bde: A Python Package for Bayesian Deep Ensembles via MILE

Vyron Arvanitis [1*], **Angelos Aslanidis** [1*], **Emanuel Sommer** [2,3*¶], and **David Rügamer** [3]

1 Faculty of Physics, LMU Munich, Munich, Germany **2** Department of Statistics, LMU Munich, Munich, Germany **3** Munich Center for Machine Learning, Munich, Germany ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 03 February 2026
Published: unpublished

Summary

bde is a Python package designed to bring state-of-the-art sampling-based Bayesian Deep Learning (BDL) to practitioners and researchers. The package combines the speed and high-performance capabilities of JAX and blackjax (Bradbury et al., 2018; Cabezas et al., 2024) with the user-friendly API of scikit-learn (Pedregosa et al., 2011). It targets tabular supervised learning tasks, including distributional regression and (multi-class) classification, providing a seamless interface for Bayesian Deep Ensembles (BDEs) (Sommer et al., 2024) specifically implementing **Microcanonical Langevin Ensembles (MILE)** (Sommer et al., 2025).

The workflow of bde implements the two-stage BDE inference process of MILE. First, it optimizes n_members independent instances of a flexibly configurable feed-forward neural network using regularized empirical risk minimization (with the negative log-likelihood as loss) via the AdamW optimizer (Loshchilov & Hutter, 2019). Second, it transitions to a sampling phase using Microcanonical Langevin Monte Carlo (Robnik et al., 2023; Robnik & Seljak, 2024), enhanced with a tuning phase adapted for Bayesian Neural Networks. This combination is referred to as MILE (Sommer et al., 2025). In essence, the optimization of the ensemble of neural networks first finds diverse high-likelihood modes, from which sampling then explores local posterior structure. This process generates an ensemble of samples (models) that constitute an implicit posterior approximation.

Key Software Design Feature. Because optimization and sampling across ensemble members are independent, bde exploits JAX's parallelization and just-in-time compilation to scale efficiently across CPUs, GPUs, and TPUs. Given new test data, the package approximates the posterior predictive, enabling point predictions, credible intervals, coverage estimates, and other uncertainty metrics through a unified interface.

Statement of Need and State of the Field

Reliable uncertainty quantification (UQ) is increasingly viewed as a critical component of modern machine learning systems, and BDL provides a principled framework for achieving it (Papamarkou et al., 2024). While several libraries support optimization-based approaches such as variational inference or classical Bayesian modeling, accessible tools for sampling-based inference in Bayesian neural networks remain scarce. Existing probabilistic programming frameworks offer MCMC but require substantial manual configuration to achieve competitive performance on neural network models.

bde addresses this gap by providing the first user-friendly implementation of MILE - a hybrid sampling technique shown to deliver strong predictive accuracy and calibrated uncertainty for

Bayesian neural networks (Sommer et al., 2025). By providing full scikit-learn compatibility, the package enables seamless integration into existing machine learning workflows, allowing users to obtain principled Bayesian uncertainty estimates without specialized knowledge of MCMC dynamics, initialization strategies, or JAX internals.

Through automated orchestration of optimization, sampling, parallelization, and predictive inference, `bde` offers a fast, reproducible, and practical solution for applying sampling-based BDL methods to tabular supervised learning tasks.

Research Impact. `bde` bridges the gap between high-performance MCMC research and practical data science. By providing a curated implementation of MILE for tabular data, it enables researchers and practitioners alike to easily apply BDEs. Its inclusion in the `scikit-learn-contrib` organization ensures adherence to rigorous software standards and API consistency, making it a trusted, community-ready tool for reproducible UQ in tabular machine learning.

Usage Example

The following example illustrates a regression task with UQ using `bde` in only a few lines of code. Training inputs are assumed to be preprocessed and normalized. The workflow specifies the ensemble and sampling hyperparameters, fits the model, and obtains posterior predictive quantities, including predictive moments, credible intervals, and raw ensemble outputs.

```python
from bde import BdeRegressor

regressor = BdeRegressor(
        n_members=8, # parallelizes over the available cores
        hidden_layers=[16, 16], # 2 hidden layers of width 16
        activation="relu", # (default)
        epochs=1000,
        validation_split=0.15,
        lr=1e-3,
        weight_decay=1e-4,
        patience=20,
        warmup_steps=5000,
        n_samples=200, # 200/10 x 8 = 160 final posterior samples
        n_thinning=10,
        # Controls MCMC exploration (default):
        desired_energy_var_start=0.5,
        desired_energy_var_end=0.1,
)

regressor.fit(x=X_train, y=y_train)

means, sigmas = regressor.predict(X_test, mean_and_std=True)
means, intervals = regressor.predict(X_test, credible_intervals=[0.1, 0.9])
raw = regressor.predict(X_test, raw=True)
```

Classification follows analogously using `BdeClassifier`.

Regression Benchmark

We provide a small benchmark of `bde` on the `airfoil` (Dua & Graff, 2017) and the `bikesharing` (Fanaee-T, 2013) datasets. We report mean predictive performance (RMSE), UQ metrics (NLL in the distributional and mean regression formulation), reported as mean ± standard deviation over 5 independent runs. The results show competitive out-of-the-box performance of BDE especially in UQ with its native distributional regression capability.

`airfoil`	RMSE	NLL (distr. regr.)	NLL (mean regr.)
Linear Model	0.6598 ± 0.0000	-	1.0032 ± 0.0000
Random Forest	0.2560 ± 0.0015	-	0.0567 ± 0.0057
XGBoost	0.2025 ± 0.0055	-	-0.1782 ± 0.0269
CatBoost	0.2393 ± 0.0036	0.1479 ± 0.1414	-0.0109 ± 0.0152
Deep Ensemble	0.3650 ± 0.0038	0.1760 ± 0.0043	0.4100 ± 0.0105
TabPFN (V2)	0.1359 ± 0.0028	**-0.9338 ± 0.0195**	-0.5769 ± 0.0205
BDE	**0.1215 ± 0.0042**	-0.9126 ± 0.0131	**-0.6888 ± 0.0347**

`bikesharing`	RMSE	NLL (distr. regr.)	NLL (mean regr.)
Linear Model	0.7796 ± 0.0000	-	1.1700 ± 0.0000
Random Forest	0.2440 ± 0.0020	-	0.0085 ± 0.0009
XGBoost	0.2143 ± 0.0010	-	-0.1215 ± 0.0049
CatBoost	0.2652 ± 0.0021	-0.3737 ± 0.0229	0.0918 ± 0.0080
Deep Ensemble	0.2880 ± 0.0052	-0.4550 ± 0.0173	0.1740 ± 0.0181
TabPFN (V2)	**0.2103 ± 0.0008**	-0.6856 ± 0.0063	**-0.1400 ± 0.0041**
BDE	0.2261 ± 0.0016	**-0.7315 ± 0.0098**	-0.0679 ± 0.0071

All models used 10 CPU cores without additional tuning. For BDE, we generated 1000 posterior samples from a feed-forward neural network with four hidden layers of width 16. While even highly optimized BDEs generally incur a higher computational cost than optimization-based competitors due to the iterative nature of MCMC sampling, this investment enables the construction of a flexible, non-parametric posterior approximation. This trade-off, as demonstrated above, yields strong performance and rigorous epistemic UQ, e.g. through calibrated credible intervals. All experimental configurations are provided in the released codebase to ensure reproducibility.

AI usage

Generative AI was used via GitHub Copilot for local, line- or block-level code autocompletion during software development, using the Claude Sonnet 3.7 and 4 models. Codex was additionally used to assist with the initial draft of the package documentation and minor refactors to ensure compatibility with the `scikit-learn` API. No AI tools were used for ideation, architectural or design decisions, code review, or testing strategy. All AI-generated suggestions were critically reviewed, modified where necessary, and fully validated by the authors, who retain complete responsibility.

References

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/google/jax

Cabezas, A., Corenflos, A., Lao, J., & Louf, R. (2024). *BlackJAX: Composable Bayesian inference in JAX*. https://arxiv.org/abs/2402.10797

Dua, D., & Graff, C. (2017). *UCI machine learning repository*. University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml

Fanaee-T, H. (2013). *Bike Sharing Dataset*. UCI Machine Learning Repository.

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. *International Conference on Learning Representations*.

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., … Zhang, R. (2024). Position: Bayesian Deep Learning is Needed in the Age of Large-Scale AI. *Proceedings of the 41st International Conference on Machine Learning*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Robnik, J., De Luca, G. B., Silverstein, E., & Seljak, U. (2023). Microcanonical hamiltonian monte carlo. *The Journal of Machine Learning Research*, *24*(1), 14696–14729.

Robnik, J., & Seljak, U. (2024). Fluctuation without dissipation: Microcanonical langevin monte carlo. *Symposium on Advances in Approximate Bayesian Inference*, 111–126.

Sommer, E., Robnik, J., Nozadze, G., Seljak, U., & Rügamer, D. (2025). Microcanonical langevin ensembles: Advancing the sampling of bayesian neural networks. *The Thirteenth International Conference on Learning Representations*.

Sommer, E., Wimmer, L., Papamarkou, T., Bothmann, L., Bischl, B., & Rügamer, D. (2024). Connecting the dots: Is mode-connectedness the key to feasible sample-based inference in bayesian neural networks? *Proceedings of the 41st International Conference on Machine Learning*.