DMRN+20: DIGITAL MUSIC RESEARCH NETWORK
ONE-DAY WORKSHOP 2025

KING'S COLLEGE LONDON
TUE 16 DECEMBER 2025

Korvax: Audio Processing and Machine Learning in JAX

David Marttila* and Joshua D. Reiss

Centre for Digital Music, Queen Mary University of London, United Kingdom, d.sudholt@qmul.ac.uk

***Abstract*— We present Korvax, a package for audio processing and machine learning in JAX. Korvax provides differentiable, GPU-ready implementations of common audio loss functions and signal processing utilities. We benchmark the performance on CPU and GPU and find that Korvax can achieve noticeable speedups compared to equivalent PyTorch libraries.**

I. INTRODUCTION

PyTorch is the prevalent framework in the audio machine learning community. Many common workflows are implemented in the officially supported Torchaudio library or a mature ecosystem of community-maintained PyTorch packages.

JAX [1] is a newer framework for high-performance array-oriented computations in Python that has gained popularity in recent years, positioning itself as one of the main alternatives to PyTorch. It focuses on providing function transformations for just-in-time compilation, vectorization, automatic differentiation, and parallelization. This allows users to compose complex graphs of numerical operations in Python and execute them efficiently, often faster than in other frameworks.

Despite these advantages, JAX has seen little adoption in audio signal processing, likely in part due to the lack of domain-specific tooling compared to PyTorch. Korvax aims to address this gap by porting commonly-used features from Torchaudio, librosa [2], and other packages into JAX. The source code of Korvax and documentation of currently implemented features is available online.[1] We welcome feedback and contributions to advance the state of audio machine learning research in JAX.

II. PERFORMANCE BENCHMARKS

We benchmark selected Korvax features against reference PyTorch packages. The results are shown in Table 1.

Loss Functions: Korvax provides a general interface for calculating frame-wise losses at multiple resolutions. We benchmark its performance by generating two random audio signals x, y with a length of 64000 samples and a batch size of 256. We measure the time it takes to compute the loss $f(x, y)$ and the partial gradient of f w.r.t. x.

For the multi-scale spectral loss (MSS) benchmark, we use 6 resolutions and compare to the auraloss [3] PyTorch package. Korvax computes the gradients consistently about 2.5x faster. We also benchmark spectral optimal transport (SOT) [4]. We use only one resolution and compare to the sot-loss[2] PyTorch package. The choice of framework has a significant effect on performance: Korvax is 2–3x slower than sot-loss on the CPU, but dramatically faster on the GPU, by a factor of about 40x.

Table 1: Performance comparison of Korvax with PyTorch packages. The time taken to compute a given value is measured in seconds and averaged over 100 runs. The CPU is an Apple M2 Pro chip with 12 cores. The GPU is an NVIDIA H100.

		CPU		GPU	
		$f(x,y)$	$\frac{\partial f(x,y)}{\partial x}$	$f(x,y)$	$\frac{\partial f(x,y)}{\partial x}$
MSS	Korvax	0.608	0.993	0.010	0.018
	auraloss	0.966	2.450	0.032	0.045
SOT	Korvax	1.588	1.957	0.039	0.043
	sot-loss	0.554	0.844	1.443	1.760
LPC	Korvax	0.022	0.091	0.106	0.198
	torchlpc	0.016	0.153	0.097	0.197

Time-Varying All-Pole Filters: Korvax implements efficient differentiation of linear predictive coding (LPC) with time-varying all-pole filters as proposed in [5], using the torchlpc[3] PyTorch package as a reference. We compute the sum of the squared filtered signal as $f(x, y)$ and the partial gradient of f w.r.t. x, where x are the time-varying filter coefficients and y is the input audio signal. In both implementations, the actual filtering code is written in C++ and compiled ahead-of-time. This means that there is less room for performance difference due to the choice of framework.

We perform the benchmark using randomly generated audio signals with a length of 64000 samples, sample-wise coefficients with order 16, and a batch size of 128. On the GPU, both implementations run at similar speeds. On the CPU, Korvax is slightly slower during the forward pass, but noticeably faster while computing the gradients.

III. REFERENCES

[1] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[2] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python." in *SciPy*, 2015.

[3] C. J. Steinmetz and J. D. Reiss, "auraloss: Audio focused loss functions in PyTorch," in *DMRN+15*, 2020.

[4] B. Torres, G. Peeters, and G. Richard, "Unsupervised Harmonic Parameter Estimation Using Differentiable DSP and Spectral Optimal Transport," in *ICASSP*, 2024.

[5] C.-Y. Yu, C. Mitcheltree, A. Carson, S. Bilbao, J. D. Reiss, and G. Fazekas, "Differentiable all-pole filters for time-varying audio systems," in *DAFx*, 2024.

*D. Marttila is supported by UK Research and Innovation [grant number EP/S022694/1].

[1] https://github.com/davidmarttila/korvax

[2] github.com/bernardo-torres/spectral-optimal-transport/

[3] https://github.com/DiffAPF/torchlpc