`laplax`
Laplace Approximations with JAX

Tobias Weber [*1] **Bálint Mucsányi** [*1] **Lenard Rommel** [1] **Thomas Christie** [1] **Lars Kasüschke** [1]
Marvin Pförtner [1] **Philipp Hennig** [1]

Abstract

The Laplace approximation provides a scalable and efficient means of quantifying weight-space uncertainty in deep neural networks, enabling the application of Bayesian tools such as predictive uncertainty and model selection via Occam's razor. In this work, we introduce `laplax`[1], a new open-source Python package for performing Laplace approximations with `jax`. Designed with a modular and purely functional architecture and minimal external dependencies, `laplax` offers a flexible and researcher-friendly framework for rapid prototyping and experimentation. Its goal is to facilitate research on Bayesian neural networks, uncertainty quantification for deep learning, and the development of improved Laplace approximation techniques.

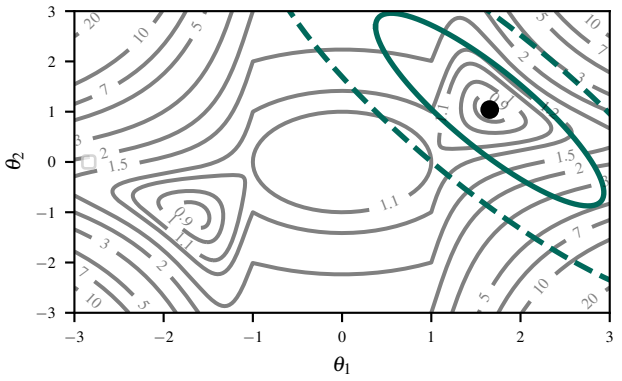

Figure 1: Linearised Laplace approximation on a two-parameter ReLU network $f(x, \theta) = \theta_2 \,\mathrm{ReLU}(\theta_1 x + 1)$ trained on $\mathcal{D} = \{(1, -1), (-1, -1)\}$. Gray contours: energy with square loss; black dot: optimised weights θ^*; green ellipses: 1σ and 2σ levels of the Laplace approximation.

1 Introduction

Bayesian modelling provides principled approaches to several open challenges in modern deep learning (Papamarkou et al., 2024), including overconfidence in predictions (Kristiadi et al., 2020), catastrophic forgetting in continual learning (Ritter et al., 2018), and the incorporation of prior knowledge into model predictions (Cinquin et al., 2024). The Laplace approximation (MacKay, 1992) offers a computationally efficient, post-hoc method for approximating the posterior distribution over neural network weights, effectively transforming standard deep architectures into Bayesian neural networks. This enables the use of Bayesian tools such as predictive uncertainty estimation, marginal likelihood evaluation, and model selection.

Despite its conceptual simplicity, implementing the Laplace approximation involves several non-trivial choices, ranging from curvature estimation and posterior parameterization to calibration and inference techniques. While a comprehensive implementation exists for PyTorch (Daxberger et al., 2021a), a similarly extensive but more flexible and research-oriented solution for the `jax` ecosystem has been lacking.

To address this gap, we introduce `laplax`, a lightweight and modular Python library for Laplace approximations built entirely on `jax` (Bradbury et al., 2018). Designed with research flexibility in mind, `laplax` supports seamless integration with any `jax`-based deep learning framework. It features both a high-level, functional API for rapid experimentation (see Code Snippet 1 producing Figure 1) and low-level building blocks to support in-depth analyses and changing the algorithm itself.

In this paper, we outline the design principles of `laplax`, describe its core components, and demonstrate its application on a simple regression and classification task.

[*]Equal contribution [1]Tübingen AI center, University of Tübingen, Tübingen, Germany. Correspondence to: Tobias Weber <t.weber@uni-tuebingen.de>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

[1]https://github.com/laplax-org/laplax

```python
from jax.nn import relu
from jax.numpy import array
from laplax import laplace
from plotting import plot_figure_1

# You need a model...
def model_fn(input, params):
    return relu(
        params["theta1"] * input - 1
    ) * params["theta2"]
params = { # optimized weights,
    "theta1": array(1.6556547),
    "theta2": array(1.0420421)
}
data = {  # and training data.
    "input": array([1., -1.]),
    "target": array([1., -1.])
}
# ... then apply laplax ...
posterior_fn, _ = laplace(
    model_fn, params, data,
    loss_fn="mse", curv_type="full",
)
arg = {"prior_prec": 0.2}
curv = posterior_fn(arg).state['scale']
# ... to get Figure 1.
plot_figure_1(model_fn, params, curv)
```

Code Snippet 1.1: `laplax` code for generating Figure 1.

2 Making Neural Networks Bayesian

Given labelled training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, loss function ℓ and regularizer Ω, the parameters θ of a neural network f_θ are typically obtained by minimising the regularised empirical risk $\mathcal{L}(\mathcal{D}, f_\theta)$. From a probabilistic perspective, this procedure corresponds to finding a maximum a posteriori (MAP) estimate of the weights under a likelihood and prior. Formally, both views lead to the following optimisation problem:

$$\begin{aligned}\theta^* &= \arg\min_\theta \mathcal{L}(\mathcal{D}, f_\theta) \\ &= \arg\min_\theta \underbrace{\sum_{n=1}^N \ell(f(x_n, \theta), y_n) + \Omega(\theta)}_{\mathcal{L}(\mathcal{D}, f_\theta)} \\ &= \arg\max_\theta \sum_{n=1}^N \log p(y_n \mid f(x_n, \theta)) + \log p(\theta)\,.\end{aligned}$$

The weight-space uncertainty is then described by the posterior distribution given the training data:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{\int p(\mathcal{D} \mid \theta)\, p(\theta)\, d\theta}\,.$$

However, for deep neural networks, the integral in the denominator is usually intractable. The Laplace approximation circumvents this by utilising a Gaussian distribution to approximate the posterior. To this end, we apply a second-order Taylor approximation to the negative log-posterior loss $\mathcal{L}$ around the MAP estimate θ^*, which yields

$$\begin{aligned}\mathcal{L}(\mathcal{D}, f_\theta) \approx &\mathcal{L}(\mathcal{D}, f_{\theta^*}) + \nabla_\theta \mathcal{L}(\mathcal{D}, f_{\theta^*})^\top (\theta - \theta^*) \\ &+ \frac{1}{2}(\theta - \theta^*)^\top \nabla^2_{\theta\theta} \mathcal{L}(\mathcal{D}, f_{\theta^*})(\theta - \theta^*),\end{aligned}$$

where the first-order term vanishes due to the assumed local optimality of θ^*. Negation and exponentiation yield

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}\left(\theta^*, \boldsymbol{H}_{\theta^*}^{-1}\right) \tag{1}$$

with $\boldsymbol{H}_{\theta^*} = \nabla^2_{\theta\theta} \mathcal{L}(\mathcal{D}, f_{\theta^*})$ being the posterior precision. To obtain predictive uncertainty estimates, the weight space uncertainty is pushed forward into the neural network's output space. This is either done via sampling a set of S weights from the approximate posterior and using these in the neural network forward pass to obtain S predictions, or by *linearising* the network around the MAP estimate as

$$f^{\text{lin}}(\cdot, \theta) = f(\cdot, \theta^*) + \mathcal{J}_{\theta^*}(\cdot)(\theta - \theta^*)$$

and using the closure properties of Gaussian distributions under affine maps (Immer et al., 2021b), yielding *closed-form* output-space uncertainty.[2] The linearised approach is guaranteed to yield positive-definite weight-space covariance matrices for a strictly convex regulariser Ω at any weight configuration θ, not just at MAP estimates (that are hard to obtain exactly in deep learning settings). Usually, further approximations are needed to reduce the computational and memory requirements of the curvature. These are discussed in Section 3.

An important Bayesian tool for model selection is the log marginal likelihood given by

$$\log p(\mathcal{D} \mid \mathcal{M}) \approx \log p(\mathcal{D}, \theta^* \mid \mathcal{M}) - \frac{1}{2} \log \left| \frac{1}{2\pi} \boldsymbol{H}_{\theta^*} \right|. \tag{2}$$

This term is often used for the selection of the model hyperparameters $\mathcal{M}$ via maximization (Immer et al., 2021a), since it represents an analytic trade-off between complexity and expressivity – the so-called Occam's razor (Rasmussen & Ghahramani, 2000). Tractability and scalability depend on the structure of the estimated $\boldsymbol{H}_{\theta^*}$, but compared to the predictive uncertainty above (cf. Equation (1)), no inversion is needed.

3 `laplax`—A Modular Toolbox

The high-level API of `laplax` provides the main function `laplace(...)` for fitting weight space curvature

[2] For classification, the logit-space uncertainty is analytic, but the predictive distribution has to be approximated, e.g., through Monte Carlo sampling and averaging the softmax probabilities.

approximations (see Code Snippet 1 for usage), as well as additional functions for calibrating hyperparameters (`calibration(...)`) and a framework for evaluations (`evaluation(...)`). The latter two operate on the weight-space posterior function `posterior_fn`, which maps hyperparameters, e.g., the prior precision, to the posterior covariance. While these high-level functions enable quick experimentation, the core strength of `laplax` lies in its modular design. Each step of the Laplace approximation pipeline is exposed as an independent, composable function, closely reflecting the general algorithmic scaffold. These components are designed to be easily replaceable, encouraging experimentation and replacement with alternative functionality from complementary packages (e.g., (Pinder & Dodd, 2022)). In the following, we provide an overview of the currently available features. Complementary details are listed in the Appendix A.

Curvature-vector products. A central element of the Laplace approximation is the choice of curvature. Following the motivation in (Dangel et al., 2025), `laplax` represents and handles all curvatures as *matrix-vector products*. This matrix-free approach significantly reduces memory usage, enhances computational efficiency, and improves flexibility. Currently, `laplax` supports both Hessian- and Generalized Gauss-Newton (GGN)-vector products, which can be computed on arbitrary iterables of data points, including PyTorch DataLoaders (Ansel et al., 2024) and TensorFlow Datasets (TFD).

`CurvApprox`: from curvature to posterior precision. The primary trade-off between speed and accuracy lies in how the curvature is approximated and the structure it assumes. Once the curvature approximation method (`CurvApprox`) is selected, it is processed into a function (`posterior_fn`) that returns the posterior precision ($\boldsymbol{H}_{\theta^*}$) *matrix-vector product* given hyperparameters. These include the prior precision – modelled as an identity matrix scaled by the `prior_prec` (τ), representing an isotropic Gaussian prior – and additional hyperparameters $\mathcal{C}$ of the negative log-likelihood loss, such as the observation noise σ^2 for regression (Daxberger et al., 2021a). `laplax` currently supports the following curvature approximations:

- `CurvApprox.FULL` materializes the full matrix in memory by applying the curvature-vector product to the columns of an identity matrix. The posterior function is then given by

$$(\tau, \mathcal{C}) \mapsto \left[v \mapsto \left(\textbf{Curv}(\mathcal{C}) + \tau \boldsymbol{I} \right)^{-1} v \right]. \quad (3)$$

- `CurvApprox.DIAGONAL` approximates the curvature using only its diagonal, obtained by evaluating the curvature-vector product with standard basis vectors from both sides. This leads to:

$$(\tau, \mathcal{C}) \mapsto \left[v \mapsto \left(\text{diag}(\textbf{Curv}(\mathcal{C})) + \tau \boldsymbol{I} \right)^{-1} v \right] .3 \quad (4)$$

- **Low-Rank** employs either a custom Lanczos routine (`CurvApprox.LANCZOS`) or a variant of the LOBPCG algorithm (`CurvApprox.LOBPCG`). These methods approximate the top eigenvectors $\boldsymbol{U}$ and eigenvalues $\boldsymbol{S}$ of the curvature via matrix-vector products. The posterior is then given by a low-rank plus scaled diagonal

$$(\tau, \mathcal{C}) \mapsto \left[v \mapsto \left(\left[\boldsymbol{U} \boldsymbol{S} \boldsymbol{U}^\top \right] (\mathcal{C}) + \tau \boldsymbol{I} \right)^{-1} v \right]. \quad (5)$$

In addition to the `posterior_fn`, the main `laplace(...)` function also returns the curvature estimate that is relevant, e.g., for computing the log marginal likelihood. More details on the transformations are provided in Appendix A.1.

Standard approximation variants such as *last-layer* or *subnetwork Laplace* – where only a subset of model parameters is treated probabilistically (Daxberger et al., 2021b) – are package-independent, since all computations in `laplax` depend on a generic model signature:

$$\texttt{model\_fn} : (\text{input}, \ \text{params}) \mapsto \text{output},$$

and take arbitrary PyTrees as parameters θ. This also allows for flexible integration with arbitrary `jax`-based models.

`Pushforward`: from weight space to output space. `laplax` supports two strategies for propagating uncertainty from weight space to output space: sampling-based (`Pushforward.NONLINEAR`) and linearisation-based (`Pushforward.LINEAR`) pushforwards. In classification settings, this results in uncertainty over the *logits*, which must be further transformed to obtain predictive uncertainty over class probabilities, either by sampling in logit space and applying the `softmax` independently, or via analytic approximations to the integral

$$\int \text{softmax}(f_\theta(x_n)) \, p(f_\theta(x_n) \mid \mathcal{D}) \, d\theta \, . \quad (6)$$

A full list of supported predictive approximations is provided in Appendix A.3.

`calibration(...)`. `laplax` supports the calibration of all posterior hyperparameters, such as τ (prior precision) or σ^2 (observation noise). There are two primary strategies for calibration: (1) maximising the log marginal likelihood (see Equation (2)) with respect to the hyperparameters; or (2) optimising for a downstream metric such as the negative log-likelihood (regression) or the expected calibration error (classification). The library includes a basic grid search

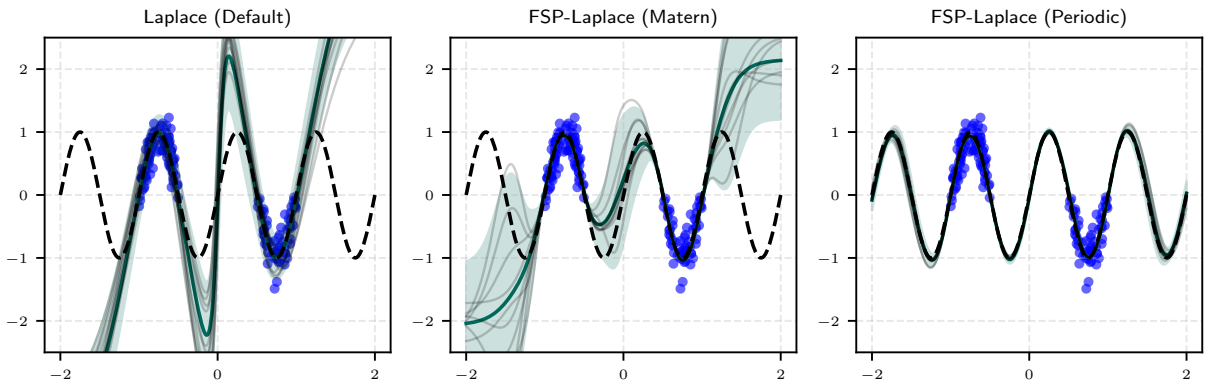

Figure 2: Comparison of linearised Laplace with Lanczos-approximated curvature (NLL-GD-L) (left) and FSP-Laplace using Matérn-5/2 and periodic prior kernels. The shaded region denotes predictive uncertainty, the blue points represent training data.

Table 1: NLL results for Laplace approximation under different curvature and calibration settings (Log Marginal Likelihood objective (LML) or Negative Log-Likelihood (NLL); Grid Search (GS) or Gradient Descent (GD) – latter also calibrating the observation noise σ^2 – with either (Linear (L) or Nonlinear (NL) pushforward.

	LML-GS-L	NLL-GS-L	LML-GD-L	NLL-GD-L	LML-GS-NL	NLL-GS-NL	LML-GD-NL	NLL-GD-NL
`.FULL` (last-layer)	1.5283	**1.2843**	**0.4799**	**0.4482**	1.8311	**1.4998**	**0.5480**	**0.5400**
`.FULL`	0.8457	1.4409	0.8988	0.5104	4.2959	2.1243	4.3586	3.4363
`.DIAGONAL`	**0.7687**	2.1212	2.2358	2.5784	**1.0230**	1.6201	2.5373	2.4986
`.LANCZOS`	0.9437	1.3771	0.5109	0.5008	2.4008	2.4717	2.3857	2.8676

routine over τ, but all functions are differentiable, allowing the use of gradient-based optimization, e.g., via `optax` (DeepMind et al., 2020), for calibrating all hyperparameters at the same time. While calibration with respect to a downstream metric often yields better predictive performance, marginal likelihood maximization is computationally more efficient, as it avoids matrix inversions and pushforward computations.

`evaluation(...)`. To assess the predictive quality of a Laplace approximation, `laplax` provides a unified evaluation interface. This combines the pushforward step with the computation of summary statistics (e.g., mean, standard deviation, covariance) of standard uncertainty quantification metrics such as negative log-likelihood, continuously ranked probability score, and others. All components are modular and can be easily extended to support custom evaluation pipelines.

4 Experiments

To illustrate the practical use of `laplax`, we combine its core components in a simple regression task, focusing on functionality and modularity rather than comprehensive evaluation of the quantified uncertainty. For in-depth empirical comparisons and benchmarks, refer to the Related Works section in Appendix B. Here, we report only the negative log-likelihood (NLL) as a measure of calibrated uncertainty. FSP-Laplace (Cinquin et al., 2024), as a demonstration of recent advancement, is included in our visual comparison (and `laplax`). All experimental code is available in the package repository; additional classification results are reported in Appendix D.

Regression. We train a three-layer MLP with `tanh` activations and 50 hidden units on noisy samples of $y = \sin(2\pi x)$. Various curvature and calibration settings are compared by NLL in Table 1. Figure 2 visualizes predictive uncertainty for the full Laplace and FSP-Laplace (with periodic kernel and Matérn-5/2).

5 Perspective and conclusion

By having a modular, functional, and loosely-typed implementation without any hard dependency besides `jax`, we aim at a flexible framework where parts can be easily reused – to study or implement other UQ pipelines – or swapped with building blocks from different libraries, supporting the common research workflow. Furthermore, the framework inherits standard features from `jax`, such as `jit`-compilation, hardware-agnosticity, parallelization (`vmap`), and autodiff (`grad`, `jvp`, `vjp`).

Limitations and outlook. Some important building blocks are yet to be implemented in `laplax`. These include a general Kronecker-Factored approximation for curvature-vector products (KFAC) and the family of Fisher curvature-vector products. While these curvature proxies are easy for the user to register and use (e.g., with the KFAC implementation of (Botev & Martens, 2022)), we are currently working on providing them directly in future iterations of the library.

Acknowledgements

The authors gratefully acknowledge financial support by the European Research Council through ERC CoG Action 101123955 ANUBIS; the DFG Cluster of Excellence "Machine Learning - New Perspectives for Science", EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); the DFG SPP 2298 (Project HE 7114/5-1), and the Carl Zeiss Foundation (project "Certification and Foundations of Safe Machine Learning Systems in Healthcare"), as well as funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg.

Impact Statement

This paper describes a software library designed to facilitate research in approximate Bayesian uncertainty quantification for deep neural networks. There are many potential societal consequences of our work, as predictive uncertainty quantification contributes to mitigating the risk of real-world AI applications.

References

TensorFlow Datasets, A Collection of ready-to-use datasets. https://www.tensorflow.org/datasets.

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Suo, M., Tillet, P., Wang, E., Wang, X., Wen, W., Zhang, S., Zhao, X., Zhou, K., Zou, R., Mathews, A., Chanan, G., Wu, P., and Chintala, S. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL https://docs.pytorch.org/assets/pytorch2-2.pdf.

Botev, A. and Martens, J. KFAC-JAX, 2022. URL https://github.com/google-deepmind/kfac-jax.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: Composable Transformations of Python+NumPy Programs, 2018. URL http://github.com/jax-ml/jax.

Cinquin, T., Pförtner, M., Fortuin, V., Hennig, P., and Bamler, R. FSP-Laplace: Function-Space Priors for the Laplace Approximation in Bayesian Deep Learning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. URL https://arxiv.org/abs/2407.13711.

Dangel, F., Eschenhagen, R., Ormaniec, W., Fernandez, A., Tatzel, L., and Kristiadi, A. Position: Curvature Matrices Should Be Democratized via Linear Operators, 2025. URL https://arxiv.org/abs/2501.19183.

Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M., and Hennig, P. Laplace Redux – Effortless Bayesian Deep Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021a. URL https://arxiv.org/abs/2106.14806.

Daxberger, E., Nalisnick, E., Allingham, J. U., Antorán, J., and Hernández-Lobato, J. M. Bayesian Deep Learning via Subnetwork Inference. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research (PMLR)*, 2021b.

DeepMind, Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., Budden, D., Cai, T., Clark, A., Danihelka, I., Dedieu, A., Fantacci, C., Godwin, J., Jones, C., Hemsley, R., Hennigan, T., Hessel, M., Hou, S., Kapturowski, S., Keck, T., Kemaev, I., King, M., Kunesch, M., Martens, L., Merzic, H., Mikulik, V., Norman, T., Papamakarios, G., Quan, J., Ring, R., Ruiz, F., Sanchez, A., Sartran, L., Schneider, R., Sezener, E., Spencer, S., Srinivasan, S., Stanojević, M., Stokowiec, W., Wang, L., Zhou, G., and Viola, F. The DeepMind JAX Ecosystem, 2020. URL http://github.com/google-deepmind.

Dhahri, R., Immer, A., Charpentier, B., Günnemann, S., and Fortuin, V. Shaving Weights with Occam's Razor: Bayesian Sparsification for Neural Networks Using the Marginal Likelihood. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2402.15978.

Eschenhagen, R., Daxberger, E., Hennig, P., and Kristiadi, A. Mixtures of Laplace Approximations for Improved Post-Hoc Uncertainty in Deep Learning. In *Bayesian Deep Learning Workshop, NeurIPS*, 2021. URL https://arxiv.org/abs/2111.03577.

Fortuin, V. Priors in Bayesian Deep Learning: A Review. *International Statistical Review*, 90(3):563–591, 2022. URL https://arxiv.org/abs/2105.06868.

Hobbhahn, M., Kristiadi, A., and Hennig, P. Fast Predictive Uncertainty for Classification with Bayesian Deep Networks. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 180 of *Proceedings of Machine Learning Research*, 2022. URL https://arxiv.org/abs/2003.01227.

Immer, A., Bauer, M., Fortuin, V., Rätsch, G., and Khan, M. E. Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2021a. URL https://arxiv.org/abs/2104.04975.

Immer, A., Korzepa, M., and Bauer, M. Improving Predictions of Bayesian Neural Nets via Local Linearization. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 130 of *Proceedings of Machine Learning Research (PMLR)*, 2021b. URL http://arxiv.org/abs/2008.08400.

Kristiadi, A., Hein, M., and Hennig, P. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/2002.10118.

Kristiadi, A., Hein, M., and Hennig, P. Learnable Uncertainty under Laplace Approximations. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 161 of *Proceedings of Machine Learning Research*, 2021. URL https://arxiv.org/abs/2010.02720.

Kristiadi, A., Immer, A., Eschenhagen, R., and Fortuin, V. Promises and Pitfalls of the Linearized Laplace in Bayesian Optimization. In *Fifth Symposium on Advances in Approximate Bayesian Inference*, 2023. URL https://arxiv.org/abs/2304.08309.

Lu, Z., Ie, E., and Sha, F. Mean-Field Approximation to Gaussian-Softmax Integral with Application to Uncertainty Estimation, 2021. URL https://arxiv.org/abs/2006.07584.

MacKay, D. J. A Practical Bayesian Framework for Backpropagation Networks. *Neural computation*, 4(3):448–472, 1992.

Magnani, E., Pförtner, M., Weber, T., and Hennig, P. Linearization Turns Neural Operators into Function-Valued Gaussian Processes. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research (PMLR)*, 2025. URL https://arxiv.org/abs/2406.05072.

Mucsányi, B., Kirchhof, M., and Oh, S. J. Benchmarking Uncertainty Disentanglement: Specialized Uncertainties for Specialized Tasks. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Papamarkou, T., Skoularidou, M., Palla, K., Aitchison, L., Arbel, J., Dunson, D., Filippone, M., Fortuin, V., Hennig, P., Hernández-Lobato, J. M., Hubin, A., Immer, A., Karaletsos, T., Khan, M. E., Kristiadi, A., Li, Y., Mandt, S., Nemeth, C., Osborne, M. A., Rudner, T. G. J., Rügamer, D., Teh, Y. W., Welling, M., Wilson, A. G., and Zhang, R. Position: Bayesian Deep Learning is Needed in the Age of Large-Scale AI. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research (PMLR)*, 2024. URL https://arxiv.org/abs/2402.00809.

Pinder, T. and Dodd, D. GPJax: A Gaussian process framework in JAX. *Journal of Open Source Software*, 7(75):4455, 2022. doi: 10.21105/joss.04455. URL https://doi.org/10.21105/joss.04455.

Rasmussen, C. and Ghahramani, Z. Occam's Razor. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 13, 2000.

Ritter, H., Botev, A., and Barber, D. Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL http://arxiv.org/abs/1805.07810.

Roy, H., Miani, M., Ek, C. H., Hennig, P., Pförtner, M., Tatzel, L., and Hauberg, S. Reparameterization Invariance in Approximate Bayesian Inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. URL https://arxiv.org/abs/2406.03334.

Sliwa, J., Schneider, F., Bosch, N., Kristiadi, A., and Hennig, P. Efficient Weight-Space Laplace-Gaussian Filtering and Smoothing for Sequential Deep Learning, 2024. URL http://arxiv.org/abs/2410.06800.

Sun, S., Zhang, G., Shi, J., and Grosse, R. Functional Variational Bayesian Neural Networks, 2019. URL https://arxiv.org/abs/1903.05779.

Tran, B.-H., Rossi, S., Milios, D., and Filippone, M. All You Need is a Good Functional Prior for Bayesian Deep Learning. *Journal of Machine Learning Research*, 23 (74):1–56, 2022.

van der Ouderaa, T. F., Immer, A., and van der Wilk, M. Learning Layer-Wise Equivariances Automatically Using Gradients. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL http://arxiv.org/abs/2310.06131.

van der Ouderaa, T. F. A., Nagel, M., van Baalen, M., Asano, Y. M., and Blankevoort, T. The llm surgeon. In *International Conference of Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2312.17244.

A Computational details

In the following section, we provide additional details of how different computations are performed and list more available building blocks of `laplax`.

A.1 Curvatures

The package supports both Hessian- and GGN-vector products:

$$v \mapsto \sum_{n=1}^{N} \nabla^2_{\theta\theta} \ell(f_\theta(x_n), y_n)\, v \quad \text{(Hessian-vector product)},$$

$$v \mapsto \sum_{n=1}^{N} \mathcal{J}_\theta^\top(f_{\theta^*}(x_n)) \nabla^2_{f^*_\theta(x_n), f^*_\theta(x_n)} \ell(f_{\theta^*}(x_n), y_n) \mathcal{J}_\theta(f_{\theta^*})\, v \quad \text{(GGN-vector product)}$$

We then provide the following approximation pipelines. Each of them also comes with their individual log marginal likelihood implementation.

A.2 `CurvApprox.FULL`

Estimation. The total curvature $\mathbf{Curv}(\mathcal{C})$ is computed by multiplying the curvature-vector product by the identity matrix.

Posterior precision. The posterior precision is then computed in a dense form with

$$\boldsymbol{H}_{\theta^*} = \mathbf{Curv}(\mathcal{C}) + \tau \boldsymbol{I}_P$$

where we $\tau \boldsymbol{I}_P$ represent the prior with $\theta \in \mathbb{R}^P$.

Posterior scale. We compute a left square root of the posterior covariance matrix by means of a Cholesky decomposition of the posterior precision matrix and then solve a triangular system to recover the scale matrix. This matches the implementation in `torch.distributions.multivariate.normal` (Ansel et al., 2024).

Log marginal likelihood. The log marginal likelihood combines the joint log-likelihood at the MAP estimate with a precision-based evidence correction:

$$\log p(\mathcal{D} \mid \mathcal{M}) = \underbrace{-\frac{1}{2\sigma^2} \sum_{n=1}^{N} \ell(y_n | f_\theta(x_n), \mathcal{M}) - \tfrac{\tau}{2}\|\theta\|^2}_{=: p(\mathcal{D}, \theta | \mathcal{D})} - \tfrac{1}{2}\Big(P \log(2\pi) \; - \; \log\big|\boldsymbol{H}_{\theta^*}\big|\Big).$$

Here, the joint likelihood is given by the training objective, including the prior / regularization term, including σ^2 and τ, both of which would also appear in the posterior precision $\boldsymbol{H}_{\theta^*}$. In regression settings, σ^2 plays the role of observation noise. It can also be viewed as a curvature hyperparameter $\mathcal{C}$.

A.2.1 `CurvApprox.DIAGONAL`

Estimation. The diagonal curvature approximation extracts only the diagonal of the full curvature matrix $\mathbf{Curv}(\mathcal{C})$, using basis-vector multiplications with the given curvature-vector product.

Posterior precision. The diagonal entries of the posterior precision are formed by adding the isotropic prior precision τ to the curvature diagonal $c_i = \mathrm{diag}(\mathbf{Curv}(\mathcal{C}))_i$:

$$d_i = c_i + \tau\,, \quad i = 1, \ldots, P.$$

Posterior scale. The scale factors are simply the element-wise square-root of the inverse precision:

$$L_{ii} \;=\; \sqrt{\frac{1}{d_i}},$$

so that $L\,L^\top$ yields the diagonal covariance.

Log marginal likelihood. In the diagonal case, the evidence correction reduces to $\sum_{i=1}^{P} \log d_i$. Writing the joint objective at the MAP as before, the log marginal likelihood is

$$\log p(\mathcal{D} \mid \mathcal{M}) = \underbrace{-\tfrac{1}{2\sigma^2} \sum_{n=1}^{N} \ell(y_n | f_\theta(x_n), \mathcal{M}) - \tfrac{\tau}{2}\|\theta\|^2}_{=:\log p(\mathcal{D}, \theta^* | \mathcal{M})} - \tfrac{1}{2}\Big(P \log(2\pi) - \sum_{i=1}^{P} \log d_i\Big).$$

A.2.2 Low Rank: `CurvApprox.LANCZOS` and `CurvApprox.LOBPCG`

Estimation. We approximate the leading R eigenpairs of the full curvature via a low-rank method (Lanczos or LOBPCG) applied to the curvature-vector product. Both extract $\boldsymbol{U} \in \mathbb{R}^{P \times R}$ and eigenvalues $\boldsymbol{S} \in \mathbb{R}^{R}$, but Lanczos typically requires significantly fewer matrix-vector products.

Posterior precision. Denoting the prior precision by τ, the low-rank posterior precision is

$$\boldsymbol{H}_{\theta^*} = \boldsymbol{U}\ \mathrm{diag}(\boldsymbol{S})\, \boldsymbol{U}^\top + \tau \boldsymbol{I}\,.$$

Posterior scale. We transform the posterior precision to its inverse square root by means of the procedure outlined by Roy et al. (2024, Section E.1):

$$v \mapsto \tau^{-1/2} v\ +\ \boldsymbol{U}\big(\mathrm{diag}(\overline{\boldsymbol{S}})\, (\boldsymbol{U}^\top v)\big)\,,$$

where we have

$$\overline{\boldsymbol{S}} = (\boldsymbol{S} + \tau)^{-1/2} - \tau^{-1/2}\,.$$

Log marginal likelihood. The evidence correction uses the matrix determinant lemma:

$$\log|H| = P \log \tau\ +\ \sum_{i=1}^{R} \log\big(1 + \tau^{-1} \boldsymbol{S}_i\big),$$

so that

$$\log p(\mathcal{D} \mid \mathcal{M}) = \log p(\mathcal{D}, \theta_* \mid \mathcal{M}) - \tfrac{1}{2}\Big(P \log(2\pi) - \big[P \log \tau + \sum_{i=1}^{R} \log(1 + \tau^{-1} \boldsymbol{S}_i)\big]\Big).$$

A.3 Pushforward and predictives

We distinguish two ways of pushing forward the weight space uncertainty for a new data sample onto the output space;

- `Pushforward.LINEAR` This takes the weight space covariance $\boldsymbol{H}_{\theta^*}^{-1}$ and Jacobian-vector products ($\mathcal{J}_\theta(x_{new})$) to return a covariance in output space:

$$\mathcal{N}\Big(f(x_n, \theta^*), \mathcal{J}_\theta(f(x_n, \theta^*)) \boldsymbol{H}_{\theta^*}^{-1} \mathcal{J}_\theta(f(x_n, \theta^*))^\top\Big).$$

- `Pushforward.NONLINEAR` This samples new weights in weight space and uses the neural network with the new weight samples to get an ensemble in output space, for which then empirical estimates can be computed.

$$f(x_n, \theta_s), \quad \theta_s \sim \mathcal{N}\big(\theta_*, \boldsymbol{H}_{\theta^*}^{-1}\big)\,.$$

For classification, additional approximations are needed to push the uncertainty from logit space onto the class labels. `laplax` supports the following predictives (`Predictive.*`) for approximating the integral in Equation (6).

- `MC_BRIDGE` Draw $z_s \sim \mathcal{N}\big(\mu, \Sigma\big)$, compute $p_s = \mathrm{softmax}(z_s)$ for $s = 1 \dots S$, and form $\frac{1}{S}\sum_s p_s$.

- `LAPLACE_BRIDGE` Transforms the Gaussian over logits into a Dirichlet by moment-matching ("bridge"), yielding closed-form Dirichlet parameters and thus an analytic predictive mean. The Laplace Bridge predictive (Hobbhahn et al., 2022) approximates the true predictive as follows:

$$\hat{\boldsymbol{p}} := \frac{\frac{1}{\tilde{\boldsymbol{\sigma}}^2}\left(1 - \frac{2}{C} + \frac{e^{\tilde{\boldsymbol{\mu}}}}{C^2}\sum_{c=1}^{C} e^{-\tilde{\mu}_c}\right)}{\sum_{c=1}^{C}\frac{1}{\tilde{\sigma}_c^2}\left(1 - \frac{2}{C} + \frac{e^{\tilde{\boldsymbol{\mu}}}}{C^2}\sum_{c'=1}^{C} e^{-\tilde{\mu}_{c'}}\right)} \tag{7}$$

  where

$$\tilde{\boldsymbol{\mu}}^2 := \sqrt{\frac{\sqrt{C/2}}{\sum_{c=1}^{C}\sigma_c^2}}\boldsymbol{\mu},\ \tilde{\boldsymbol{\sigma}}^{\boldsymbol{2}} := \frac{\sqrt{C/2}}{\sum_{c=1}^{C}\sigma_c^2}\boldsymbol{\sigma}^2. \tag{8}$$

- `MEAN_FIELD_0_PREDICTIVE`. A zeroth-order mean-field (probit-style) approximation (Lu et al., 2021).

$$\mathbb{E}[\mathrm{softmax}_i(\boldsymbol{z})] \approx \mathrm{softmax}_i\left(\frac{\boldsymbol{\mu}}{\sqrt{1 + \lambda_0\ \mathrm{diag}(\boldsymbol{\Sigma})}}\right)$$

  which rescales each mean logit by its variance.

- `MEAN_FIELD_1_PREDICTIVE` A first-order pairwise approximation: for each i we approximate $\Pr(z_i > z_j)$ under the bivariate Gaussian of (z_i, z_j) and then normalize:

$$\mathbb{E}[\mathrm{softmax}_i(\boldsymbol{z})] \approx \frac{1}{1 + \sum_{i \neq k}\exp\left(-\frac{(\mu_k - \mu_i)}{\sqrt{1+\lambda_0(\Sigma_{kk}+\Sigma_{ii})}}\right)}. \tag{9}$$

- `MEAN_FIELD_2_PREDICTIVE`. A second-order correction that incorporates full covariance: uses all bivariate variances and covariances in the exponentiated difference integrals:

$$\mathbb{E}[\mathrm{softmax}_i(\boldsymbol{z})] \approx \frac{1}{1 + \sum_{i \neq k}\exp\left(-\frac{(\mu_k - \mu_i)}{\sqrt{1+\lambda_0(\Sigma_{kk}+\Sigma_{ii}-2\Sigma_{ik})}}\right)}. \tag{10}$$

Each method trades off cost versus fidelity: sampling is asymptotically exact but can be slow; the mean-field approximations incur only $O(C^2)$ or $O(C)$ work; and the Laplace bridge often gives the best calibrated probabilities when variances are large.

B Applications and extensions of the Laplace approximation

Section 4 discusses Laplace approximation with the goal of calibrated predictive uncertainty. Here, a variety of follow-up work with Laplace exists (Kristiadi et al., 2021; Eschenhagen et al., 2021; Cinquin et al., 2024) and Magnani et al. (2025) lift the method to the setting of operator learning. Notable work shows that adding some weight space uncertainty fixes overconfidence in classification networks (Kristiadi et al., 2020). In comparison with other uncertainty quantification methods Laplace performs comparable on the uncertainty disentanglement benchmark of (Mucsányi et al., 2025).

In our opinion, a huge potential of linearised Laplace approximation is given by its analytic uncertainty due to the Gaussian structure. A wide field of applications opens up via the marginal log-likelihood, which provides scalable and tractable formulation of Occam's razor (Immer et al., 2021a), which was successfully used for weight pruning (Dhahri et al., 2024; van der Ouderaa et al., 2024) or model selection, e.g. for learning layerwise equivariance (van der Ouderaa et al., 2023).

The analytic uncertainty structure provided by Laplace has also been used to apply filtering techniques to neural network learning with the goal of online/continual learning (Ritter et al., 2018; Sliwa et al., 2024) or to explore Bayesian optimization (Kristiadi et al., 2023).

Various other applications exist and this non-extensive list aimed only at provided some pointers for potential use cases.

C FSP-Laplace

The BNN literature offers only a modicum of methods for eliciting informative priors, with few exceptions (Tran et al., 2022; Sun et al., 2019). The commonly used isotropic Gaussian prior imposes problematic assumptions, including unimodality and weight independence that oppose the reality in neural networks and compromise uncertainty calibration and prediction reliability. As network weights are not interpretable, formulating a good prior on them is virtually impossible (Fortuin, 2022). As a remedy, *FSP-Laplace* (Cinquin et al., 2024) extends the linearised Laplace approximation by placing interpretable Gaussian Process (GP) priors directly in function space, thereby overcoming the non-interpretability of the weight-space prior. This yields a more refined MAP estimate and well-calibrated epistemic uncertainties when prior knowledge is available. This is particularly useful in scientific machine learning, where prior knowledge and ideas about boundary conditions and the domain are often abundant. We therefore offer FSP-Laplace to become part of the plethora of functionalities of `laplax` as a configurable inference method, allowing users to seamlessly specify interpretable Gaussian process priors in function space and apply the Laplace approximation to their neural network models. As can be seen in the example experiment in Figure 2, we reproduce one of the experiments of Cinquin et al. (2024), which shows the incorporation of a periodic GP prior in function space, yielding better predictive results.

FSP-Laplace approximation

FSP-Laplace consists of two major components: first, training the neural network with an RKHS regulariser to obtain the MAP estimate, described in Algorithm 1, followed by the linearised Laplace approximation around this estimate, described in Algorithm 2.

C.1 FSP Training

FSP-Laplace differs both from vanilla Laplace approximation (MacKay, 1992) and standard linearised Laplace approximation (Immer et al., 2021b) in that it requires training the model with a Reproducing Kernel Hilbert Space (RKHS) $\mathbb{H}_\Sigma$ regulariser. This regulariser (eq. (12)) is added to the negative log-likelihood (eq. (11)) to form the FSP objective function (eq. (13)). The resulting MAP estimate incorporates the functional constraints imposed by the RKHS $\mathbb{H}_\Sigma$ regulariser, which is essential for the subsequent linearised approximation step.

$$R_{FSP}^{(1)}(\theta^{(i)}) = -\frac{n}{b}\sum_{j=1}^{b} \log p\big(y^{(j)} \mid f(x^{(j)}, \theta^{(i)})\big) \tag{11}$$

The NLL is multiplied by the factor $\frac{n}{b}$ to account for the difference between the full dataset size n and the mini-batch size b, ensuring that the regularisation strength remains consistent regardless of batch size during stochastic optimisation.

$$R_{FSP}^{(2)}(\theta^{(i)}) = \frac{1}{2}\big(f(\mathcal{C}^{(i)}, \theta^{(i)}) - \boldsymbol{\mu}(\mathcal{C}^{(i)})\big)^\top \boldsymbol{\Sigma}(\mathcal{C}^{(i)}, \mathcal{C}^{(i)})^{-1}\big(f(\mathcal{C}^{(i)}, \theta^{(i)}) - \boldsymbol{\mu}(\mathcal{C}^{(i)})\big) \tag{12}$$

$$R_{FSP}(\theta^{(i)}) = R_{FSP}^{(1)}(\theta^{(i)}) + R_{FSP}^{(2)}(\theta^{(i)}). \tag{13}$$

Algorithm 1 implements the FSP training procedure by computing two loss components at each iteration: the NLL $R_{FSP}^{(1)}$ on the current mini-batch, and the RKHS regularisation term $R_{FSP}^{(2)}$ which approximates the RKHS norm $\|\cdot\|_{\mathbb{H}_\Sigma}$ of the difference between the network's prediction and the prior mean at the context points. The optimiser then updates the parameters using the combined objective $R_{FSP}^{(1)} + R_{FSP}^{(2)}$. The selection and sampling strategy for context points $\mathcal{C}^{(i)}$ is discussed in Section C.3.

C.2 FSP curvature estimation

Once the MAP estimate θ^* is obtained through FSP training, we compute the linearised Laplace approximation around this point:

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta^*, \boldsymbol{\Lambda}^{-1}), \tag{14}$$

Algorithm 1 RKHS-regularised model training (Cinquin et al., 2024, Algorithm 1)

function FSPLAPLACETRAIN(f, $\theta^{(0)}$, $\mathcal{GP}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $P_{\mathcal{C}}$, $\mathcal{D}$, b)
  $i \leftarrow 0$
  for all minibatch $\mathcal{B} = (X_{\mathcal{B}}, Y_{\mathcal{B}}) \sim \mathbb{D}$ of size b **do**
    $R^{(1)}_{\text{FSP}}(\theta^{(i)}) \leftarrow -\frac{n}{b} \sum_{j=1}^{b} \log p(y^{(j)}_{\mathcal{B}} \mid f(x^{(j)}_{\mathcal{B}}, \theta^{(i)}))$
    Sample context points $\mathcal{C}^{(i)} = \{\mathcal{C}^{(i)}_j\}_{j=1}^{nc} \overset{\text{i.i.d.}}{\sim} P_{\mathcal{C}}$
    $R^{(2)}_{\text{FSP}}(\theta^{(i)}) \leftarrow \frac{1}{2}(f(\mathcal{C}^{(i)}, \theta^{(i)}) - \boldsymbol{\mu}(\mathcal{C}^{(i)}))^\top \boldsymbol{\Sigma}(\mathcal{C}^{(i)}, \mathcal{C}^{(i)})^{-1}(f(\mathcal{C}^{(i)}, \theta^{(i)}) - \boldsymbol{\mu}(\mathcal{C}^{(i)}))$
    $\theta^{(i+1)} \leftarrow \text{optimiserStep}(R^{(1)}_{\text{FSP}} + R^{(2)}_{\text{FSP}}, \theta^{(i)})$
    $i \leftarrow i + 1$
  return $\theta^{(\text{final})}$

Algorithm 2 linearised Laplace Approximation with GP Priors (Cinquin et al., 2024, Algorithm 2)

function FSP-LAPLACE(f, $\mathcal{GP}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathcal{C}$, $\mathbb{D}$, θ^*)
  $\boldsymbol{v} \leftarrow \boldsymbol{J}_{\theta^*}(\mathcal{C})\mathbf{1}/\|\boldsymbol{J}_{\theta^*}(\mathcal{C})\mathbf{1}\|_2$
  $\boldsymbol{L} \leftarrow \text{Lanczos}(\boldsymbol{\Sigma}(\mathcal{C}, \mathcal{C}), \boldsymbol{v})$
  $\boldsymbol{M} \leftarrow \boldsymbol{J}_{\theta^*}(\mathcal{C})^\top \boldsymbol{L}$
  $(\boldsymbol{U}_M, \boldsymbol{D}_M, \cdot) \leftarrow \text{svd}(\boldsymbol{M})$
  $\boldsymbol{A} \leftarrow \boldsymbol{D}_M^2 - \sum_{i=1}^{n} \boldsymbol{U}_M^\top \boldsymbol{J}_{\theta^*}(x^{(i)})^\top \boldsymbol{L}^{(i)}_{\theta^\star} \boldsymbol{J}_{\theta^*}(x^{(i)}) \boldsymbol{U}_M$
  $(\boldsymbol{U}_A, \boldsymbol{D}_A) \leftarrow \text{eig}(\boldsymbol{A})$
  $\boldsymbol{S} \leftarrow \boldsymbol{U}_M \boldsymbol{U}_A \boldsymbol{D}_A^{-1/2}$
  Find smallest k s.t. $\text{diag}(\boldsymbol{J}_{\theta^*}(\mathcal{C}_i)\boldsymbol{S}_{:,k:r}\boldsymbol{S}_{:,k:r}^\top \boldsymbol{J}_{\theta^*}(\mathcal{C}_i)^\top) \leq \text{diag}(\boldsymbol{\Sigma}(\mathcal{C}_i, \mathcal{C}_i))\ \forall i$
  return $\mathcal{N}(\theta^\star, \boldsymbol{S}_{:,k:r}\boldsymbol{S}_{:,k:r}^\top)$

with

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{\dagger}_{\theta^*} + \sum_{i=1}^{n} \boldsymbol{J}_{\theta^*}(x^{(i)})^\top \boldsymbol{L}^{(i)}_{\theta^*} \boldsymbol{J}_{\theta^*}(x^{(i)}), \quad \text{and} \tag{15}$$

$$\boldsymbol{L}^{(i)}_{\theta^*} = \nabla^2_f[-\log p(y^{(i)} \mid f)]_{f=f(x^{(i)},\theta^*)}. \tag{16}$$

Unlike the training phase, computing the Laplace approximation requires using a large number of context points to capture the prior beliefs accurately. While the prior precision $\boldsymbol{\Sigma}^{\dagger}_{\theta^*}$ only needs to be computed once after training, directly forming or storing Λ becomes computationally infeasible for large networks and extensive context sets. Since the RKHS inner products in $\boldsymbol{\Sigma}^{\dagger}_{\theta^*}$ do not admit closed-form expressions, we approximate the posterior covariance as $\Sigma^{\dagger}_{\theta^*} \approx \boldsymbol{J}_{\theta^*}(\mathcal{C})^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{J}_{\theta^*}(\mathcal{C})$. The choice of context points $\mathcal{C}$ for this phase is detailed in Section C.3. To address the computational challenges, Algorithm 2 employs an efficient procedure that avoids the explicit formation of Λ in high dimensions. The procedure begins by precomputing the prior precision once using a Lanczos process on $\boldsymbol{\Sigma}(\mathcal{C}, \mathcal{C})$ to obtain $\boldsymbol{L}$ such that $\boldsymbol{L}\boldsymbol{L}^\top \approx \boldsymbol{\Sigma}(\mathcal{C}, \mathcal{C})^{-1}$, avoiding repeated kernel inversions. Then the cross-covariance $\boldsymbol{M} = \boldsymbol{J}_{\theta^*}(\mathcal{C})^\top \boldsymbol{L}$ is computed using backward-mode automatic differentiation. On this, we operate a singular value decomposition $\boldsymbol{M} = \boldsymbol{U}_M \boldsymbol{D}_M \boldsymbol{V}_M^\top$ and form the matrix $\boldsymbol{A} = \boldsymbol{D}_M^2 - \sum_{i=1}^{n} \boldsymbol{U}_M^\top \boldsymbol{J}_{\theta^*}(x^{(i)})^\top L^{(i)}_{\theta^*} \boldsymbol{J}_{\theta^*}(x^{(i)}) \boldsymbol{U}_M$, where $\boldsymbol{L}^{(i)}_{\theta^*}$ is defined in eq. (16), followed by its eigendecomposition $(\boldsymbol{U}_A, \boldsymbol{D}_A)$. The untruncated posterior factor is then $\boldsymbol{S} = \boldsymbol{U}_M \boldsymbol{U}_A \boldsymbol{D}_A^{-1/2}$ To prevent exploding predictive variance caused by numerical instability when pseudo-inverting small eigenvalues, rank truncation is applied to regularise the posterior covariance. This leverages the property that, in linear-Gaussian models, the posterior precision $\boldsymbol{\Lambda}_{\text{posterior}} = \boldsymbol{\Lambda}_{\text{prior}} + \sum_{i=1}^{n} \boldsymbol{J}_i^\top \boldsymbol{L}_i \boldsymbol{J}_i \succeq \boldsymbol{\Lambda}_{prior}$ since $\boldsymbol{L}_i \succeq 0$, implying posterior variance $\leq$ prior variance.

Therefore, the smallest rank k is selected such that the posterior marginal variance at each context point remains bounded by the prior marginal variance. Specifically, k is chosen such that for all context points $\text{diag}\left(\boldsymbol{J}_{\theta^*}(\mathbf{c}_i)\boldsymbol{S}_{:,k:r}\boldsymbol{S}_{:,k:r}^\top \boldsymbol{J}_{\theta^*}(\mathbf{c}_i)^\top\right) \leq \text{diag}\left(\boldsymbol{\Sigma}(\mathbf{c}_i, \mathbf{c}_i)\right)$. This constraint provides a principled approach to eliminate unstable eigenpairs, i.e., eigenvalues with small magnitudes, while preserving meaningful uncertainty quantification, ensuring the posterior remains statistically consistent with the theoretical bounds of linear-Gaussian inference.

C.3 Choice of context points

The selection and evaluation of context points is crucial for both FSP training and posterior approximation, but serves different purposes in each phase.

Training phase: During FSP training, context points must be kept small ($n_{\mathcal{C}} \ll n$) for computational efficiency, as the regulariser requires solving a linear system in $n_{\mathcal{C}}$ dimensions at each optimiser step. An effective strategy is to sample context points i.i.d. from a distribution $P_{\mathcal{C}}$ at every training iteration. This stochastic sampling exposes the network to diverse regularisation constraints throughout training while avoiding the computational burden of large, fixed context sets. Common choices for $P_{\mathcal{C}}$ include uniform sampling over the input domain or sampling from additional unlabeled datasets. In some cases, it even makes sense to use the training batch $\mathcal{B}$ itself to compute the RKHS-regulariser $\|\cdot\|_{\mathbb{H}_\Sigma}$.

Posterior approximation: Unlike training, computing the Laplace approximation requires a large number of context points to accurately capture prior beliefs. Since this computation occurs only once after training, computational constraints are less stringent. The context points should adequately cover the regions where predictions will be made, ensuring well-calibrated uncertainty estimates beyond the immediate vicinity of training data. Low-discrepancy sequences (e.g., Halton sequences) are often preferred for their effective coverage of high-dimensional spaces.

The key principle is that context points should span the inference domain of interest, as the function-space prior only regularises the network at these locations. Insufficient coverage can lead to poorly calibrated uncertainties in uncovered regions.

D Experiment: Classification

Table 2: ECE results for Laplace approximation under different curvature approximations calibrated targeting the ECE on a validation set, either with grid search (GS) or gradient descent (GD). The MAP model has an ECE of 0.0755.

`CurvApprox`	GS	GD
`.FULL` (last layer)	0.0762	0.0755
`.LANCZOS`	**0.0166**	0.0754
`.DIAGONAL`	0.0281	0.0754

For the classification experiment, we train a three-layer Convolutional Neural Network on the CIFAR-10 dataset. Similarly to the regression case, we compare different curvature approximations and calibration strategies, evaluating their performance in terms of expected calibration error (ECE). Results are reported in Table 2.