Check for updates

SOFTWARE TOOL ARTICLE

Besra: An interactive tool for speech annotation, phonetic analysis, and computer-assisted fieldwork

[version 1; peer review: awaiting peer review]

Kellen Parker van Dam[1], David Snee[1], Johann-Mattis List[1], Andre Goderich[2], Luca Ciucci[1], Carlo Y Meloni[1], Keen Thaam[3], Nora Muheim[4], Katja Bocklage[1], Arne Rubehn[1]

[1]Chair for Multilingual Computational Linguistics, University of Passau, Passau, Bavaria, Germany
[2]Department of English, National Changhua University of Education, Changhua City, Taiwan
[3]Department of Linguistics, Nagaland University, Meriema, Nagaland, India
[4]University of Helsinki, Helsinki, Uusimaa, Finland

v1 **First published:** 12 Oct 2026, **6**:420
https://doi.org/10.12688/openreseurope.25173.1
Latest published: 12 Oct 2026, **6**:420
https://doi.org/10.12688/openreseurope.25173.1

Abstract

Time-aligned speech annotation is typically done in Praat and ELAN, tools developed prior to the current Python-based NLP ecosystem. Taking advantage of moden NLP workflows means exporting files and reimporting the results. We address this with Besra, an open source desktop application for editing and analysing time-aligned annotation, written entirely in Python as a headless core with a cross-platform PySide6 interface. It is extensible, as Plugins are ordinary Python files executed in the host interpreter that receive the running application as an argument. The extension surface is every distribution installed in the environment, including the machine learning stack and the standards infrastructure of comparative linguistics. Waveform, spectrogram, pitch, formant, amplitude, and annotation displays share one time axis and one selection. It offers with a wider choice of estimation methods than annotation editors usually offer. Real-time quality measures are drawn on the same as the annotation is made, including a per-interval score for suitability as speech recognition training data, and a heatmap flagging transcriptions unlike the rest of their tier. Both are computed from the recording and its own annotation, so they need no lexicon or pretrained model and work on undescribed languages. Besra reads and writes Praat TextGrid, ELAN.eaf, and a documented version-controllable tab-separated format, with round-trip fidelity under test, so a project can adopt it for a single task. Besra is released under the GNU Affero General Public License version 3, developed on Codeberg and archived on Zenodo.

Open Peer Review

Approval Status *AWAITING PEER REVIEW*

Any reports and responses or comments on the article can be found at the end of the article.

Plain Language summary
Researchers who work on spoken languages spend much of their time marking up recordings: dividing them into utterances, words, and sounds, and writing down what was said and when. The two programs almost everyone uses for this, Praat and ELAN, were designed before the current generation of speech technology, and adding new capabilities to them means learning a special-purpose scripting language or writing Java. Besra does the same job but is written entirely in Python, the language in which most speech and machine learning research is now written. Anything published as a Python package, such as a speech recogniser, a statistical model, or a client for a linguistic database, can be used from inside Besra by writing a short file, with no exporting and reimporting of data. While the annotation is being made, Besra also shows whether a recording is clean enough to be useful for training speech software and whether a transcription has been written differently from the rest of the file, so that problems can be fixed while the speaker is still present. Because Besra reads and writes the file formats that Praat and ELAN use, it can be added to an existing project without disturbing anything else. It is free and open source and runs on macOS, Linux, and Windows.

Keywords
phonetics, language documentation, speech annotation, Python



This article is included in the Horizon Europe gateway.



This article is included in the European Research Council (ERC) gateway.

Corresponding author: Kellen Parker van Dam (kellenparker.vandam@uni-passau.de)

Author roles: van Dam KP: Conceptualization, Data Curation, Investigation, Project Administration, Software, Supervision, Validation, Writing – Original Draft Preparation, Writing – Review & Editing; **Snee D**: Conceptualization, Data Curation, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **List JM**: Conceptualization, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Goderich A**: Conceptualization, Software, Writing – Review & Editing; **Ciucci L**: Conceptualization, Data Curation, Software, Writing – Review & Editing; **Meloni CY**: Software, Writing – Review & Editing; **Thaam K**: Data Curation, Investigation, Validation, Writing – Review & Editing; **Muheim N**: Conceptualization, Data Curation, Writing – Review & Editing; **Bocklage K**: Writing – Review & Editing; **Rubehn A**: Software, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (Grant agreement No. 101044282). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency, nor any other funding agencies involved. Neither the European Union nor the granting authority can be held responsible for them.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2026 van Dam KP *et al*. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: van Dam KP, Snee D, List JM *et al.* **Besra: An interactive tool for speech annotation, phonetic analysis, and computer-assisted fieldwork [version 1; peer review: awaiting peer review]** Open Research Europe 2026, **6**:420 https://doi.org/10.12688/openreseurope.25173.1

First published: 12 Oct 2026, **6**:420 https://doi.org/10.12688/openreseurope.25173.1

Introduction

Time-aligned annotation of speech is a shared substrate for phonetics, phonology, language documentation, sociolinguistics, and speech technology. It is also slow work. An annotator begins from audio, and at times video, recordings of native speakers, divides the signal into utterances, words, and segments, and aligns a phonetic transcription to it. Further tiers are then laid over that alignment to describe what the transcribed forms do, through interlinear glossing, morphological and syntactic analysis, and the marking of prosody. The result is the basis for grammatical description, dictionary compilation, cross-linguistic comparison, and language preservation.

Computational speech analysis has advanced quickly of late, largely on the back of the Transformer architecture (Vaswani et al., 2017), but the tools in which annotation is actually made were built before that shift, and combining the two means moving files between them. A framework is therefore needed that offers the functionality of an annotation interface together with direct access to automatic speech analysis and to computer-assisted cross-linguistic comparison. Besra is a first attempt at such a tool.

Speech annotation tools

For more than two decades, two tools dominate the field: Praat (Boersma and Weenink, 2024) for acoustic analysis and phonetic segmentation, and ELAN (Wittenburg et al., 2006) for multi-tier annotation of longer recordings.[1] Both are mature, well documented, and deeply embedded in researcher training and established workflows, and neither was designed to sit inside the Python ecosystem, in which contemporary speech and language processing is written.

Other frameworks address related needs at corpus level: SPPAS (Bigi, 2015) offers automatic annotation pipelines across many languages, EMU-SDMS (Winkelmann et al., 2017) unifies database management, query, and signal analysis around R, and PhonBank and Phon (Rose and MacWhinney, 2014) target phonological development corpora. These are efficient in batch processing and corpus query, whereas Besra focuses on the annotation phase: a model is run on the interval under the cursor and its output is immediately editable. Besra recognises Praat's tier model and ELAN's tier dependency system directly, so it can be inserted into an existing workflow instead of replacing it.

Automatic speech analysis

Bridging an annotation tool and a model produces a familiar loop. An annotator in Praat who wants to apply a self-supervised phone recogniser (Baevski et al., 2020; Conneau et al., 2021; Li et al., 2020) exports a TextGrid, writes a script that reads it, slices the audio, runs the model, writes a new TextGrid, and reopens it. The serialisation, path management, and format inconsistencies that accumulate on every iteration cost time, and in low-resource settings this overhead discourages the approach that makes model assistance worthwhile: running a model on one interval, correcting the result, and retraining on the correction (Michaud et al., 2018; Foley et al., 2018). The same applies in phonetics, where analysis may rely on pitch listings from Praat but needs further scripting for normalisation and visualisation (van Dam, 2024).

Parselmouth (Jadoul et al., 2018) exposes Praat's analysis routines to Python and is used inside Besra for the Praat-equivalent methods; the two connect Praat and Python in opposite directions, since Parselmouth makes Praat's algorithms available within Python scripts, whereas Besra makes the Python ecosystem available within the annotation environment. Elpis (Foley et al., 2018), Persephone (Adams et al., 2018; Michaud et al., 2018), and forced aligners such as the Montreal Forced Aligner (McAuliffe et al., 2017) show that even modest automatic transcription accelerates documentation work when supervised by a human; Besra is complementary, since it is the editor in which their output is corrected and it can host the same models directly.

Cross-linguistic data integration

Because Besra is written in Python, it sits directly on the standards infrastructure of comparative linguistics instead of reaching it through an export. CLDF (Forkel et al., 2018), CLTS (List et al., 2021), and the associated libraries are ordinary dependencies, importable from any plugin, so transcriptions can be normalised against a controlled transcription system and validated against a CLDF dataset as they are entered, which is when inconsistency is easiest to correct. Annotations can also leave Besra as a wordlist and be imported directly into EDICTOR (List et al., 2025), so that forms transcribed from audio reach a cognate coding environment without an intermediate reformatting step.

[1] FieldWorks Language Explorer (SIL International, 2025) is at least as widely used in language documentation, but its focus is more often dictionary building and text interlinearisation than acoustic signal annotation, and time alignment is therefore typically handled by exchanging material with ELAN or Praat (Bodt, 2022; Skilton et al., 2025).

Besra

Besra covers each step of such a workflow in one system. The application is written entirely in Python with a PySide6 (The Qt Company, 2025) interface for cross-platform cohesion. Plugins and user scripts are ordinary Python files that receive the running application as an argument, giving them direct access to the audio, the annotation tiers, the computed analysis tracks, the selection, the plotting figure, and the application window. There is no inter-process protocol and no export step.

The design claim. Because plugins are executed inside the host interpreter, the set of operations a user can perform while annotating corresponds to the Python distributions installed in the environment. Praat scripts rely on a self-contained imperative language with no package ecosystem, and ELAN extensions are written in Java against ELAN's own interfaces, so in both cases the reachable functionality is limited by the choices of the respective authors. Besra's extension surface is limited only by the Python Package Index (Python Software Foundation, 2026), so a capability need not have been anticipated by the maintainer (The extension surface).

Quality assurance in situ. Corpus quality control is normally a batch activity, run over a finished collection once the recordings are made and the annotation is complete. Because Besra's analysis displays share the annotation's environment, quality measures are instead rendered as lanes under the annotation and read during the session that produces it (Quality assurance displays).

Methods

Implementation

Architecture, formats, and responsiveness. Besra separates a headless core from the interface. The core module implements the annotation data model, the file readers and writers, and the signal analysis functions, resting on NumPy (Harris et al., 2020), SciPy (Virtanen et al., 2020), and librosa (McFee et al., 2015), and can be imported without a display; the interface module builds the Qt application on top of it, and plugins may import either. The data model mirrors Praat's: a `TextGrid` holds an ordered set of tiers, each either an `IntervalTier` of contiguous, non-overlapping intervals or a `PointTier` of points. Three formats are read and written into that representation: Praat TextGrid, ELAN `.eaf`, and `.besra`, a documented five-column UTF-8 tab-separated file with a metadata block recording tier types and the ELAN hierarchy. Audio and video are opened in any format FFmpeg (FFmpeg Developers, 2025) can decode, and since field recordings are often hours long, Besra derives a downsampled copy at 8 kHz for the whole-file overview while detailed analysis runs against the full-rate signal for the visible window in a background thread. The extended data give all three formats in full.

Analysis displays. Waveform, spectrogram, pitch, formant, amplitude, quality, n-gram entropy, and annotation displays all share one time axis and one selection, as shown in Figure 1; the extended data list the estimation methods available for each. Pitch and formant estimation offer a wider choice than annotation editors typically provide: the familiar Praat methods are included through Parselmouth while others are implemented on librosa or in Besra itself, formant tracking with KARMA (Mehta et al., 2012) offers per-frame uncertainty that the interface can overlay, and both track types may be post-processed with a Rauch–Tung–Striebel smoother (Rauch et al., 1965). That table is what Besra provides, not the limit of what it can draw: any measure computable in Python can be drawn on the same axis (The extension surface), so an estimator can be chosen on methodological grounds. The epoch-based REAPER (Talkin, 2015), among the most reliable available for voicing detection and F0 tracking (Vaysse et al., 2022), is not implemented in Praat; in Besra it is used like any other pitch method.

The extension surface. A plugin is a single Python file containing one top-level function named after it. At start-up, Besra scans the user's plugin directory, constructs a module specification for each file with `spec_from_file_location` from `importlib.util`, and executes it in the running process; the named function is bound to a menu action and called with the live application window as its single argument.[2] Because the plugin module is executed by the interpreter that runs the application, its `import` statements resolve against that interpreter's `sys.path`, so any distribution installed in the environment Besra was installed into is importable immediately and without registration, the machine-learning stack of speech and language processing included. None of it required Besra-side work, and weight is opt-in: Besra runs without PyTorch or Transformers, and a plugin whose dependencies are absent fails with a readable message instead of preventing start-up. Through the injected application object a plugin reads the audio, the transcript, the analysis tracks, the selection, and the Matplotlib (Hunter, 2007) figure, and modifies the application through documented helpers;

[2] A plugin is therefore ordinary Python running with the user's privileges, and only trusted plugins should be installed; see Quality control and validation.

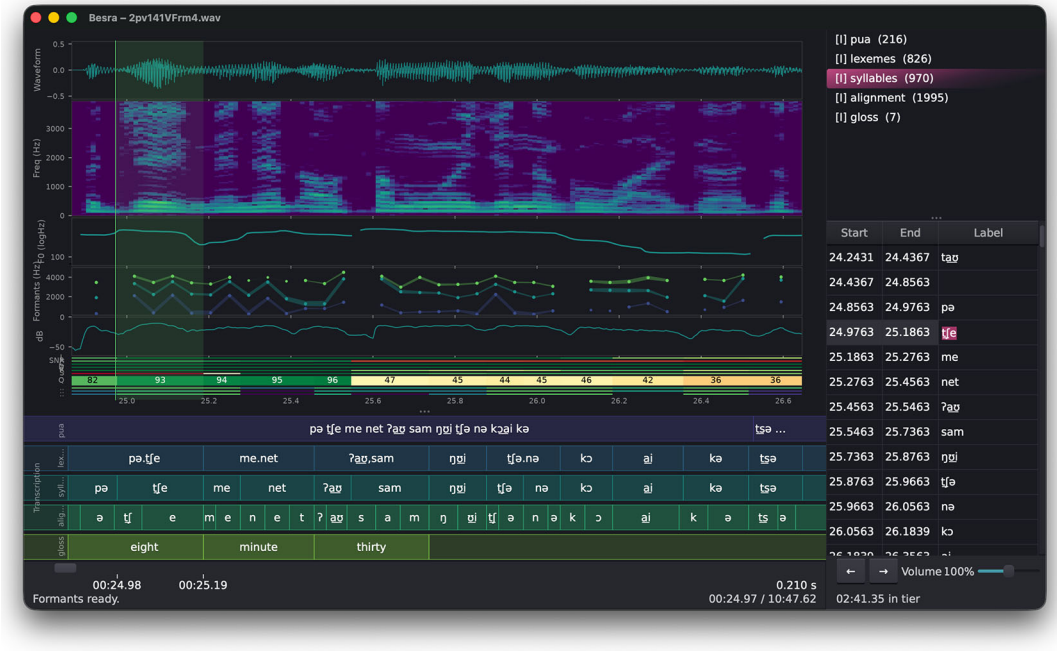


Figure 1. The Besra interface with a single recording open. Every display shares one time axis and one selection, so the quality assurance measures can be read as lanes beneath the annotation they describe, at the position where the problem occurs.

because the objects are live rather than serialised, a plugin that adds a tier sees it appear as the call returns. The extended data document the metadata header and the application object and show a complete plugin.

Writing generated annotation back. An interval tier in the Praat model tiles the timeline: intervals never overlap and never leave holes. Since people other than the maintainer will write generating plugins, Besra documents one contract that every such plugin follows, so that surrounding hand annotation is clipped against the span the output occupies and the file stays valid for Praat and ELAN. The extended data state it in full.

Quality assurance displays. Annotation is increasingly done with an eye to training data, yet two failure modes are invisible in a waveform: recordings unusable for model training, and transcriptions that are internally inconsistent. Besra renders both as displays on the shared time axis, so that a problem appears at its position in the recording while the annotator is looking at it. The first scores each annotated interval from 0 to 100 for suitability as ASR training material, combining signal-to-noise ratio, effective bandwidth, and reverberation, with clipping and level as gates; the estimates are annotation-aware, taking the noise floor from the gaps between annotations and the speech level from frames inside them. The second flags transcriptions unlike the rest of the same tier, implementing the phonotactic inconsistency detection method of van Dam and Stephen (2026), evaluated there on a Kokborok (kokb1239; Tibeto-Burman) wordlist: a character n-gram model built from the labels of the selected tier itself scores each interval, and the intervals are drawn as a heatmap along the timeline. Because the model comes from the tier being checked, it needs no external resource, lexicon, or pretrained language model, and works on initial data from undescribed languages. The extended data give both in detail.

Transcription systems and interface language. Phonetic transcription strains Unicode's model: a single grapheme often consists of several codepoints, so editing operations, string comparisons, and model tokenisers may disagree about segment boundaries (Moran and Cysouw, 2018). Besra therefore offers a Private Use Area encoding as a working representation, in which a plugin normalises an IPA tier with CLTS (List et al., 2021) and maps each multi-codepoint grapheme to a single codepoint on a parallel tier, rendered with a bundled font. One segment is then one character, so cursor movement, deletion, length, syllabification, and n-gram models agree with the analyst's own notion of a segment; the cost of skipping this step is reported under Use cases. Transliteration between IPA and a practical orthography uses orthography profiles (Moran and Cysouw, 2018), and the interface is localised through JSON files identified by Glottolog code (Hammarström et al., 2024) (see the extended data).

Quality control and validation. The test suite contains just over three hundred tests at roughly 94% statement coverage, run with `pytest` under `tox` against Python 3.12, 3.13, and 3.14. Format handling is tested by round-tripping: a file is read, written, and re-read, and the result compared to the original, the property that matters for a tool inserted into someone else's Praat or ELAN workflow. Pitch and formant estimation are validated against Praat, since where Besra offers a method Praat also implements, that method is a Parselmouth wrapper around the corresponding Praat command (Jadoul et al., 2018). A command-line validator checks a plugin against the required structure, and Besra statically scans a plugin's syntax tree before running it. We are explicit in the documentation and in the code that this scan is a heuristic aid and *not* a security boundary: obfuscated code evades it, and an empty report means nothing suspicious was found, not that the file is safe. That is the unavoidable cost of executing plugins in the host interpreter, which is what puts the whole package index within reach. The extended data give the coverage and the validator's repairs.

Operation

System requirements. Besra runs on macOS (Apple silicon and Intel), Linux, and Windows, and requires Python 3.12 or later, being tested against 3.12, 3.13, and 3.14. There is no requirement beyond a 64-bit processor able to run those versions; analysis is single-threaded per window and no GPU is needed, although the recognition and training plugins will use one through PyTorch if present. Memory scales with the duration of the open recording and not with the corpus, since Besra opens one recording at a time and holds both the decoded signal and its 8 kHz overview in memory; the application is developed and used on stock consumer laptops with no configuration beyond the defaults. A graphical display is required for the interface and an audio device for playback; the core runs headless, and FFmpeg is required for video and for audio in formats other than WAV.

Installation and dependencies. Two installation routes are supported on every platform. Besra can be installed from source into a virtual environment with `pip`, the route that makes the plugin mechanism fully available, since extending the application means installing further distributions into that environment. Alternatively, a self-contained bundle is built with PyInstaller from the supplied specification file, producing a macOS `besra.app`, a Linux bundle, and a Windows `besra.exe` from one configuration. Sixteen core dependencies are declared, chiefly NumPy, SciPy, librosa, Matplotlib, praat-parselmouth, PySide6, scikit-learn, pyclts, and pycldf; the optional `asr` extra, required by the recognition and training plugins, adds PyTorch, Transformers, and Allosaurus. The extended data give the full list with the version constraints declared in `pyproject.toml`, the build routes, and the disk footprint.

Running the application. Besra opens one recording at a time. The user opens an audio or video file, optionally with an existing TextGrid, `.eaf`, or `.besra` annotation, and the waveform, spectrogram, analysis tracks, annotation tiers, and quality displays are drawn on the shared time axis of Figure 1. Navigation relies on scrolling and zooming over the whole-file overview; a selection can be played back, and the label of the interval under the cursor is edited in place. If the file carries video, it plays in a panel alongside the displays, in step with the audio. Analysis parameters, including the estimation methods detailed in the extended data, are set in the preferences and take effect on the next redraw. Plugins appear as menu entries grouped by their filename prefix and act on the current selection, tier, or file. Annotation is saved back to any of the three formats.

Use cases

We report two short annotation workflows and one extended deployment described in a manuscript currently under review and available on OpenReview (Anonymous, 2026).

Basic annotation example

This hypothetical workflow illustrates opening any audio recording with an utterance-level transcript; it does not refer to a specific recording or dataset. The example takes a single recording with a Praat TextGrid holding one interval tier of utterance-level transcription. Opening the pair draws the analysis tracks on the shared time axis, with the transcription tier beneath them and the per-interval quality lanes beneath that. The analysis parameters are the shipped defaults: pitch estimated with pYIN between a floor of 75 Hz and a ceiling of 400 Hz with Rauch–Tung–Striebel smoothing applied, four LPC formants below 5500 Hz, and a spectrogram computed over a 5 ms window at a 2 ms step with a 512-point FFT. Running the CLTS normalisation plugin over the transcription tier rewrites each label into a controlled transcription, and saving as `.besra` produces the tab-separated form described under Architecture, formats, and responsiveness. Each row is one interval: an identifier, the tier it belongs to, its begin and end time in seconds at full floating-point precision, and its transcription. Reopening the file in Praat returns the original TextGrid boundaries unchanged, the round-trip property tested in the suite described under Quality control and validation.

Extended annotation example

This hypothetical workflow illustrates editing a layered annotation; it does not refer to a specific recording or dataset. The example uses a layered annotation in which a phone tier is a child of a word tier which is in turn a child of an utterance tier. The input is an ELAN `.eaf` file with its adjacent `.pfsx` preference file. Besra resolves the referring annotations of the child tiers to time-aligned intervals on opening, so that all three are editable against the signal, and preserves the parent and child relation when the file is written as `.besra` or back as `.eaf` (see the extended data). Splitting a long utterance interval and redistributing its segmental transcription across the split leaves the hierarchy intact, and the file reopens in ELAN unchanged.

Complete case study

The most extensive deployment we can report is a study currently under review (Anonymous, 2026), in which models for automatic speech transcription and analysis were developed for an under-documented Tibeto-Burman language. That paper is in blind review, so we withhold its authors and the name of the language and cite it anonymously. The input at the outset was a single wordlist recording, from which an annotated multi-speaker broadcast corpus spanning three dialectal groups was created; all transcription, correction, quality-based data selection, and phone-level alignment were carried out in Besra over roughly two weeks of part-time work, with the quality display deciding which material was clean enough for training and the recognition and training plugins detailed in the extended data supplying the models. The study reports the annotate-retrain loop in continuous use: a phone recogniser was run over the interval under the cursor, its output was corrected in place, and the corrected material retrained the model in increments of two to three minutes of audio, without leaving the application. Over the deployment, the ratio of human correction time to audio duration fell from roughly six to one to below one to one. It also documents what it costs to skip the transcription-system step of Transcription systems and interface language, since a character-level CTC vocabulary double-charges a segment written as a base letter plus a modifier. The outputs are ordinary annotation files, written back through the contract of Writing generated annotation back, so they open unchanged in Praat and ELAN. The extended data report the deployment in more detail.

Evaluation is to date deployment-based. Besra has additionally been used for fieldwork and corpus analysis by the authors, some colleagues, and their students, and three further supervised deployments are in preparation. This does not amount to a controlled comparison against Praat or ELAN, or to any systematic study of user experience; a formal usability investigation would benefit future versions.

Discussion

Reuse potential

Besra as a quality-control instrument during data collection. The quality displays of Quality assurance displays are reusable outside the projects they were built for and are most useful when recordings are analysed in the field. Any project building a speech corpus with an eye to model training can use the per-interval score to decide what to re-record while this is still possible, and the surprisal display to catch transcription drift before it propagates. Both are computed from the recording and its own annotation, so neither depends on prior description of the language, and both can be imported from the core module into a batch script for whole-corpus triage.

Besra as an interoperable editor within existing workflows. Besra implements Praat's annotation model and reads and writes TextGrid and `.eaf`, so a corpus can be segmented in ELAN, corrected in Besra with a model in the loop, and analysed in Praat. The `.besra` format serves collaborators who have neither tool: it opens in a spreadsheet, loads in one line of `pandas`, and is diffable in version control, making annotation reviewable by the same means as code.

Besra as a tool for data that cannot leave the local machine. Everything runs locally. Models are downloaded once and executed on the user's own hardware, and no audio or annotation is transmitted anywhere. For endangered-language material, recordings under community access restrictions, and data governed by consent agreements that do not contemplate commercial cloud transcription, local execution is a precondition for using automatic methods at all, and such a project can still add whatever local method it needs.

Extension potential

The reuse case that matters most to the design is the code other people run inside Besra, by the mechanism of The extension surface. This inverts the usual maintenance relationship for research software: a user with a new method unknown to the authors need not file a feature request, fork the application, or wait for a release. Around forty plugins are distributed alongside the application and are best read as realisations of this design principle rather than as contributions in their own right; the extended data group them. The loop this affords is short: a recogniser runs on the interval under the cursor, its output is immediately editable, and the corrected output is the training data for the next model, which the training plugins can fit without leaving the application. The environment is also a sufficient specification of an analysis,

since plugin dependencies and each plugin's settings file are ordinary artefacts on disk (see the extended data). It holds the comparative-linguistics stack too, so a plugin can normalise a transcription with CLTS (List et al., 2021), check a form against a CLDF dataset (Forkel et al., 2018), or align with LingPy (List and Forkel, 2023) in the annotation window.

Limitations and open problems

Several limitations bear directly on reuse. Besra is at version 0.9 at the time of initial release, so bugs may exist. Only one recording is open at a time, so corpus-wide operations are handled by walking a folder rather than by batch commands. The static plugin scan is a heuristic and not a sandbox (Quality control and validation), so malicious code could still be run by the user.

Outlook

Besra is an annotation and analysis environment that belongs to the Python ecosystem, in which speech and language processing is now written. Because plugins are executed in the host interpreter, with no export step and no bespoke scripting language, the capability available while annotating is set by what the user has installed, and one environment holds both the machine learning stack and the standards infrastructure of comparative linguistics. Support for Praat and ELAN formats means none of this requires a project to commit to Besra for more than a single task. Work in progress addresses the limitations of Limitations and open problems, but since the extension surface is the package index and not a fixed API, we expect the most useful additions to come from users; the plugin specification, validator, and tier-writing contract exist to make that path short.

Author contributions

Kellen Parker van Dam: Conceptualization, Investigation, Methodology, Project Administration, Software, Writing – Original Draft Preparation, Writing – Review & Editing.

David Snee: Conceptualization, Data preparation, Software, Writing – Original Draft Preparation, Writing – Review & Editing.

Johann-Mattis List: Conceptualization, Software, Writing – Original Draft Preparation, Writing – Review & Editing.

Andre Goderich: Conceptualization, Software, Writing – Review & Editing.

Luca Ciucci: Conceptualization, Data preparation, Software, Writing – Review & Editing.

Carlo Y. Meloni: Software, Writing – Review & Editing.

Keen Thaam: Investigation, Validation, Data preparation, Writing – Review & Editing.

Nora Muheim: Conceptualization, Data preparation, Writing – Review & Editing.

Katja Bocklage: Writing – Review & Editing.

Arne Rubehn: Writing – Review & Editing.

Ethics and consent

Ethical approval and consent were not required.

Software and code availability

Software and source code available from: `https://codeberg.org/besra/besra`. The plugin collection is available from `https://codeberg.org/besra/plugins`, and the documentation is available from `https://codeberg.org/besra/documentation`.

Archived source code at time of publication: Besra (version 0.9), Zenodo. `https://doi.org/10.5281/zenodo.22680513` (van Dam, 2026).

License: GNU Affero General Public License v3.0 (AGPL-3.0). Support requests, bug reports, and contributions are handled through the issue trackers of the corresponding Codeberg repositories. The corresponding author may also be contacted directly at the address given above.

Data availability

Underlying data

Data sharing is not applicable to this article as no data were created or analysed in this research. The basic and extended annotation examples describe hypothetical workflows, not analyses of specific recordings or datasets. The deployment results summarise the separately cited study (Anonymous, 2026); no reanalysis of that study's data was undertaken for this article.

Extended data

Zenodo: Extended data for 'Besra: An Interactive Tool for Speech Annotation, Phonetic Analysis, and Computer-Assisted Fieldwork'. https://doi.org/10.5281/zenodo.22680513 (van Dam, 2026).

This includes implementation, operation, and use-case detail supporting the corresponding parts of the Methods and Use cases sections: file format specifications, the plugin extension model, the tier-writing contract, quality assurance methods in full, transcription system detail, validation coverage, system requirements and dependencies, the full table of analysis displays and estimation methods, the full table of distributed plugins, and supporting material for the use cases.

The extended documentation is available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

References

Adams O, Cohn T, Neubig G, *et al.*: **Evaluating phonemic transcription of low-resource tonal languages for language documentation.** *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC).* 2018; pages 3356–3365.

Anonymous: **Bootstrapping speech recognition for an under-documented Tibeto-Burman language.** 2026. Manuscript under review; authors and language name withheld to preserve blind review.

Baevski A, Zhou Y, Mohamed A, *et al.*: **wav2vec 2.0: A framework for self-supervised learning of speech representations.** *Advances in Neural Information Processing Systems.* 2020; **33**.

Bigi B: **SPPAS: Multi-lingual approaches to the automatic annotation of speech.** *The Phonetician.* 2015; **111–112**: 54–69.

Bodt TA: **An integrated FLEx–ELAN workflow for linguistic analysis with multiple transcriptions and translations and multiple participants.** *Language Documentation & Conservation.* 2022; **16**: 417–452.

Boersma P, Weenink D: **Praat: Doing phonetics by computer.** 2024. Version 6.4, computer program.
Reference Source

Conneau A, Baevski A, Collobert R, *et al.*: **Unsupervised cross-lingual representation learning for speech recognition.** *Proceedings of Interspeech.* 2021; **2021**: 2426–2430.
Publisher Full Text

van Dam KP: **Preparing acoustic pitch data for computational analysis and presentation.** *Computer-Assisted Language Comparison in Practice.* 2024; **7**(2): 79–94.
Publisher Full Text

van Dam KP: **Besra.** *Zenodo.* 2026. Version 0.9, computer software.
Publisher Full Text

van Dam KP, Stephen A: **Automated quality control for language documentation: Detecting phonotactic inconsistencies in a Kokborok wordlist.** *Proceedings of the Fifth Workshop on NLP Applications to Field Linguistics.* Association for Computational Linguistics; 2026; pages 1–7. Rabat.
Publisher Full Text

FFmpeg Developers: **FFmpeg.** 2025. Computer program.
Reference Source

Foley B, Arnold J, Coto-Solano R, *et al.*: **Building speech recognition systems for language documentation: The CoEDL endangered language pipeline and inference system (Elpis).** *Proceedings of the 6th Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU).* 2018; pages 205–209.
Publisher Full Text

Forkel R, List J-M, Greenhill SJ, *et al.*: **Cross-linguistic data formats, advancing data sharing and re-use in comparative linguistics.** *Scientific Data.* 2018; **5**: 180205.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Hammarström H, Forkel R, Haspelmath M, *et al.*: **Glottolog 5.0.** 2024.
Reference Source

Harris CR, Jarrod Millman K, van der Walt SJ, *et al.*: **Array programming with NumPy.** *Nature.* 2020; **585**: 357–362.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Hunter JD: **Matplotlib: A 2D graphics environment.** *Computing in Science & Engineering.* 2007; **9**(3): 90–95.
Publisher Full Text

Jadoul Y, Thompson B, de Boer B: **Introducing Parselmouth: A Python interface to Praat.** *Journal of Phonetics.* 2018; **71**: 1–15.
Publisher Full Text

Li X, Dalmia S, Li J, *et al.*: **Universal phone recognition with a multilingual allophone system.** *Proceedings of ICASSP.* 2020; **2020**: 8249–8253.
Publisher Full Text

List J-M, Forkel R: **LingPy: A Python library for historical linguistics.** 2023. Version 2.6.13, computer software.
Reference Source

List J-M, Anderson C, Tresoldi T, *et al.*: Cross-linguistic transcription systems (CLTS). dataset. 2021. Version 2.1.0.
Reference Source

List J-M, van Dam KP, Blum F: **EDICTOR 3: Interactive tool for computer-assisted language comparison.** 2025. Version 3.1, computer software.
Reference Source

McAuliffe M, Socolof M, Mihuc S, *et al.*: **Montreal forced aligner: Trainable text-speech alignment using Kaldi.** *Proceedings of Interspeech.* 2017; **2017**: 498–502.
Publisher Full Text

McFee B, Raffel C, Liang D, *et al.*: **librosa: Audio and music signal analysis in Python.** *Proceedings of the 14th Python in Science Conference.* 2015; pages 18–24.
Publisher Full Text

Mehta DD, Rudoy D, Wolfe PJ: **Kalman-based autoregressive moving average modeling and inference for formant and antiformant tracking.** *The Journal of the Acoustical Society of America.* 2012; **132**(3): 1732–1746.
PubMed Abstract | **Publisher Full Text**

Michaud A, Adams O, Cohn T, *et al.*: **Integrating automatic transcription into the language documentation workflow: Experiments with Na data and the Persephone toolkit.** *Language Documentation & Conservation.* 2018; **12**: 393–429.

Moran S, Cysouw M: *The Unicode Cookbook for Linguists: Managing Writing Systems Using Orthography Profiles.* Berlin: Language Science Press; 2018.

Python Software Foundation: **The Python package index (PyPI).** 2026. Software repository.
Reference Source

Rauch HE, Tung F, Striebel CT: **Maximum likelihood estimates of linear dynamic systems.** *AIAA Journal.* 1965; **3**(8): 1445–1450.
Publisher Full Text

Rose Y, MacWhinney B: **The PhonBank project: Data and software-assisted methods for the study of phonology and phonological development.** Durand J, Gut U, Kristoffersen G, editors. *The Oxford Handbook of Corpus Phonology.* Oxford University Press; 2014; pages 380–401.
Publisher Full Text

SIL International: **FieldWorks language explorer (FLEx).** 2025. Version 9.2, computer software.
Reference Source

Skilton A, Ananthanarayan S, Pierson SG, *et al.*: **Flibl: A tool to ease text transfer between ELAN and FLEx.** *Language Documentation & Conservation.* 2025; **19**: 158–175.

Talkin D: **REAPER: Robust epoch and pitch EstimatoR.** 2015. Computer program.
Reference Source

The Qt Company: **Qt for Python (PySide6).** 2025. Computer software.
Reference Source

Vaswani A, Shazeer N, Parmar N, *et al.*: **Attention is all you need.** *Advances in Neural Information Processing Systems.* 2017; **30**.

Vaysse R, Astésano C, Farinas J: **Performance analysis of various fundamental frequency estimation algorithms in the context of pathological speech.** *The Journal of the Acoustical Society of America.* 2022; **152**(5): 3091–3101.
PubMed Abstract | **Publisher Full Text**

Virtanen P, Gommers R, Oliphant TE, *et al.*: **SciPy 1.0: Fundamental algorithms for scientific computing in Python.** *Nature Methods.* 2020; **17**(3): 261–272.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**

Winkelmann R, Harrington J, Jänsch K: **EMU-SDMS: Advanced speech database management and analysis in R.** *Computer Speech & Language.* 2017; **45**: 392–410.
Publisher Full Text

Wittenburg P, Brugman H, Russel A, *et al.*: **ELAN: A professional framework for multimodality research.** *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC).* 2006; pages 1556–1559.
Publisher Full Text