LARS 2.0: A DRUMS DEMIXING PLUGIN

Giuliano Galadini, Francisco Messina

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano
Piazza Leonardo Da Vinci 32, 20122 Milano, Italy
[giuliano.galadini,francisco.messina]@mail.polimi.it

ABSTRACT

The field of music source separation has seen remarkable progress with the advent of deep learning, yet the integration of deep learning models into music production applications has been limited. This work introduces LARS 2.0, a VST3/Standalone plugin designed for high-fidelity drum source separation. The plugin integrates the state-of-the-art MDX23C, a U-Net based architecture that achieved 3rd place in the 2023 Sound Demixing Challenge. We implement and evaluate two checkpoints of the MDX23C architecture, one from Aufr33 and Jarredou, the other one from Politecnico di Milano. Furthermore, we explore a cascaded separation pipeline to isolate drum stems from full musical mixes. Objective evaluations using the signal-to-distortion ratio (SDR) on the StemGMD dataset show that the first model achieves a mean SDR of 11.14 dB, outperforming the second one. While the cascaded approach extends the plugin's utility, it suffers from significant performance degradation with respect to a direct separation on a percussive track, rather than a two-phase separation from a full mix. The plugin is implemented in C++ with the JUCE framework, featuring a user-friendly interface and multithreaded processing for efficient operation.

Index Terms— Audio plugin, drum source separation, JUCE, music demixing, TorchScript, LibTorch

1. INTRODUCTION

The field of music demixing has witnessed significant advancements over recent years, driven by the rise of deep learning techniques in music signal processing. Music demixing generally involves separating individual musical sources, such as vocals, instruments, or drums, from a mixed audio signal. Its applications span a wide range of music production tasks, including remixing, mastering, and sampling.

The advent of deep learning has been a transformative force in music source separation. Models like the U-Nets [1] and convolutional neural networks (CNNs) have demonstrated remarkable efficacy in separating vocals and musical sources with unprecedented performance [2]. For instance, Liu et al. [3] propose a confidence-based ensemble approach that automatically selects from multiple pre-trained separation models, significantly improving separation quality for diverse musical sources without the need for ground-truth references. These innovations have pushed the boundaries of what is achievable in music demixing, offering professional-grade quality even for complex mixtures.

Within the overarching domain of music demixing, drums demixing has emerged as a specialized and impactful task, which focuses specifically on isolating individual drum stems, such as the kick, snare, and hi-hats, from a mixture. While conceptually similar to general music demixing, drum source separation is particularly challenging due to the percussive and transient nature of drum sounds, which feature sharp attacks and broadband frequency content rather than continuous harmonic tones.

Large-scale, high-quality datasets such as MUSDB18 [4] and StemGMD [5] have played a pivotal role in this progress. MUSDB18 comprises 150 full-track songs covering various musical genres and includes both stereo mixtures and their corresponding isolated sources. Similarly, StemGMD includes 1224 hours of isolated drum recordings synthesized from extensive MIDI mappings.

Early efforts in drum source separation relied on traditional signal processing approaches like Nonnegative Matrix Factorization (NMF) [6] and Nonnegative Matrix Factor Deconvolution (NMFD) [7]. While these methods achieved moderate success, they were often hindered by inter-channel leaks and spectral artifacts, limiting their utility in professional applications.

The introduction of deep learning architectures has revolutionized drum source separation. State-of-the-art models like LarsNet employ multiple U-Nets to isolate drum stems, leveraging time-frequency domain masking to achieve high-fidelity separation [8]. Similarly, HT-Demucs integrates both time-domain and time-frequency processing, demonstrating robust generalization to unseen drum kits [9]. Meanwhile, BS-RoFormer adopts a transformer-based approach to excel in music demixing tasks, though its performance on drum source separation is constrained by the limited timbre diversity in drum datasets [10]. Notably, the confidence-based ensemble approach demonstrated by Liu et al. [3] highlights the potential for combining multiple pre-trained models to address the unique challenges posed by drum demixing tasks.

Despite these advancements, challenges remain in the field of drum source separation. Techniques like Band-Split processing struggle to separate instruments with overlapping frequency ranges while preserving the natural timbre of drums [11]. Additionally, achieving a balance between real-time processing and high-quality separation continues to constrain practical applications [12].

In this context, our work presents a novel drums demixing plugin, Lars 2.0, which builds upon the foundations of its predecessor, Lars [13]. By leveraging the MDX23C architecture and combining it with a user-friendly interface, Lars 2.0 delivers high-fidelity separation of drum stems. Unlike other solutions, such as iZotope's Music Rebalance plug-in[1], which offers stem isolation within a broader mixing suite, or web-based services like lalal.ai[2], known for quick AI-driven vocal and instrument extraction, and UVR online[3], which provides an accessible online interface for general source separation, Lars 2.0 focuses specifically on drums, delivering minimal

[1]https://www.izotope.com/en/products/rx/features/music-rebalance
[2]https://www.lalal.ai/
[3]https://uvronline.app/ai

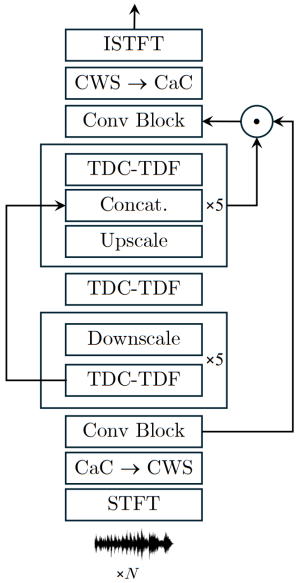

Figure 1: MDX23C architecture. Image from [11].

artifact results tailored for producers' needs. The tool offers high-quality stem isolation with seamless integration into popular DAWs. This report provides an in-depth review of the development process, evaluation results, and practical use cases for the proposed plugin.

2. MODEL ARCHITECTURE: MDX23C

2.1. Background and Context

The proposed plugin utilizes MDX23C, a state-of-the-art architecture for music source separation originally developed for the 2023 SDX Challenge (MDX track, leaderboard C) where it achieved 3rd place [14]. This architecture builds upon the KUIELab-MDX-Net framework [15], representing an advanced implementation of the TDF-TDC-U-Net architecture first introduced for singing voice separation [16].

2.2. Architectural Overview

The MDX23C architecture employs a U-Net structure where conventional convolutional layers are replaced with specialized TFC-TDF blocks as seen in Fig. 1. Each block processes audio through:

- Time-Frequency Convolution (TFC) layers with normalization and pre-activation
- Time-Distributed Fully-connected (TDF) layers operating on frame-level latent representations and processed again by a second TFC block
- Complex-as-Channels (CaC) processing of STFT coefficients

Key innovations include a CaC to Channel-Wise Subband (CWS) transformation module, which partitions frequency bins into K subbands, allowing the model to learn different weights for each subband [17].

The MDX23C architecture used features:

- 5 encoder/decoder blocks with channel expansion (128 channels/block)
- Dual STFT resolutions ($N_{\text{FFT}} = 4096$ and 8192)
- Hybrid loss function combining:

$$\mathcal{L}_{\text{MDX23C}} = \|x_i - \hat{x}_i\|_1 + \sum_{s=1}^{S} \|\mathbf{X}_i^{(s)} - \hat{\mathbf{X}}_i^{(s)}\|_1 \,, \qquad (1)$$

where x_i and $\hat{x}_i$ are the ground-truth and predicted stem waveforms, and $\mathbf{X}_i$ and $\hat{\mathbf{X}}_i$ are the ground-truth and predicted magnitude STFTs, respectively, at resolution level $s = 1, ..., S$. The multi-resolution STFT loss enhances spectral fidelity [18]. The architecture processes stereo inputs through independent complex-valued operations while maintaining temporal consistency via skip connections.

3. IMPLEMENTATION DETAILS

For our plugin, written in C++ using the JUCE framework[4], we provide three distinct separation models, all grounded in the MDX23C framework.

The first model follows the architecture made available by ZFTurbo[5] and was trained by Aufr33 and Jarredou. From this point forward, we will refer to it as the "small model". This model separates the drum track into six individual stems: kick, snare, hi-hat, tom, crash, and ride.

Another implementation of MDX23C, trained by the Image and Sound Processing Lab at the Politecnico di Milano, will be referred to as the "large model". It separates the drums into five stems: kick, snare, hi-hat, tom, and a grouped cymbals stem (which includes crash and ride).

Although the difference in the number of parameters between the two is modest, the small model has approximately 109.4 million parameters, while the large model has approximately 109.9 million, we adopt these labels to reflect the slightly larger architectural footprint and higher computational cost observed in the latter.

The third model performs a full separation of a track into four broad stems: drums, vocals, bass, and other instruments. This is accomplished using a multi-stem MDX23C model from ZFTurbo, trained on the MUSDB18HQ dataset. The resulting drum stem is then further processed using the small model to extract individual drum components. In this way, the user can obtain isolated drum stems, such as kick, snare, toms, hi-hats, crash, and ride, even when starting from a full mix. From this point forward, we refer to this two-stage configuration as the "cascaded model".

3.1. Interaction

Our implementation offers essential functionalities for audio playback and separation. The plugin consists of two sections. The first section allows users to either drop an audio file (in formats such as MP3, WAV, or FLAC) or load it using a designated button. Users can play and stop the file using the provided control, and the audio will loop once it reaches the end. Additionally, users can move the transport bar to navigate to specific positions in the file or double-click on the waveform to jump directly to the beginning. A button for initiating the separation process is also available, which generates new WAV files.

[4] https://juce.com/

[5] https://github.com/ZFTurbo/Music-Source-Separation-Training

In the second section of the plugin, the same transport options are present. Based on the selected model, there are buttons available for each separated track. Users can push them to select a specific track or drag and drop them to transfer the separated tracks into an audio track within a Digital Audio Workstation (DAW) or any other location in the computer. A "Save All Tracks" button facilitates quick saving of all separated stems.

Users can easily change the separation model by utilizing the drop-down menu located at the top of the GUI.

Figure 2: Lars 2.0 GUI.

To optimize the performance of our plugin, we adopted a multithreaded approach, as the separation process can be time-consuming, especially when executed on a consumer-grade CPU. This design choice allows users to play and stop playback while the separation process continues in the background. For more details about our implementation, please refer to our repository [6].

3.2. Model Conversion via TorchScript

To deploy our MDX23C-based separation models within a C++ plugin framework, we converted the original Python-trained networks into TorchScript [7] modules compatible with LibTorch [8], the official C++ distribution of PyTorch [9]. Starting from each MDX23C implementation, whether the small, large, or cascaded model, we first ensured that the model was structured in a TorchScript-friendly manner. In practice, this meant annotating all tensor inputs and outputs with explicit `torch.Tensor` types and refactoring any dynamic Python constructs (for example, loops over heterogeneous lists or runtime attribute assignments) into statically analyzable code paths. By doing so, the `torch.jit.script()` mechanism could successfully trace and compile the entire computation graph without ambiguities.

Once the type annotations and code adjustments were in place, we invoked TorchScript on the model instance already loaded with its checkpoint and configuration. The resulting serialized `.pt` file encapsulates the full network, eliminating any dependence on the original Python runtime. In our C++ plugin code, this TorchScript artifact is loaded via LibTorch's `torch::jit::load()` API. All input and output tensors are explicitly declared (e.g., `torch::Tensor`), which satisfies C++'s static typing requirements and enables seamless forward passes.

3.3. Preprocessing and postprocessing

In C++, we handle all preprocessing and postprocessing steps required to interface audio data with the TorchScript model. First, the incoming JUCE `AudioBuffer` is converted into a tensor, preserving sample rate and channel layout. Once in tensor form, the waveform is segmented into smaller chunks using a sliding window with configurable overlap. Rather than processing each window independently, we batch up several windows together to run through the separation model in a single inference call. The model outputs for each window are then accumulated into a larger result tensor using an overlap-add strategy. Simultaneously, a parallel counter tensor tracks the number of contributions at each sample index, allowing us to average the overlapping regions by dividing the accumulated result by the counter values. Any NaNs introduced during processing are detected and replaced with zeros to prevent artifacts. Finally, we remove any padding applied during segmentation so that the output tensor matches the original audio length. The fully reconstructed, overlap-averaged tensor is then converted back into a JUCE `AudioBuffer`, ready for playback or export as demixed audio. This end-to-end C++ pipeline ensures that all audio transformations occur efficiently without resorting to Python, allowing Lars 2.0 to deliver accurate, high-quality drum stems directly within the JUCE plugin environment.

[6] https://github.com/FranciscoMessina00/Lars-2.0
[7] https://docs.pytorch.org/docs/stable/jit.html
[8] https://docs.pytorch.org/cppdocs/installing.html
[9] https://pytorch.org/

4. RESULTS

To assess the separation quality of our first two MDX23C-based models (small and large), we conducted objective evaluations on four drum sets from the StemGMD dataset. For each drum set, we isolated the kick, snare, toms, hi-hats, and cymbals stems, where cymbals combines the ride and crash outputs of the small model so that both models use the same five-stem grouping, and evaluated separation performance using the signal-to-distortion ratio (SDR), defined as

$$\mathrm{SDR} = 10 \log_{10} \frac{\|s_{\text{target}}\|^2}{\|s_{\text{target}} - \hat{s}\|^2}, \tag{2}$$

where s_{target} is the clean reference signal, $\hat{s}$ is the estimated (separated) signal, and $\|\cdot\|$ denotes the Euclidean norm.

Overall, both models achieved very similar performance trends across all drum components. Figure 3 plots the mean SDR for each component, comparing the two models side by side. The kick drum consistently exhibited the highest separation quality (mean SDR 26.5 dB), followed by the snare (22.0 dB). In contrast, hi-hats, cymbals, and toms showed the lowest mean SDR values (5.0 dB, -0.5 dB, and -4.5 dB, respectively), which may suggest increased difficulty in isolating high-frequency or less prominent elements.

However, informal listening and dataset inspection revealed that many of the test tracks contained few or no tom or cymbal hits. As a result, the lower SDR values for these components are likely due not only to model limitations but also to increased leakage from other instruments when the true source is mostly silent. This highlights the need to interpret SDR scores in context, particularly when evaluating sparse or infrequent sources.

When averaged over all drum components, the small model achieved an overall mean SDR of 11.14 dB, while the large model reached 9.05 dB. This difference of over 2 dB indicates a noticeable performance gap, with the small model having shown consistently stronger results across drum types. While both models followed similar performance patterns, excelling on kick and snare, and struggling with cymbals, hi-hats, and toms, the smaller model offers higher separation quality overall.

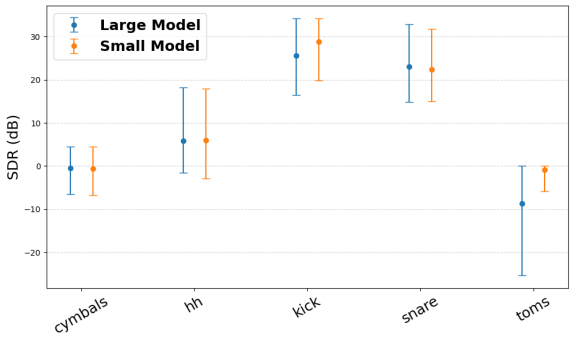

Figure 3: SDR comparison between the two models.

Model	RTR at 15 s	RTR at 30 s	RTR at 45 s
Small	4.13	3.51	3.36
Large	8.20	7.03	6.72

Table 1: Real-time ratios for different loop lengths.

4.1. Cascaded model Separation

In addition to evaluating our two single-stage MDX23C variants, we also tested the cascaded model separation pipeline. This two-step approach emulates a real-world scenario where a user may lack access to isolated drum tracks and must recover them from a full mix.

We applied this pipeline to a small set of commercially released tracks covering diverse arrangements. SDR was computed on the recovered kick, snare, toms, hi-hats, and cymbals exactly as in our previous evaluations. Across nearly all tracks and drum components, cascaded SDR values clustered around 0 dB, indicating very poor separation quality. In many cases, the residual bleed and artifacts introduced in the first stage overwhelmed the second pass, especially when multiple non-drum instruments were layered heavily over the drum track.

While this cascaded method does extend Lars 2.0 applicability to arbitrary full-length songs, the consistently low SDR scores reveal that accumulated errors from sequential passes significantly degrade output quality. These results highlight the need for more robust multi-stage strategies, such as joint end-to-end training of both separation stages or dedicated post-processing denoising, to make full-mix drums demixing viable in practice.

4.2. Real-Time Ratio

We assess processing efficiency via the real-time ratio (RTR), defined as [13]

$$\mathrm{RTR} = \frac{T_\mathrm{s}}{T_\mathrm{a}}, \tag{3}$$

where T_s is the time required to separate a given audio segment, and T_a is the duration of that segment. An RTR greater than 1 indicates that processing is slower than real-time playback.

All measurements were carried out on an Intel Core 7-150U CPU. To investigate dependence on signal length, we ran each model on three loop lengths: 15 s, 30 s, and 45 s. The resulting RTRs for the small and large models are reported in Table 1.

Both models exhibit a slight decrease in RTR for longer segments, and the large one runs at roughly twice the RTR of the small one across all lengths.

4.3. Discussion

The objective results highlight a clear performance gap between the two main models, with the small model achieving a mean SDR over 2 dB higher than the larger one. Both models demonstrated strong performance in separating kick (26.5 dB) and snare (22.0 dB) drums, which have distinct and powerful transient characteristics. Conversely, components like hi-hats, toms, and cymbals yielded significantly lower SDR scores. As noted, this may be partially attributed to the sparsity of these instruments in the test data, where low source energy can amplify the impact of bleed from other stems, rather than solely indicating a failure of the model.

Our investigation into a cascaded separation pipeline, designed to extract drum stems from a full mix, proved to be a challenging endeavor. The resulting SDR values were consistently poor, hovering around 0 dB, which indicated that errors and artifacts from the initial separation stage were compounded in the second stage. This suggests that for effective real-world application on full mixes, a more sophisticated, possibly end-to-end trained, multi-stage approach is necessary to mitigate quality loss.

From an efficiency standpoint, the real-time ratio (RTR) analysis showed that processing is not yet real-time on a consumer-grade CPU. The small model was notably more efficient than the large one. The slight decrease in RTR for longer audio segments suggests that the overhead associated with initializing the process is a factor, and performance becomes marginally more efficient as the duration increases.

5. CONCLUSIONS

In this work, we presented LARS 2.0, a drums demixing plugin that leverages the state-of-the-art MDX23C architecture.

LARS 2.0 provides a powerful and user-friendly tool for drum separation, particularly when working with isolated drum mixes. The integration of the high-performing architecture from ZFTurbo ensures state-of-the-art results directly within a DAW. Future work should focus on improving the performance of the cascaded separation to make full-mix drum extraction viable. This could involve exploring joint training of the two separation stages or developing dedicated post-processing techniques to reduce artifacts introduced during the initial separation. Furthermore, optimizing the models for lower computational complexity could bring the plugin's performance closer to real-time, expanding its utility for producers and musicians.

6. REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, 2015.

[2] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde, "Singing voice separation with deep u-net convolutional networks," *18th International Society for Music Information Retrieval Conference*, 2017.

[3] A. Liu, P. Seetharaman, and B. Pardo, "Model selection for deep audio source separation via clustering analysis," *arXiv preprint arXiv:1910.12626*, 2019.

[4] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "The musdb18 corpus for music separation," 2017.

[5] A. I. Mezza, R. Giampiccolo, A. Bernardini, A. Sarti, *et al.*, "Stemgmd: A large-scale multi-kit audio dataset for deep drums demixing," in *SDX Workshop 2023*, 2023.

[6] C. Dittmar and D. Gärtner, "Real-time transcription and separation of drum recordings based on nmf decomposition." in *DAFx*, 2014.

[7] P. Smaragdis, "Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs," in *Independent Component Analysis and Blind Signal Separation: Fifth International Conference, ICA 2004, Granada, Spain, September 22-24, 2004. Proceedings 5*, 2004.

[8] A. I. Mezza, R. Giampiccolo, A. Bernardini, and A. Sarti, "Toward deep drum source separation," *Pattern Recognition Letters*, 2024.

[9] S. Rouard, F. Massa, and A. Défossez, "Hybrid transformers for music source separation," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[10] W.-T. Lu, J.-C. Wang, Q. Kong, and Y.-N. Hung, "Music source separation with band-split rope transformer," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024.

[11] A. I. Mezza, R. Giampiccolo, A. Bernardini, and A. Sarti, "Benchmarking music demixing models for deep drum source separation," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024.

[12] S. Venkatesh, A. Benilov, P. Coleman, and F. Roskam, "Real-time low-latency music source separation using hybrid spectrogram-tasnet," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024.

[13] A. Mezza, R. Palma, E. Morena, A. Orsatti, R. Giampiccolo, A. Bernardini, and A. Sarti, "Lars: An open-source vst3 plugin for deep drums demixing with pretrained models," 2023.

[14] G. Fabbro, S. Uhlich, C.-H. Lai, W. Choi, M. Martínez-Ramírez, W. Liao, I. Gadelha, G. Ramos, E. Hsu, H. Rodrigues, F.-R. Stöter, A. Défossez, Y. Luo, J. Yu, D. Chakraborty, S. Mohanty, R. Solovyev, A. Stempkovskiy, T. Habruseva, N. Goswami, T. Harada, M. Kim, J. Hyung Lee, Y. Dong, X. Zhang, J. Liu, and Y. Mitsufuji, "The sound demixing challenge 2023 – music demixing track," *Transactions of the International Society for Music Information Retrieval*, 2024.

[15] M. Kim, W. Choi, J. Chung, D. Lee, and S. Jung, "Kuielab-mdx-net: A two-stream neural network for music demixing," *arXiv preprint arXiv:2111.12203*, 2021.

[16] W. Choi, M. Kim, J. Chung, D. Lee, and S. Jung, "Investigating u-nets with various intermediate blocks for spectrogram-based singing voice separation," *arXiv preprint arXiv:1912.02591*, 2019.

[17] H. Liu, L. Xie, J. Wu, and G. Yang, "Channel-wise subband input for better voice and accompaniment separation on high resolution music," in *Interspeech 2020*, 2020.

[18] C. J. Steinmetz and J. D. Reiss, "auraloss: Audio focused loss functions in pytorch," in *Digital music research network one-day workshop (DMRN+ 15)*, 2020.