

# UNMIXER: AN INTERFACE FOR EXTRACTING AND REMIXING LOOPS

Jordan B. L. Smith    Yuta Kawasaki    Masataka Goto

National Institute of Advanced Industrial Science and Technology (AIST), Japan

## ABSTRACT


To create their art, remix artists would like to have segmented stem tracks at their disposal; that is, isolated instances of the loops and sounds that the original composer used to create a track. We present Unmixer, a web service that will analyze and extract loops from any audio uploaded by a user. The loops are presented in an interface that allows users to immediately remix the loops; if users upload multiple tracks, they can easily create mash-ups with the loops, which are automatically matched in tempo. To analyze the audio, we use a recently-proposed method of source separation based on the nonnegative Tucker decomposition of the spectrum. To reduce interference among the extracted loops, we propose an extra factorization step with a sparseness constraint and demonstrate that it improves the source separation result. We also propose a method for selecting the best instances of the extracted loops and demonstrate its effectiveness in an evaluation. Both of these improvements are incorporated into the backend of the interface. Finally, we discuss the feedback collected in a set of user evaluations.

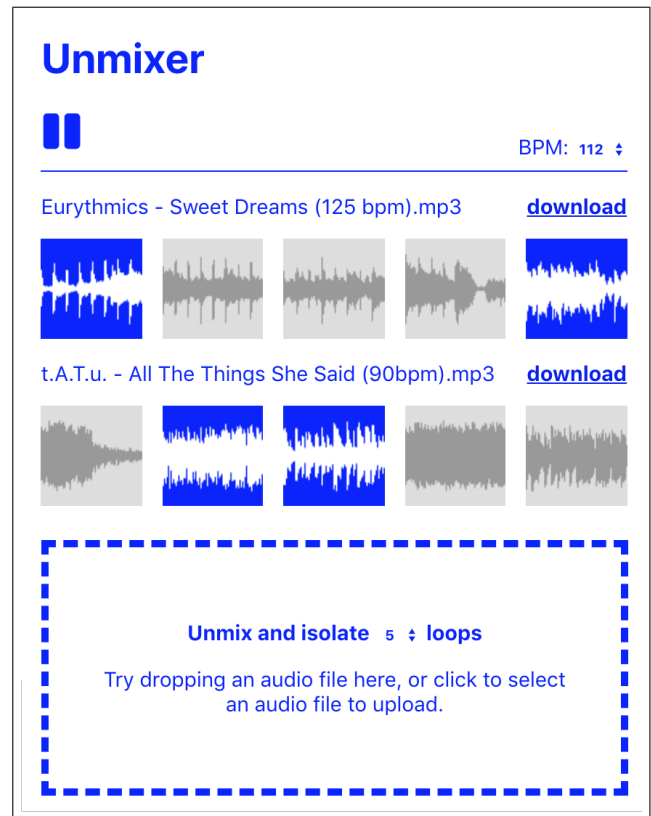
## 1. INTRODUCTION

Professional and amateur composers across the world enjoy creating remixes and mashups. Remixes are pieces of music that are composed, in whole or in part, using snippets of another audio recording, whereas mashups juxtapose snippets of two or more recordings [8]. Creators of official remixes usually have access to the stem tracks for a recording, but these resources are not typically available to amateur remixers. Unofficial remixes, sometimes called ‘bootlegs’, can still be created using clips of the down-mixed recording of the song [8], but this presents two challenges: first, it is time-consuming to manually segment an audio track to select the most prominent or interesting bars or sounds. Second, because the sources in the original audio recording are down-mixed, the artist may have to use equalization (i.e., filtering out certain frequencies) to achieve a simple separation of sources.

We present Unmixer<sup>1</sup>, an interface that accomplishes

<sup>1</sup> Available at: <https://unmixer.ongaaccel.jp>

 © Jordan B. L. Smith, Yuta Kawasaki, Masataka Goto. Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** Jordan B. L. Smith, Yuta Kawasaki, Masataka Goto. “Unmixer: An interface for extracting and remixing loops”, 20th International Society for Music Information Retrieval Conference, Delft, The Netherlands, 2019.



**Figure 1.** Screenshot of Unmixer interface with two songs loaded. In its current state, two loops from each song are playing.

both of these tasks for the user, using a source separation technique instead of equalization (see Fig. 1). The interface allows a user to upload any song; it then processes the audio and returns a set of loops. The user can then play with the loops on the spot, re-combining the loops live. If the user uploads more songs, they can also juxtapose loops from different songs, which are automatically tempo-matched, to create live mash-ups. Finally, users can download the loops to remix offline.

The website was inspired in part by Adventure Machine<sup>2</sup>, a Webby-nominated site designed to promote an album by the musician Madeon. That site allowed visitors to remix stem samples from a Madeon track, and it attracted significant traffic, according to the designers<sup>3</sup>. The thought that inspired us was: what if visitors could populate a remixing interface with samples from any track

<sup>2</sup> <https://www.madeon.fr/adventuremachine>

<sup>3</sup> <https://developers.google.com/web/showcase/2015/adventuremachine>

they own? With Unmixer, we aim to achieve this vision.

In the next subsection, we discuss alternative methods of extracting loops. In Section 2, we explain the features of the interface and some design considerations. In Section 3, we explain the algorithm which supports it [22] and propose an extension and improvement to it. Both contributions are evaluated. In Section 4, we present a usability study, and we end with a discussion (Section 5).

### 1.1 Tools for extracting loops and creating remixes

Existing interfaces for creating live remixes from a library of samples include AdventureMachine and Beat-Sync-Mash-Coder [7], but these do not allow you to populate the interface with automatically extracted loops. The web application *Girl Talk in a Box*<sup>4</sup> cuts any song into chunks for a user and offers novel resequencing options, but does not separate sources or allow the users to play multiple chunks at once. Advanced users can always use a Digital Audio Workstation (DAW), which is the most powerful and flexible way to compose a remix, but they will need to do the work of extracting loops on their own. That said, software exists to support this tedious task, such as [3] and [18], both of which require users to guide the algorithm by indicating regions of the spectrum to ignore or focus on. In sum, we are not aware of another interface that, given an input song, extracts source-separated loops and presents them to the user for remixing.

To extract repeating patterns, two broad approaches are popular: kernel-additive modeling [10], including harmonic-percussive source separation (HPSS) [4] and REPET, a method of foreground-background separation that models a looping background, and its variants [17]. However, these are binary separations: only two sources are obtained. The family of non-negative matrix factorization (NMF) approaches includes NMF [21], NMF deconvolution [19], and non-negative tensor factorization (NTF) [5]. All take advantage of redundancies in the signal—repeating spectral templates, transpositions, stereo dependence, etc. By applying NMF iteratively, [20] separated layers of electronic music that built up progressively, but did not model the extracted sources in terms of loops. This is a drawback shared by all of these algorithms: they produce full-length separated tracks. To obtain short, isolated loops, some extra step is required. However, a recent NTF system [22] models the periodic dependencies in the signal and is thus suitable for extracting loops directly; it forms the basis for our system and is described in Section 3. An orthogonal approach to extracting loops is that of [12], which seeks only to extract drum breaks (short drum solos that are desirable for remixes) by devising a percussion-only classifier.

## 2. WEB SERVICE AND INTERFACE

The purpose of the interface is to allow users to remix and mash-up the songs they love, and to perform automatically

<sup>4</sup><http://girltalkinabox.playlistmachinery.com>

the work of isolating loops, normally done through editing and equalization or source separation.

The interface has just a few, simple features, all visible in the screenshot (see Fig. 1). To begin, a user must upload a song from their hard drive, using the box at the bottom of the page. They may adjust the number of loops to extract using the drop-down list, with possible values between 3 and 10. The audio is processed on the web server using the algorithm outlined in Section 3; once the audio has been analyzed, the interface is populated with a set of loop ‘tiles’, each tile bearing a waveform sketch. The possible actions are then:

1. click on loop tiles to activate or deactivate them;
2. change the global tempo (drop-down list at top-right of the interface);
3. pause playback (button at top-left);
4. download a zip archive containing all the loops for a given song;
5. choose an additional song (and number of loops to extract) using the box at the bottom of the page.

Uploading and processing a new file can take a while (currently between 5 and 10 minutes), but users can still use the other functions (playing tiles and changing the tempo) while waiting for the next batch of tiles. Users can add any number of songs to a single workspace.

All the loops have the same duration (equivalent to two bars of audio), and the playback of the loops is synchronized so that the downbeats align, no matter when the user activates them. When a user uploads the first song, the global tempo is set to the detected tempo. New songs added to the workspace are tempo-shifted to match the global tempo.

The interface was built as a web application using React<sup>5</sup>, making it available on any web-accessible device through a browser. To handle audio playback, we use the Web Audio API, making it easy to synchronize playback of all the tiles and control the playback rate. The audio analysis is run on our server, using the algorithm described in the next section. The web server keeps a log of uploaded audio files and the analysis computed for each. To save time, if the system recognizes an uploaded audio file, it re-uses the old analysis. However, it does not re-use the old audio; it re-extracts the component loops from the new audio to return to the user, thereby avoiding any copyright issues related to redistributing audio.

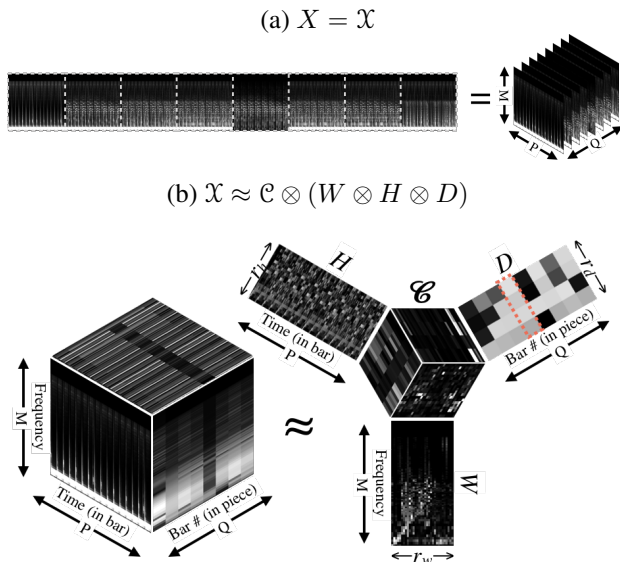
## 3. LOOP EXTRACTION

Our approach to extracting loops is based on [22], but we make two contributions: first, we suggest and evaluate methods for selecting individual loops; second, we propose and evaluate an extra “core purification” step.

### 3.1 Review of NTF for source separation

The pipeline for our system, based on the work of [22], is shown in Fig. 2. First, the mono spectrogram  $X$  is di-

<sup>5</sup><https://reactjs.org/>



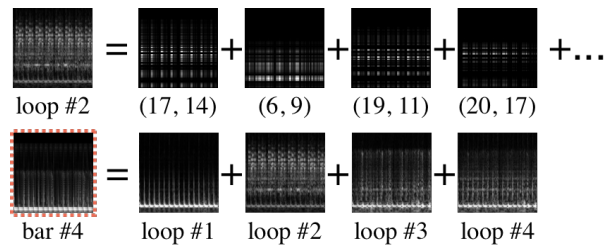
**Figure 2.** Overview of system, using example ‘125\_acid’ [11]. (a) Spectrum of 8-bar song reshaped into tensor. (b) Tucker decomposition expresses song as product of frequency templates  $W$ , rhythm templates  $H$ , repetition templates  $D$ , and a core tensor  $\mathcal{C}$ . In this example,  $(M, P, Q) = (1025, 661, 8)$  and  $(r_w, r_h, r_d) = (32, 20, 4)$ .

vided into downbeat-sized windows using a beat-tracker. (Unmixer uses the madmom system [1], but the illustrated example uses the known downbeats.) The  $M \times P$ -shaped spectrogram windows (one for each bar) are stacked into a new dimension, creating an  $M \times P \times Q$  tensor  $X$  (Fig. 2a). Then, using Tensorly [9], we compute the non-negative Tucker decomposition (NTD) with ranks  $(r_w, r_h, r_d)$ , approximating the tensor as the outer product  $X \approx \mathcal{C} \otimes (W \otimes H \otimes D)$  (Fig. 2b). In plain terms, the NTD models the spectrum with three meaningful components: a set of spectral templates  $W$  (the *sounds*), a set of within-bar time-activation templates  $H$  (the *rhythms*), and a set of loop-activation templates  $D$  (the *layout*, i.e., the arrangement of loops in the song). In Fig. 2b, a decomposition of an artificial 8-bar ( $W = 8$ ) stimulus [11] has been approximated using  $r_d = 4$  loop templates, giving a good estimate of the layout of the piece (shown in Fig. 6). The templates are diverse: some sounds are monophonic, others polyphonic; some rhythms are percussive, others sustained.

The sparse core tensor is  $\mathcal{C}$ ; a non-zero element  $\mathcal{C}_{[i,j,k]}$  indicates that sound  $i$  is played with rhythm  $j$ , and this pattern is repeated according to layout template  $k$ . To separate the contribution of the  $k^{\text{th}}$  loop, use the  $k^{\text{th}}$  row of  $D$  to take the outer product  $\mathcal{C} \otimes (W \otimes H \otimes D_k) \approx X_k$ , and unfold the tensor into  $X_k$ . The reconstructed real-valued spectrum  $X_k$  is not sufficient to recreate the signal; we must apply softmasking (as outlined in [17] and implemented in librosa [13]) and use the original phase:

$$y_k = \text{ISTFT}(\text{phase}(X) \cdot \text{mag}(X) \cdot \frac{X_k^p}{X_k^p + X^p}) \quad (1)$$

where  $p$  is the power of the softmask operation. As noted



**Figure 3.** Example reconstruction of spectrum for a single loop (#2) from multiple (freq, rhythm) combinations (above), and for a single bar (#4, also indicated in Fig. 2) from multiple loops (below).

by [6], using softmask filters is convenient, although not necessarily optimal, for non-negative approaches like ours.

The core tensor can be interpreted as a set of ‘‘recipes’’ for building the loops, the recipe for the  $k^{\text{th}}$  loop being the  $r_w \times r_h$  slice of the core tensor  $\mathcal{C}_{[:, :, k]}$ . To see how, note that  $W_i \otimes H_j$ , the outer product of the  $i^{\text{th}}$  sound with the  $j^{\text{th}}$  rhythm, is an  $M \times P$  spectrum of a single bar. The outer product  $\mathcal{C}_{[:, :, k]} \otimes (W \otimes H)$  thus represents a sum of such one-bar spectra, leading to the  $k^{\text{th}}$  loop template. Fig. 3 shows how the  $2^{\text{nd}}$  loop template is the sum (in descending order of magnitude) of individual  $W_i \otimes H_j$  components. Each bar in the piece will be a superposition of several loops: the bottom part of the figure shows how the  $4^{\text{th}}$  bar consists of copies of each loop.

### 3.2 Loop selection

The output of the algorithm in [22] is a set of full-length tracks, each corresponding to the contribution of one loop. For remixing purposes, we only want a bar-length version of each loop, as cleanly separated as possible. The plain approach is to extract the full-length track, then excerpt a single bar, but the question is then: which bar to select?

There are at least two factors to consider: how loud a given loop instance is, and how strongly that instance stands out from the other parts. Loudness can be maximized by choosing the bar that takes the maximum value in the loop activation matrix: i.e., to select the best bar for the  $k^{\text{th}}$  loop, choose  $\text{argmax}(D_{[k, :]})$ . To minimize cross-talk, we consider two approaches. First, we may normalize the columns of  $D$  by choosing  $\text{argmax}(D_{[k, :]} - \bar{D})$ , where  $\bar{D}$  is a vector of the column means of  $D$ .

Alternatively, the coefficients from the softmask operation could estimate how prominently a given loop stands out from the background. In this approach, we compute the softmask coefficients for each bar (i.e., the fractional part of Equation 1), and then select the bar which maximizes the total value of the mask. That is, if  $M_{k,i}$  gives the softmask matrix for the  $i^{\text{th}}$  bar for the  $k^{\text{th}}$  loop, we select  $\text{argmax}(\sum M_{k, :})$ . Finally, we may combine any or all of these decision criteria.

**Evaluation** To determine the most reliable selection method, we tested them on a set of stimuli assembled by [11]. The test set contains 7 compositions, each containing 4 loops arranged in the same 8-bar layout (shown

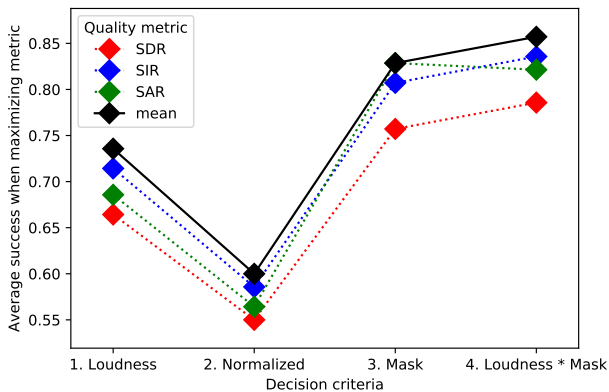


Figure 4. Main effect of bar selection strategies.

in the bottom part of Fig. 6). We ran the NTD algorithm on all the stimuli (with  $(r_w, r_h, r_d) = (32, 40, 4)$ ), and then measured the reconstruction quality of each loop in each bar. We measured reconstruction quality using SDR, SIR and SAR, the source-to-distortion, -interference, and -artefact ratios, respectively, calculated using mir\_eval [16]. (For each metric, higher is better.) Then, we tested how frequently the optimal bars were selected by maximizing each criterion. The four tested criteria were: loudness only ( $D_{[k,:]}$ ); normalized loudness ( $D_{[k,:]} - \bar{D}$ ); mask only ( $M_{[k,:]}$ ); and loudness times mask ( $D_{[k,:]} \cdot M_{[k,:]}$ ).

The main effect of the choice of criterion is shown in Fig. 4, which shows how often a given strategy (e.g., “select the loudest bar”) correctly found the bar that maximized a given metric. Using loudness alone (1), the optimal bar was selected at least two-thirds of the time. Normalizing the matrix  $D$  to diminish cross-talk (2) was too coarse, with the best bar selected less than 60% of the time. However, multiplying the loudness by the mask (4) to diminish cross-talk led to the best overall result, with the optimal bar selected around 80% of the time. The choice of criterion depends slightly on the quality metric (SDR, SIR or SAR) being maximized: if the priority is to maximize SAR, using the mask alone (3) may be advised, but using the loudness-and-mask criterion worked best overall, and it is the criterion used in the Unmixer system.

### 3.3 Loop purification

As reported by [22], a problem with the algorithm is that the extracted loops can be redundant. For example, suppose a song contains a drum pattern  $A$  and a synth pattern  $B$ , which are independent, but where  $B$  never occurs without  $A$ . Instead of modeling the independent patterns  $A$  and  $B$ , the algorithm is likely to learn one pattern for  $A$  and another for  $A + B$ . This error is not fixed by changing the number of loop templates that the model should learn.

The problem would be avoided if the core tensor were estimated with a sparsity constraint. Note that sparsity is only desired in the  $3^{rd}$  dimension, to prevent the  $r_w \times r_h$  slices (the loop ‘recipes’) from being too similar. Denseness is still desirable in the other dimensions; indeed, allowing different sources (frequency templates) to share

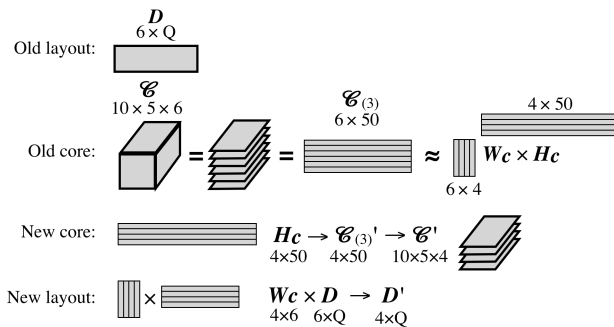


Figure 5. Illustration of core tensor purification.  $W_C$  and  $H_C$  are estimated using NMF with  $H_C$  constrained to be sparse.

rhythms (time activation templates), and vice versa, was a motivation to use NTD to begin with, since it allows an accurate and meaningful reconstruction without the model size becoming infeasibly large.

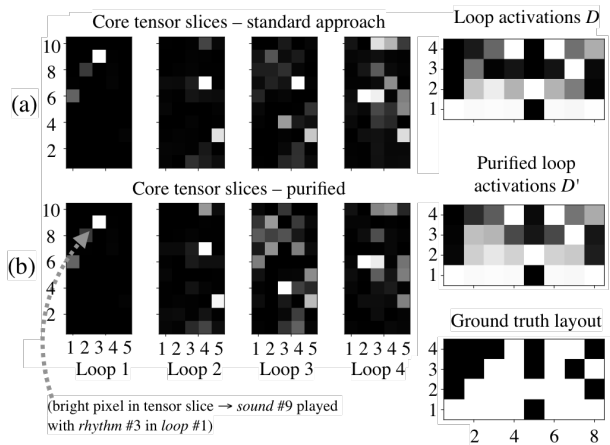
Algorithms for sparsity-constrained tensor factorizations exist [14], but we are not aware of any existing tensor decomposition packages that implement them.<sup>6</sup> Therefore, we propose to follow regular NTD with a “core-purification” step, using sparsity-constrained NMF to simplify the core tensor.

The process is illustrated in Fig. 5, supposing a decomposition of initial rank  $(10, 5, 6)$ . First, we take  $\mathcal{C}_{(3)}$ , the third-dimension unfolding of the core tensor  $\mathcal{C}$ , so that each horizontal slice of  $\mathcal{C}$  is reshaped into a row in  $\mathcal{C}_{(3)}$ . Then, we apply sparse NMF (using the SNMF function in Nimfa [23]) to model  $\mathcal{C}_{(3)} \approx W_C \times H_C$ , imposing sparsity on  $H_C$  only. (It is also possible to use simple NMF, without the sparsity constraint; this is tested later as a baseline.) The rows of  $H_C$  give a new set of maximally independent vectors; when the matrix  $H_C$  is refolded, it gives  $\mathcal{C}'$ , a set of maximally independent recipes. The matrix  $W_C$  tells us in what proportion to add the previous 6 templates to each other to obtain the new set of 4 templates; hence, we use it to transform the original layout  $D$  into  $W_C \times D = D'$ .

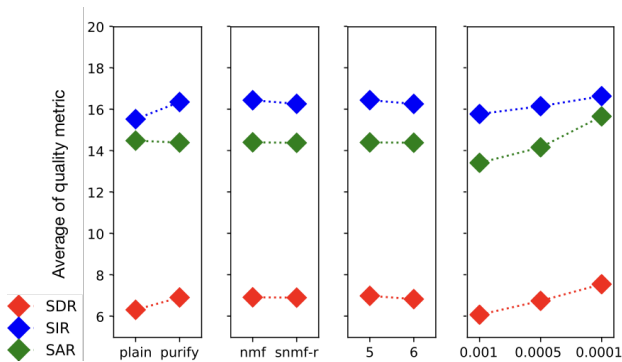
An example showing the promise of this method is shown in Fig. 6. First, we computed the NTD of a song using ranks  $(10, 5, 4)$ ; second, we computed a new NTD with ranks  $(10, 5, 6)$ , and purified the core so that the final shape was  $(10, 5, 4)$ . The 4 slices of each core are shown in parts (a) and (b) of the figure, along with the corresponding layouts: first, the loop activation matrix  $D$ ; second, the revised matrix  $D'$ . Each is an estimate of the ground truth layout at bottom. Whereas there is a clear redundancy among loops 2 and 3 in the first set (plain NTD), the redundancy has been reduced in the second (NTD with purification). The reconstruction error for the second set is actually a little higher (0.186 compared with 0.185), but the separated audio will be of higher quality. In the actual example (where we used ranks  $(32, 40, 4)$  and  $(32, 40, 6)$ ), the mean SDR and SIR for the 4 estimated loops climbed from 1.98 and 9.96 to 6.85 and 16.16, respectively, while

<sup>6</sup> NB: factorization of ‘sparse’ tensors is often a supported feature, but not the estimation of sparse outputs.





**Figure 6.** Comparison of (a) core estimated with plain NTD vs. (b) NTD with purification (core purified from 6 to 4), using stimulus ‘125\_acid’ [11]. The pixel at (4, 7) which appears redundantly in loops 2–4 in (a) has been subdued in (b). Each subfigure’s pixels are linearly scaled between its min and max.



**Figure 7.** Main effect of loop purification and runtime strategies.

SAR only decreased from 17.97 to 17.38. (These metrics are explained in the next section.)

**Evaluation** We evaluated the effectiveness of the proposed purification approach on the same dataset used in Section 3.2. We ran a fully-factorial design, exploring the following parameters and settings:

1. Plain NTD vs. purification;
2. Purification method: unconstrained NMF vs. the proposed SNMF approach;
3. Initial rank: 5 or 6 for purification methods only;
4. Tolerance: stopping criterion for computation of initial tensor (0.001, 0.0005 or 0.0001).

We use the known downbeat locations, and the final rank was always set to 4, the true number of loops in the examples. Although these must be estimated in a real-life scenario, we can still evaluate the impact of the purification method, initial rank, and tolerance. The reported metrics are SDR, SIR and SAR, as before.

Fig. 7 shows the main effects. We see that the purification step increased SDR and SIR, with only a minor de-

crease in SAR. This was the hoped-for result: less interference among the extracted signals (SIR), even if some reconstruction quality is sacrificed (more artefacts, SAR). However, we found that there was little difference between the proposed SNMF-R approach and a simpler NMF approach. We also see that purifying an estimate to rank 4 from rank 5 gave slightly better results than from rank 6. To put these effects in context, we also show the effect of reducing the tolerance, i.e., allowing the tensor factorization to run for longer. The purification step increased SIR nearly as much as did reducing the tolerance by 90%.

#### 4. USER EVALUATIONS

To assess the usability of the system and solicit feedback about the audio quality and enjoyability of app, we conducted a user evaluation. We solicited 8 participants (4 men and 4 women), all between 25 and 40 years old. The study had three components:

1. A background questionnaire covering: their musical training (using the standard Goldsmiths MSI short test [15]); their experience with audio editing interfaces and audio production; and their familiarity with the songs used in the study.
2. A 10-minute test of the interface. Participants were asked to upload a song, familiarize themselves with the interface, then upload 2–5 more and explore the combinations of sounds.
3. A feedback questionnaire with Likert-scale and free-text questions focused on usability, audio quality, enjoyment, and potential new features. Usability questions were adapted from the standard Systems Usability Scale [2]).

We limited participants to a set of 11 audio files that the system had already seen, which greatly sped up the processing time: users did not have to wait the typical 5–10 minutes for the Tucker decomposition to converge; they only needed to wait for the system to receive the audio, use the pre-existing decomposition to extract new audio files, and load said audio files into the interface, all of which takes roughly 30 seconds per file. Otherwise, the system they used is the same that is available live, now at `unmixer.ongaaccel.jp`. The ranks of the analysis are  $(r_w, r_h, r_d) = (50, 40, 30)$  (with  $r_d$  purified to 25 using SNMF-R), and the NTD is solved with tolerance 0.0001.

**User background:** The 8 user testers included musical experts and novices: 5 played musical instruments, and 6 had experience editing audio files. Of those 6, 4 also had some experience either using a DAW or creating a remix or mashup. According to the MSI test, the musical sophistication of 4 users was within a standard deviation of average [15], with 1 user above this range and 3 below.

**Usability:** There was unanimous agreement that the system was “easy to use” and that users thought “most people would learn to use this system very quickly.” In fact, in the free response, ease of use was cited by 7 users as one of the best things about Unmixer, especially for “musical novices” or “a beginner [like] myself”. One aspect of the

interface that users found inconvenient, though, was the inability to anticipate what a loop would sound like. One wrote that “you need to guess what’s in each loop based on the waveform and sometimes it’s not what you expected or wanted”, and suggested a short text label (e.g., ‘vox’, ‘drum’, ‘synth’) to indicate the content; another suggested visual hints. One user also noted that they did not know what the impact on the loop content or quality would be if they changed the number of loops to extract.

**Sound quality:** Asked whether “the sound quality of the loops was poor”, users were divided, with 4 each agreeing and disagreeing. However, one disagreeer later explained that while “audio quality of some samples was not great, ...it was possible to find good quality ones.” The quality of source separation was appreciated: 6 agreed that “most loops isolated a single source (e.g., drums, vocals, synth, bass)”, and 5 agreed that “loops within one song had a nice variety” (with 2 disagreeing). No one agreed that the “loops from different songs were too similar”.

**Enjoyment:** 6 agreed that “the combination of loops was often interesting”, and 4 agreed that they would “like to use Unmixer frequently” (with 2 disagreeing). The interface struck users as novel: none agreed that the “interface was similar to others I’ve used before”. Four agreed that “a remix artist could build a good song from these loops” (with 2 disagreeing); however, among users with experience using DAWs or creating mashups, opinion on this was split 2-against-2. Users saw different reasons to enjoy the interface: one “enjoyed the experience of doing something new”; another wrote that “it was very easy to try out new ideas very quickly”. Opinion also varied on what users would use Unmixer for, with responses including: generating mashups “for interludes or as backing music”; “using it to prototype [remix] ideas quickly”; “having fun at a house party—home-DJ style”; creating “nice effects for a video”; and “taking some of my own music” and generating “new ideas from it.”

**Potential features:** To understand what future developments for the interface would be most desirable, we polled users’ opinions on a list of 7 suggested features: controls for (1) loudness, (2) pitch-shifting and (3) equalization (e.g., to boost the bass or mid-range); (4) keyboard shortcuts for activating loops; (5) allowing more than 10 loops; and colour-coding loops by (6) type (e.g. vocals, synth) or (7) tonality. Users expressed broad approval for all of these except for (5), although if loops were colour-coded, it might become more desirable to have more loops available. Almost all users indicated that they had already thought of feature (1). When asked what change in Unmixer would make it more usable or useful, two users suggested a timeline functionality so that repeating sequences of loops could be made; two also wanted to be able to ‘save’ or ‘download’ the combinations they had created.

## 5. DISCUSSION

Our user feedback confirmed for us the usability of the system: without supervision, all participants completed the steps of the user study. The positive comments assure us

that the app holds promise as a tool for exploring interesting remix possibilities. However, to be most useful and engaging, we need to improve the sound quality of the extracted audio. The interface as it is may even be *too* simple: we expect that with a few changes, like adding keyboard shortcuts and having the tiles give some visual hint as to their content, we can increase user enjoyment and satisfaction. On the other hand, we should not implement all of the features discussed; to do so would be to program an in-browser, fully-functional DAW, whereas our focus is to provide users with loops extracted from songs and allow them to experiment with combinations.

The user study differs from the live experience in a few ways: (1) It allowed a choice among 11 pre-selected tracks, although real users are free to choose any music from their library; and (2) it featured a streamlined experience with minimal waiting for the system to analyze the files. We must collect more feedback from realistic scenarios to understand the system usability. If we cannot speed up the algorithm, we may need to adjust the interface to maintain the feeling of interactivity. For instance, we could provide the user with a quick-and-dirty set of extracted loops, and refine them in the background while the user experiments. Also, the current version kept fixed the ranks of the analysis and the amount of purification. We hope to make these tuneable, or have the system predict the optimal values.

There remain several ways to improve the source separation quality. We have reconstructed the signals using softmask filtering, but we could realize further improvements, and even speed up the algorithm, by using newer masking methods, such as the divergence-based masks proposed by [6]. To improve the system’s output, we could use equalization to refine the separated loops. As noted in the introduction, creators of unofficial remixes often use equalization to separate sources in lieu of source separation algorithms. We could blindly apply equalization or HPSS to create, say, bass, treble and percussive versions of each loop, or use a set of instrumentation-detection functions (similar to [12]) to select the best ones.

To evaluate these proposed improvements, it is important to collect more expansive test sets. The small test set of [11] was sufficient to assess the best bar-picking strategy and whether the purification step was useful. However, future work on this task ought to treat more diverse stimuli—more genres, loops per song, and loop layouts—to gauge how the system copes with realistically complex pieces.

## 6. CONCLUSION

We developed Unmixer, a web-app where users can upload music, extract loops, remix them, and mash-up loops from different songs. Expert and novice users found it easy to use; many also found it a novel way to develop remix ideas, although higher-quality audio output may be required for polished remixes. The backend uses the NTD-based source separation algorithm from [22]; we proposed and tested techniques to select the best reconstructed loop excerpts, and techniques to refine the loop layout, measurably improving on the baseline system output.

## 7. ACKNOWLEDGMENTS

This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

## 8. REFERENCES

- [1] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. madmom: a new Python Audio and Music Signal Processing Library. In *Proc. of the ACM International Conference on Multimedia*, pages 1174–1178, Amsterdam, The Netherlands, November 2016.
- [2] John Brooke. SUS: A ‘quick and dirty’ usability scale. In Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor and Francis, London, UK, 1996.
- [3] Nicholas J Bryan, Gautham J Mysore, and Ge Wang. ISSE: An interactive source separation editor. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–266. ACM, 2014.
- [4] Derry FitzGerald. Harmonic/percussive separation using median filtering and amplitude discrimination. In *Proc. of the International Conference on Digital Audio Effects*, Graz, Austria, September 2010.
- [5] Derry FitzGerald, Matt Cranitch, and Eugene Coyle. Sound source separation using shifted non-negative tensor factorisation. In *Proc. of the IEEE ICASSP*, volume 5, Toulouse, France, 2006.
- [6] Derry Fitzgerald and Rajesh Jaiswal. On the use of masking filters in sound source separation. In *Proc. of the International Conference on Digital Audio Effects*, York, UK, 2012. Dublin Institute of Technology.
- [7] Garth Griffin, Youngmoo E. Kim, and Douglas Turnbull. Beat-sync-mash-coder: A web application for real-time creation of beat-synchronous music mashups. In *Proc. of the IEEE ICASSP*, pages 437–440, Dallas, TX, USA, 2010.
- [8] Sheena D Hyndman. No money, mo’ problems: The role of the remix in restructuring compensation for producers of electronic dance music. *MUSICultures*, 41(1):57–72, 2014.
- [9] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *The Journal of Machine Learning Research*, 20(1):925–930, 2019.
- [10] Antoine Liutkus, Derry Fitzgerald, Zafar Rafii, Bryan Pardo, and Laurent Daudet. Kernel additive models for source separation. *IEEE Trans. on Signal Processing*, 62(16):4298–4310, 2014.
- [11] Patricio López-Serrano, Christian Dittmar, Jonathan Driedger, and Meinard Müller. Towards modeling and decomposing loop-based electronic music. In *Proc. of the ISMIR*, pages 502–508, New York, NY, USA, 2016.
- [12] Patricio López-Serrano, Christian Dittmar, and Meinard Müller. Finding drum breaks in digital music recordings. In *Proc. of the International Symposium on Computer Music Multidisciplinary Research*, pages 68–79, Matosinhos, Portugal, 2017.
- [13] Brian McFee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proc. of the Python in Science Conference*, 2015.
- [14] Morten Mørup, Lars Kai Hansen, and Sidse M Arnfred. Algorithms for sparse nonnegative tucker decompositions. *Neural Computation*, 20(8):2112–2131, 2008.
- [15] Daniel Müllensiefen, Bruno Gingras, Jason Jiří Musil, and Lauren Stewart. The musicality of non-musicians: An index for assessing musical sophistication in the general population. *PLOS One*, 9(2), 2014.
- [16] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel PW Ellis. mir\_eval: A transparent implementation of common MIR metrics. In *Proc. of the ISMIR*, pages 367–372, Curitiba, Brazil, 2014. Citeseer.
- [17] Zafar Rafii, Antoine Liutkus, and Bryan Pardo. REPET for background/foreground separation in audio. In G. R. Naik and W. Wang, editors, *Blind Source Separation*, Signals and Communication Technology, pages 395–411. Springer-Verlag, 2014.
- [18] Zafar Rafii, Antoine Liutkus, and Bryan Pardo. A simple user interface system for recovering patterns repeating in time and frequency in mixtures of sounds. In *2015 IEEE ICASSP*, pages 271–275. IEEE, 2015.
- [19] Mikkel N. Schmidt and Morten Mørup. Nonnegative matrix factor 2-d deconvolution for blind single channel source separation. In *International Conference on Independent Component Analysis and Signal Separation*, pages 700–707. Springer, 2006.
- [20] Prem Seetharaman and Bryan Pardo. Simultaneous separation and segmentation in layered music. In *Proc. of the ISMIR*, pages 495–501, New York, NY, USA, 2016.
- [21] Paris Smaragdis. Non-negative matrix factor deconvolution: Extraction of multiple sound sources from monophonic inputs. In *Independent Component Analysis and Blind Signal Separation*, volume 3195 of *Lecture Notes in Computer Science*, pages 494–499. Springer-Verlag, Berlin, Heidelberg, 2004.

- [22] Jordan B. L. Smith and Masataka Goto. Nonnegative tensor factorization for source separation of loops in audio. In *Proc. of the IEEE ICASSP*, pages 171–175, Calgary, AB, Canada, 2018.
- [23] Marinka Zitnik and Blaz Zupan. Nimfa: A python library for nonnegative matrix factorization. *Journal of Machine Learning Research*, 13:849–853, 2012.