1.450
Amortized CV proxy (refit at selected λ)	0.027	1.500

3.2 Student's errors

Estimating the degrees-of-freedom parameter ν of a Student-t distribution is a classical problem where standard methods encounter difficulties. The profile likelihood is flat in ν and may be strictly increasing as $\nu \to \infty$, so the MLE may not exist; improper priors on ν can lead to improper posteriors, while proper priors (e.g., exponential) may dominate the analysis [Fonseca et al., 2008]. Moreover, the marginal likelihood $p(y \mid \nu)$ is nonlinear in ν, making Metropolis–Hastings proposals difficult to tune.

Consider observations $y_1, \ldots, y_n$ from a location-scale t-distribution. The per-

observation negative log-likelihood for parameters $\theta = (\mu, \sigma)$ and fixed ν is

$$\ell(y_i \mid \mu, \sigma, \nu) = -\log \Gamma\left(\tfrac{\nu+1}{2}\right) + \log \Gamma\left(\tfrac{\nu}{2}\right) + \tfrac{1}{2}\log(\nu\pi\sigma^2) + \tfrac{\nu+1}{2}\log\left(1 + \frac{(y_i-\mu)^2}{\nu\sigma^2}\right).$$

A useful perspective is the scale-mixture-of-normals representation: $y_i \mid \tau_i \sim N(\mu, \sigma^2/\tau_i)$ with $\tau_i \sim \text{Gamma}(\nu/2, \nu/2)$, which integrates to the $t_\nu(\mu, \sigma^2)$ marginal. This representation connects the t-distribution to the auxiliary-variable framework and makes the dependence on ν enter through the mixing distribution.

In our template, ν plays the role of the hyper-parameter h, and $\theta = (\mu, \sigma)$ are the model parameters. The inner problem (1) becomes, for WBB block weights ω and fixed ν,

$$\hat{\theta}(\omega, \nu) = \arg\min_{(\mu,\sigma)} \frac{1}{n} \sum_{i=1}^{n} \omega_i \, \ell(y_i \mid \mu, \sigma, \nu),$$

solved by numerical optimization. The generator g_φ learns the map $(\omega, \nu) \mapsto (\hat{\mu}, \hat{\sigma})$; we use a linear generator with features $(1, \nu, \nu^2, \omega_1, \ldots, \omega_K)$ (where $K = 16$ is the number of WBB blocks), trained in supervised mode (A) on $B = 200$ optimizer labels drawn from random (ω, ν) pairs. The outer criterion (2) is the validation negative log-likelihood, evaluated over a grid $\nu \in \{1.5, 2.0, \ldots, 15.0\}$:

$$\hat{\nu} = \arg\min_{\nu} \sum_{i \in \mathcal{D}_{\text{val}}} \ell\big(y_i \mid g_\varphi(\mathbf{1}, \nu),\, \nu\big)\,,$$

where $g_\varphi(\mathbf{1}, \nu)$ is the generator evaluated at unit weights (point estimate). Once $\hat{\nu}$ is selected, WBB uncertainty summaries for (μ, σ) are obtained by resampling: draw $\omega^{(m)} \sim \pi(\omega)$, compute $\theta^{(m)} = g_\varphi(\omega^{(m)}, \hat{\nu})$, and examine the empirical distribution of $\{\theta^{(m)}\}_{m=1}^{M}$.

We generate $n_{\text{total}} = 500$ observations with $\nu_{\text{true}} = 4$, $\mu_{\text{true}} = 2$, and $\sigma_{\text{true}} = 1.5$, split into $n_{\text{train}} = 350$ and $n_{\text{val}} = 150$. Figure 3 shows, for one illustrative dataset, that the generator closely tracks the profile MLE validation curve, with both methods finding the minimum near $\nu = 4$.

With $n = 350$ and heavy tails ($\nu = 4$), the sample estimates of μ and σ vary substantially across realizations, so a single dataset can be misleading. Figure 4 shows the results across 20 independent replications. The location and scale estimates are centered on the true values for both methods: $\hat{\mu}_{\text{MLE}} = 2.02 \pm 0.10$, $\hat{\sigma}_{\text{MLE}} = 1.50 \pm 0.11$ versus $\hat{\mu}_{\text{gen}} = 2.02 \pm 0.10$, $\hat{\sigma}_{\text{gen}} = 1.48 \pm 0.12$ (Figure 4, middle and right panels). The selected ν has high variance and is upward-biased for both methods ($\hat{\nu}_{\text{MLE}} = 5.3 \pm 2.9$, $\hat{\nu}_{\text{gen}} = 5.2 \pm 2.4$; Figure 4, left panel), reflecting the asymmetry of the profile likelihood: steep for small ν but flat as $\nu \to \infty$ [Fonseca et al., 2008]. The average WBB posterior standard deviation is 0.089 for μ (versus cross-replicate sd of 0.10) and 0.075 for σ (versus 0.12); the WBB is slightly under-dispersed for σ, consistent with its first-order asymptotic justification [Newton et al., 2021]. Table 3 summarizes the cross-replicate results. The code is in `code/experiments/t_dist_tuning.R`.

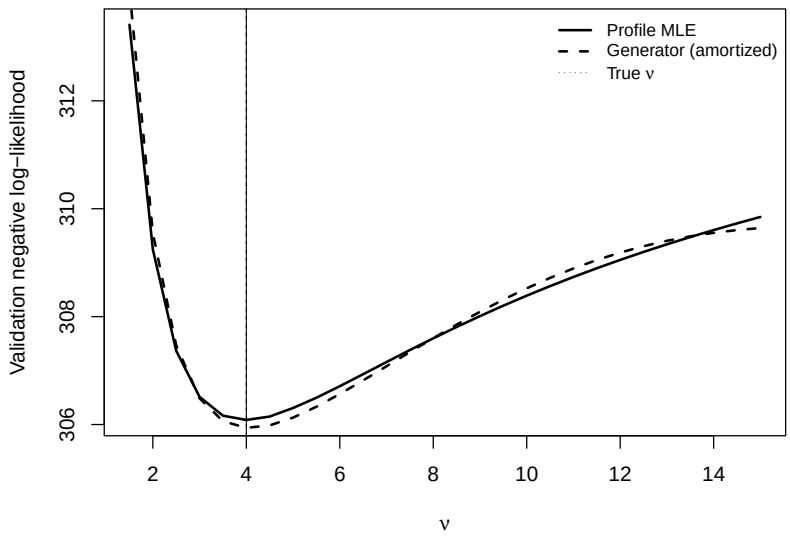

Figure 3: Student-t tuning, single dataset. Validation negative log-likelihood as a function of ν for profile MLE (solid) and the amortized generator (dashed); the vertical line marks the true $\nu = 4$. Both methods find the minimum near $\nu = 4$ on this dataset.

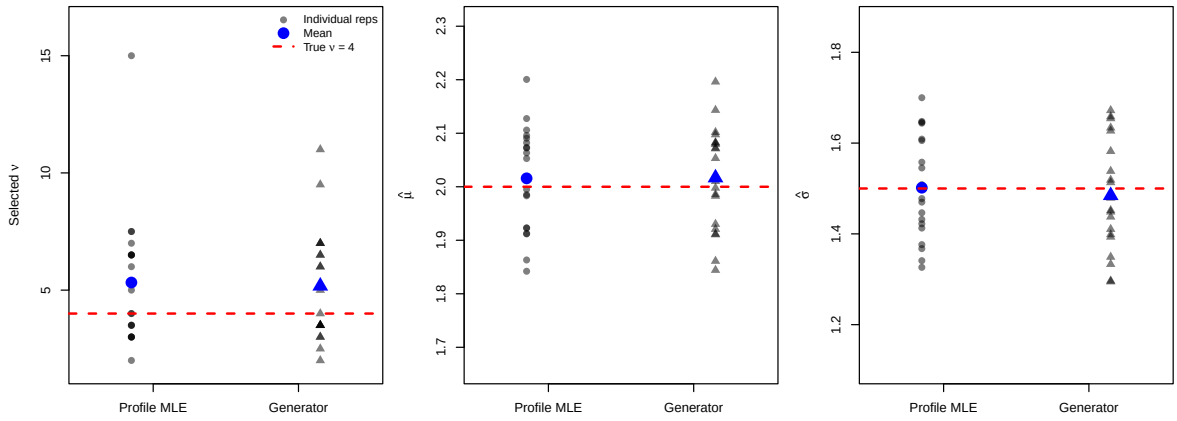

Figure 4: Student-t tuning, 20 replications. Each point is one independent dataset ($n = 500$, $\nu_{\text{true}} = 4$, $\mu_{\text{true}} = 2$, $\sigma_{\text{true}} = 1.5$). Blue markers show means. Left: selected ν (upward-biased due to flat profile likelihood). Middle and right: $\hat{\mu}$ and $\hat{\sigma}$ are centered on the true values (red dashed lines) for both profile MLE and the generator.

Table 3: Student-t tuning results over 20 replications ($n_{\text{train}} = 350$, $n_{\text{val}} = 150$). True parameters: $\nu = 4$, $\mu = 2.000$, $\sigma = 1.500$. Values are means $\pm$ standard deviations across replications.

Method	Selected ν	$\hat{\mu}$	$\hat{\sigma}$
Profile MLE	5.3 ± 2.9	2.02 ± 0.10	1.50 ± 0.11
Generator (amortized)	5.2 ± 2.4	2.02 ± 0.10	1.48 ± 0.12
WBB posterior mean ($\pm$ avg sd)	—	2.02 ($\pm$ 0.089)	1.49 ($\pm$ 0.075)

3.3 Deep learning

To demonstrate the generative WBB template in a deep-learning setting, we apply it to MNIST classification. The target model is a two-layer MLP ($784 \to 64 \to 10$, ReLU activation, cross-entropy loss) with 50,890 parameters. The generator (hyper-network) is a three-layer MLP (hidden dimension 128) that maps a $(1 + K)$-dimensional input — a normalized $\log_{10} \lambda$ concatenated with $K = 16$ block bootstrap weights ω — to the full parameter vector θ. Training uses the criterion-based objective (mode B) over 5 epochs with Adam (learning rate 10^{-3}, batch size 128), minimizing the WBB-weighted data loss plus $\lambda \|\theta\|_2^2$ penalty across a range $\lambda \in [10^{-5}, 10^{-1}]$. The 60,000 MNIST training images are split into 50,000 for training and 10,000 for validation; the standard 10,000-image test set is used only for final evaluation. Hyper-parameter selection (choosing λ) is based on validation accuracy, not the test set.

Figure 5 shows the validation loss and accuracy paths produced by the generator with unit weights ($\omega = \mathbf{1}$); Table 4 reports representative performance metrics for one run. Because the generator accepts bootstrap weights ω as input, it also produces WBB uncertainty summaries: resampling ω from a Dirichlet distribution at the selected λ yields a distribution of fitted models whose spread provides an uncertainty estimate. The baseline model is trained with explicit L_2 regularization matching the generator's penalty.

Over 20 independent replications (Figure 6), the generator achieves test accuracy $95.3\% \pm 0.5\%$ (mean $\pm$ sd) versus $96.7\% \pm 0.2\%$ for the baseline trained from scratch. The accuracy gap is consistent across seeds. The WBB posterior mean accuracy is $95.2\% \pm 0.5\%$ with average posterior standard deviation 0.12%. Evaluating the full 20-point λ tuning curve takes approximately 28s per run on CPU, compared to 35s to train a single baseline model.

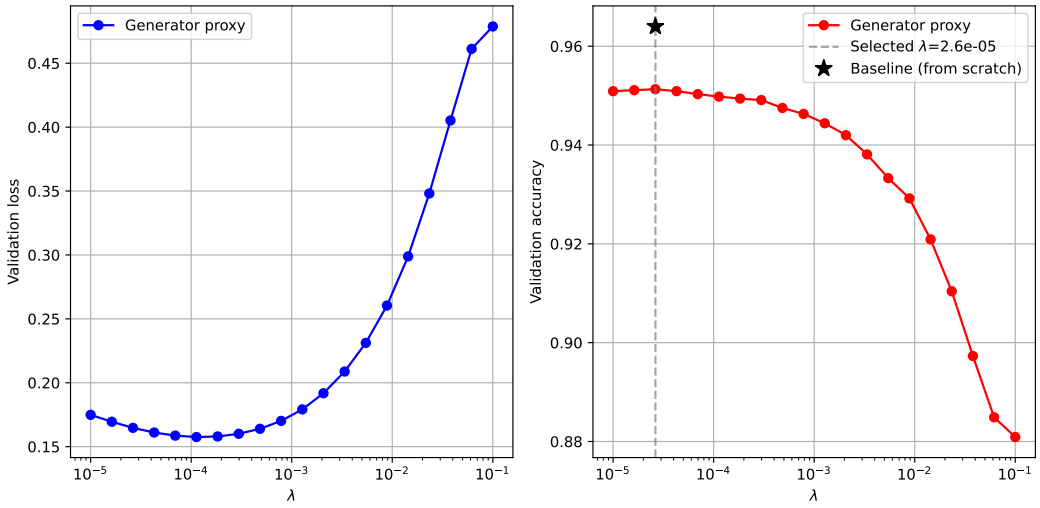

Figure 5: MNIST tuning illustration. Left: validation loss; right: validation accuracy as functions of λ, both produced by a single forward pass through the generator (with unit weights $\omega = \mathbf{1}$) for each λ value. The star marks the baseline model trained from scratch at the generator-selected λ.

Table 4: MNIST results for one illustrative run. Top rows: validation-set performance of the generator across λ values. Bottom rows: test-set accuracy at the selected λ for the generator (point estimate with $\omega = \mathbf{1}$), baseline (trained from scratch with matching L_2 penalty), and WBB posterior mean ($\pm$ standard deviation over 50 bootstrap weight draws). Cross-replicate statistics over 20 independent runs are reported in the text.

λ	Val. Loss	Val. Acc.
1.0e-05	0.1749	0.9509
1.1e-04	0.1575	0.9498
1.3e-03	0.1792	0.9444
1.4e-02	0.2989	0.9209
1.0e-01	0.4788	0.8809
Selected ($\lambda = 2.6e-05$), test	-	0.9565
Baseline ($\lambda = 2.6e-05$), test	-	0.9670
WBB posterior mean, test	-	0.9562 ± 0.0008

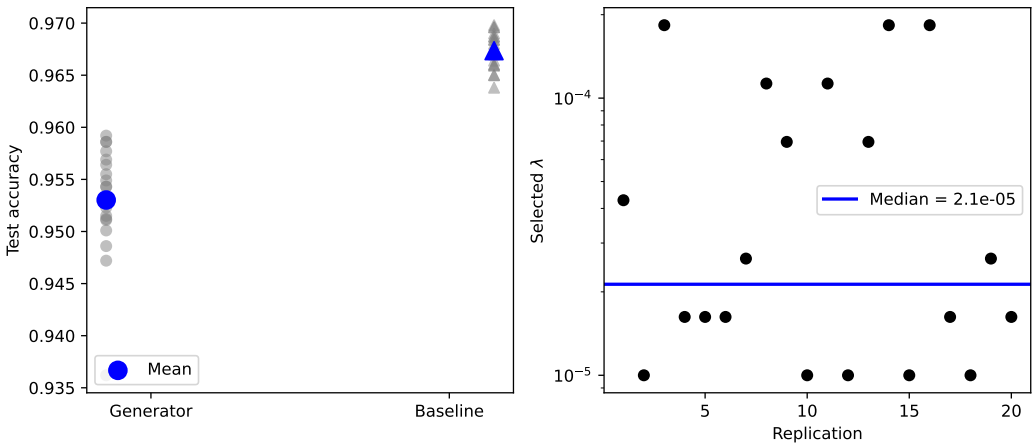

Figure 6: MNIST tuning, 20 replications. Left: test accuracy for the generator (circles) and baseline (triangles) across independent runs; blue markers show means. The accuracy gap (≈ 1.4 percentage points) is stable across seeds. Right: selected λ varies across runs (log scale) with median 2.1×10^{-5}.

4 Discussion

This paper develops a unified view of hyper-parameter tuning as an outer criterion evaluated under an induced (possibly randomized) inner optimization map, and highlights amortization via generators/transport maps as a way to make repeated tuning and uncertainty summaries computationally feasible. By connecting GCV, the weighted Bayesian bootstrap (WBB), and GMS-style criterion-based training, we provide a coherent framework for scalable hyper-parameter learning.

The central benefit of the proposed approach is the amortization of the optimization cost. Once trained, the generator produces fitted parameters for any new hyper-parameter value via a single forward pass, replacing the expensive retraining loop. This speedup is particularly valuable for exploratory analysis and naturally extends to high-dimensional tuning, where the generator can accept vectors of hyper-parameters (e.g., layer-specific penalties, dropout rates) to explore spaces where grid search is infeasible. Because the generator also accepts WBB bootstrap weights as input, it simultaneously provides posterior uncertainty summaries by resampling these weights at test time, as demonstrated in both the Student-t and MNIST experiments.

Our ridge regression experiment illustrated a key trade-off: a simple generator (linear or quadratic map) approximates the tuning curve well but may sacrifice some accuracy relative to exact refitting. The Student-t experiment shows that even a linear generator accurately tracks the profile likelihood for a non-Gaussian model, and that WBB posterior draws provide reasonable uncertainty summaries centered near the true parameters. In deep learning, the generator must have sufficient capacity relative to the complexity of the hyper-parameter-to-parameter map; the MNIST experiment shows that a modest neural generator recovers most of the baseline accuracy while enabling rapid tuning-curve exploration.

Compared to gradient-based hyperparameter optimization [Maclaurin et al., 2015] and bilevel approaches [Franceschi et al., 2018], which also avoid retraining loops by differentiating through the training procedure, the generator approach has a distinct advantage: it produces an explicit map from hyper-parameters to parameters that can be queried at arbitrary h values without backpropagating through training. The trade-off is that the generator must be trained, and its accuracy depends on having sufficient capacity.

Several caveats deserve mention. The generator itself must be trained, which is a one-time upfront cost; the approach is most attractive when many tuning evaluations or repeated uncertainty summaries are needed. The WBB posterior-approximation property on which the uncertainty summaries rest is a conjecture [Newton et al., 2021], supported by asymptotic arguments and empirical evidence but not yet a theorem; our Student-t experiments show that the WBB posterior is slightly under-dispersed relative to frequentist variability, consistent with its first-order justification. In the MNIST experiment, the generator achieves approximately 95% test accuracy versus 97% for the baseline trained from scratch — a gap that reflects the finite capacity of the generator rather than a limitation of the tuning framework itself.

The number of WBB blocks K (16 in our experiments) is a tunable setting whose effect on posterior quality we have not studied; investigating this sensitivity is left for future work. Several other avenues for future work remain. First, adaptive sampling strategies for (λ, ω) during generator training could focus computational effort on the most relevant regions of hyper-parameter space. Second, architectural choices for the generator (e.g., rank-1 modulations, LoRA-style adapters) could further reduce the overhead of training the transport map. Finally, extending this framework to non-differentiable hyper-parameters (e.g., architecture search) remains an open direction.

References

Evan Baker, Pierre Barbillon, Arindam Fadikar, Robert B Gramacy, Radu Herbei, David Higdon, Jiangeng Huang, Leah R Johnson, Pulong Ma, Anirban Mondal, et al. Analyzing stochastic computer models: A review with opportunities. *Statistical Science*, 37(1): 64–89, 2022.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, August 2019.

James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, New York, 2nd edition, 1985.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Anindya Bhadra, Jyotishka Datta, Nicholas G. Polson, Vadim Sokolov, and Jianeng Xu. Merging two cultures: Deep and statistical learning. *WIREs Computational Statistics*, 16 (2):e1647, 2024.

Pier Giovanni Bissiri, Chris Holmes, and Stephen G. Walker. A general framework for updating belief distributions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):1103–1130, 2016.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Jyotishka Datta, Nick Polson, and Vadim Sokolov. Bayesian global-local regularization. *arXiv preprint arXiv:2512.13992*, 2025.

Thaís C. O. Fonseca, Marco A. R. Ferreira, and Helio S. Migon. Objective Bayesian analysis for the Student-*t* regression model. *Biometrika*, 95(2):325–333, 2008.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Nial Friel and Anthony N. Pettitt. Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(3):589–607, 2008.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

Dani Gamerman and Hedibert F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman and Hall/CRC, Boca Raton, 2nd edition, 2006.

Gene H. Golub, Michael Heath, and Grace Wahba. Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter. *Technometrics*, 21(2):215–223, May 1979.

David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2017.

Chuanhai Liu and Donald B. Rubin. The ECME algorithm: A simple extension of EM and ECM with faster monotone convergence. *Biometrika*, 81(4):633–648, 1994.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2113–2122, 2015.

Maria Nareklishvili, Nicholas Polson, and Vadim Sokolov. Deep partial least squares for instrumental variable regression. *Applied Stochastic Models in Business and Industry*, 39 (6):734–754, November 2023a.

Maria Nareklishvili, Nicholas Polson, and Vadim Sokolov. Generative Causal Inference. *arXiv preprint arXiv:2306.16096*, June 2023b.

Maria Nareklishvili, Nick Polson, and Vadim Sokolov. Generative modeling: A review. *arXiv preprint arXiv:2501.05458*, 2024.

Michael A. Newton, Nicholas G. Polson, and Jianeng Xu. Weighted Bayesian bootstrap for scalable posterior distributions. *Canadian Journal of Statistics*, 49(2):421–437, 2021.

Nicholas G. Polson and James G. Scott. Mixtures, Envelopes and Hierarchical Duality. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 78(4):701–727, September 2016.

Nicholas G. Polson and Steven L. Scott. Data augmentation for support vector machines. *Bayesian Analysis*, 6(1):1–23, March 2011.

Nick Polson and Vadim Sokolov. Bayesian double descent. *arXiv preprint arXiv:2507.07338*, 2025a.

Nick Polson and Vadim Sokolov. Generative ai for bayesian computation. *Entropy. An International and Interdisciplinary Journal of Entropy and Information Studies*, 27(7):683, 2025b.

Nick Polson, Fabrizio Ruggeri, and Vadim Sokolov. Generative Bayesian Computation for Maximum Expected Utility. *Entropy. An International and Interdisciplinary Journal of Entropy and Information Studies*, 26(12):1076, December 2024.

Donald B. Rubin. The Bayesian bootstrap. *The Annals of Statistics*, 9(1):130–134, 1981.

Laura Schultz and Vadim Sokolov. Deep Learning Gaussian Processes For Computer Models with Heteroskedastic and High-Dimensional Outputs. *arXiv preprint arXiv:2209.02163*, September 2022.

Minsuk Shin, Shijie Wang, and Jun S. Liu. Generative Multiple-purpose Sampler for Weighted M-estimation, October 2023.

Minsuk Shin, Shijie Wang, and Jun S Liu. Generative Multi-purpose Sampler for Weighted M-estimation. *Journal of Computational and Graphical Statistics*, pages 1–14, 2024.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

M. Stone. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

A Algorithm

Algorithm 1 Generative WBB for hyper-parameter tuning and uncertainty summaries

Require: Training data $\mathcal{D}_{\text{train}}$, validation data $\mathcal{D}_{\text{val}}$; hyper-parameter proposal $\Pi(\lambda, \eta)$; weight distribution $\pi(\omega)$; samples B (training) and M (evaluation); generator family g_φ.

(Choose generator training mode) Either (A) supervised regression to match $\hat{\theta}$, or (B) criterion-based training (as in GMS).

(Training draws) For $b = 1, \ldots, B$:

  Sample $(\lambda^{(b)}, \eta^{(b)}) \sim \Pi(\lambda, \eta)$ and weights $\omega^{(b)} \sim \pi(\omega)$.

if mode (A) **then**

  Compute $\hat{\theta}^{(b)} \approx \arg\min_\theta \left\{ \frac{1}{n} \sum_{i=1}^{n} \omega_i^{(b)} \, \ell_{\eta^{(b)}}(y_i \mid \theta) + \lambda^{(b)} \phi(\theta) \right\}$ using SGD on $\mathcal{D}_{\text{train}}$ (reweighted mini-batches).

  (Train generator) Fit $\hat{\varphi} \in \arg\min_\varphi \sum_{b=1}^{B} \left\| \hat{\theta}^{(b)} - g_\varphi(\omega^{(b)}, \lambda^{(b)}, \eta^{(b)}) \right\|^2$.

else

  (Train generator) Fit $\hat{\varphi} \in \arg\min_\varphi \sum_{b=1}^{B} \left[\frac{1}{n} \sum_{i=1}^{n} \omega_i^{(b)} \, \ell_{\eta^{(b)}}(y_i \mid g_\varphi(\omega^{(b)}, \lambda^{(b)}, \eta^{(b)})) + \lambda^{(b)} \phi\big(g_\varphi(\omega^{(b)}, \lambda^{(b)}, \eta^{(b)})\big) \right]$.

(Tune hyper-parameters) For candidate (λ, η):

  Estimate $J(\lambda, \eta) \approx \frac{1}{M} \sum_{m=1}^{M} \mathcal{R}(g_{\hat{\varphi}}(\omega^{(m)}, \lambda, \eta); \mathcal{D}_{\text{val}})$ with $\omega^{(m)} \sim \pi(\omega)$, and set $(\hat{\lambda}, \hat{\eta}) \in \arg\min_{\lambda,\eta} J(\lambda, \eta)$.

(Uncertainty summaries) With tuned $(\hat{\lambda}, \hat{\eta})$, sample $\omega^{(m)} \sim \pi(\omega)$, set $\theta^{(m)} = g_{\hat{\varphi}}(\omega^{(m)}, \hat{\lambda}, \hat{\eta})$, and summarize predictive uncertainty using the empirical distribution of $\{p(y \mid x, \theta^{(m)})\}_{m=1}^{M}$.