Language Identification Software
Technical Manual

Version 1

6 October 2023

Jordan Wylde-Browne, Toby Coy, Leo Headley,

Thomas Ambrose, Theodore Tiong, Brayden Ransom-Frost

Table of Contents

Click an entry to jump to that page.

1. Application Structure & Architecture

There are two main Flutter applications, the main application and the daughter app. The main application is used for language identification, audio playback, and answer recording. The daughter app is intended to allow for easier modification of the database which manages the questions & statements and audio.

1.1. Main Application

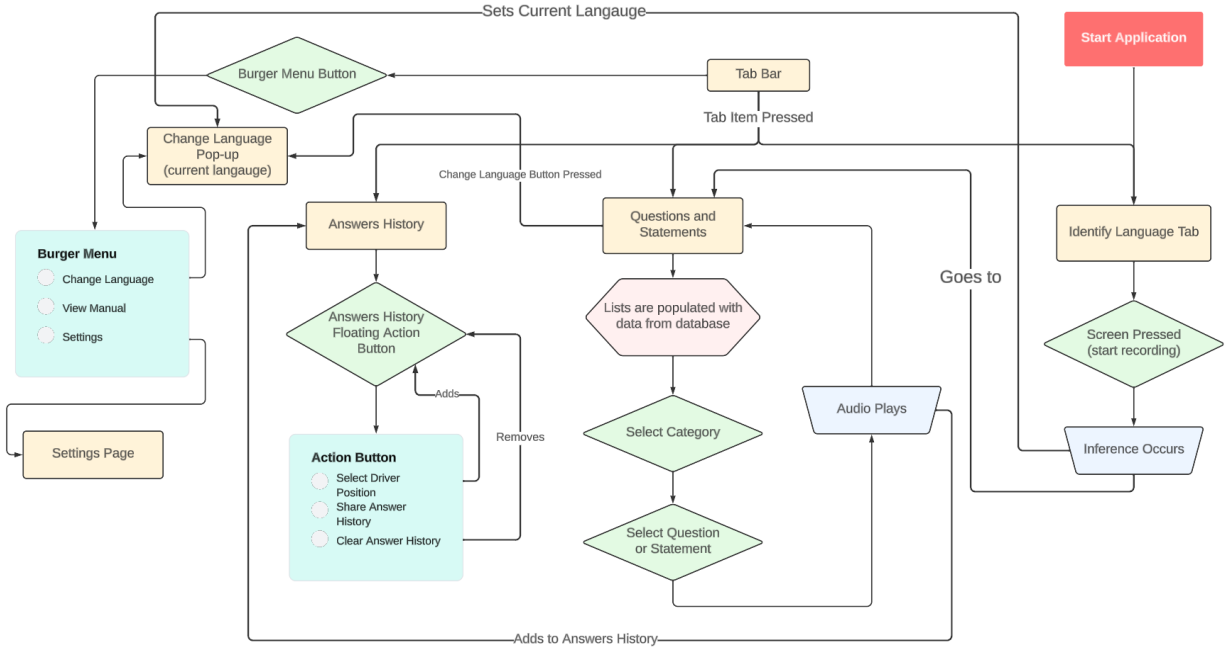


Figure 1. User flow of the main application.

As shown in figure 1, the user flow of the application can be quite complex. It begins at the invitation to speak screen, which is the expected start point for all use cases. Users can press the screen to record audio, and then it will predict the language spoken. When a language has been identified, questions can be asked through the questions and statements screen, which play audio back in the identified language. Any questions that prompt responses will have answers saved into some global state, and these can be viewed in the answers history page. The settings page, accessed by the burger menu, allows for customization of some functionality in the app as well as various utilities.

1.1.1. Major Components

The app is split into three main pages, all controlled through a central tab controller in *main.dart*. The app sits under all the providers used for managing global state, such as the categories, language, and various utilities. These providers are then passed into individual

pages and widgets as needed. Other components of the application are accessed through either the burger menu or various buttons throughout the app.

Invitation to Speak

This screen contains the audio recorder and animation. When the user presses any part of the screen, audio recording starts, and the animation changes state. When the recording is finished, language inference is completed on the saved recording, and the global language state is updated. This screen has no state of its own, but manages the state of the answers and language providers.

Questions & Statements

The questions screen displays a list of categories and a list of questions under that categories. These are loaded from the categories provider, which is initialised at the start of runtime. This screen has a lot of state to manage, as the list of questions is based on the currently selected question. Additionally, it manages audio playback within the same screen. It must communicate and interface with most providers in the application. The user is only able to directly modify the state of the viewable questions by either changing the language, or downloading new ones through the settings page.

Answers History

The answers tab exists to display the list of answers from the answers provider, as well as giving the user the avenue for managing this state, such as clearing the list or deleting answers. The screen itself manages almost no state of its own, and is primarily there to act as a view for the answers model.

Settings

The settings page communicates with a number of providers as it gives the user the ability to update and modify state in the application. It allows users to download audio and questions from the database, saving the files to the device, as well as changing colour themes, and modifying parts of the functionality of the app. It is built as a list of components which each provide different functions and widget types, and is built on top of the *settings_ui* package.

1.2. Daughter Application

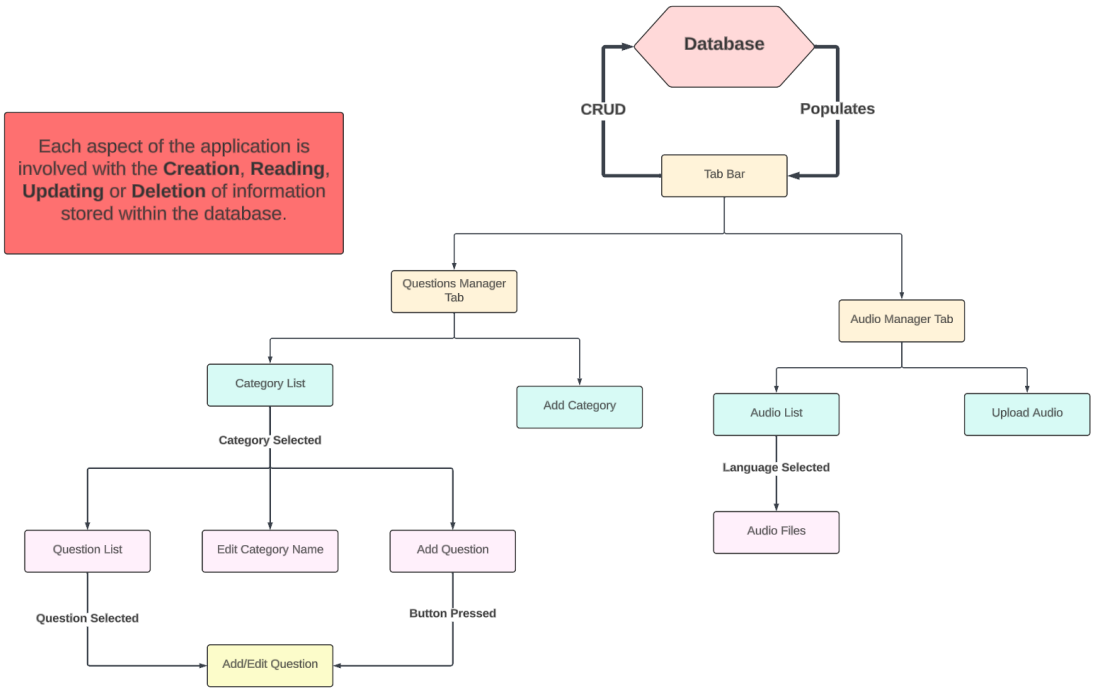


Figure 2. User flow of the daughter application.

Figure 2 highlights the relationship the daughter application has with its database. The flow is much more simple that the main application shown in section 1.1. as there are only 2 tabs. The way the daughter application is laid out can be compared to the structure of the database as found in **section X**. Administrators are able to navigate to either of the 2 tabs, if they go to the “Questions Manager”, administrators are able to create, edit or delete categories and/or questions and statements within those categories. In the “Audio Manager" tab, administrators are then able to either add or delete audio. It must also be mentioned that uploading audio needs to be a careful process, audio files must be uploaded correctly with the correctly named and formatted filename as demonstrated in **section X**.

1.2.1. Major Components

The app is split into two main pages, all controlled through *main.dart*. The app sits under all the providers used for managing global state, such as the categories and audio. These providers are then passed into individual pages and widgets as needed.

Questions Manager

In the questions manager, there are two main aspects, the first is the add category button and the second is the categories list. The add category button will take you to a page which enables the user to add a category, the only relevant information here that needs to be inputted is the category title. The category list is the second aspect, here, users can scroll through all the created categories, long hold and drag to reorder them and swipe from right to left to delete a category and all its contents. Reordering a category will affect its look within the main application as the ordering in the daughter application is reflected within the main

application. Users may also tap on a category to move into the actual questions and statements area for that specific category.

After selecting a category, users can edit the category name by clicking on category title and changing its name, returning back to the category list will confirm this change. Pressing the add question button will add a question where the user MUST enter a full length text, short length text, an audio ID number and a type. Pressing save will confirm this change but returning will not. Similarly to the categories list, users may swipe to delete questions, long press and drag to reorder (reflected in the main application also) and then finally tap on to edit. Tapping on a question or statement will take the user to the same page as the add question will, however each aspect will already be filled out in which they can then edit and click save to confirm changes, returning will not confirm these changes.

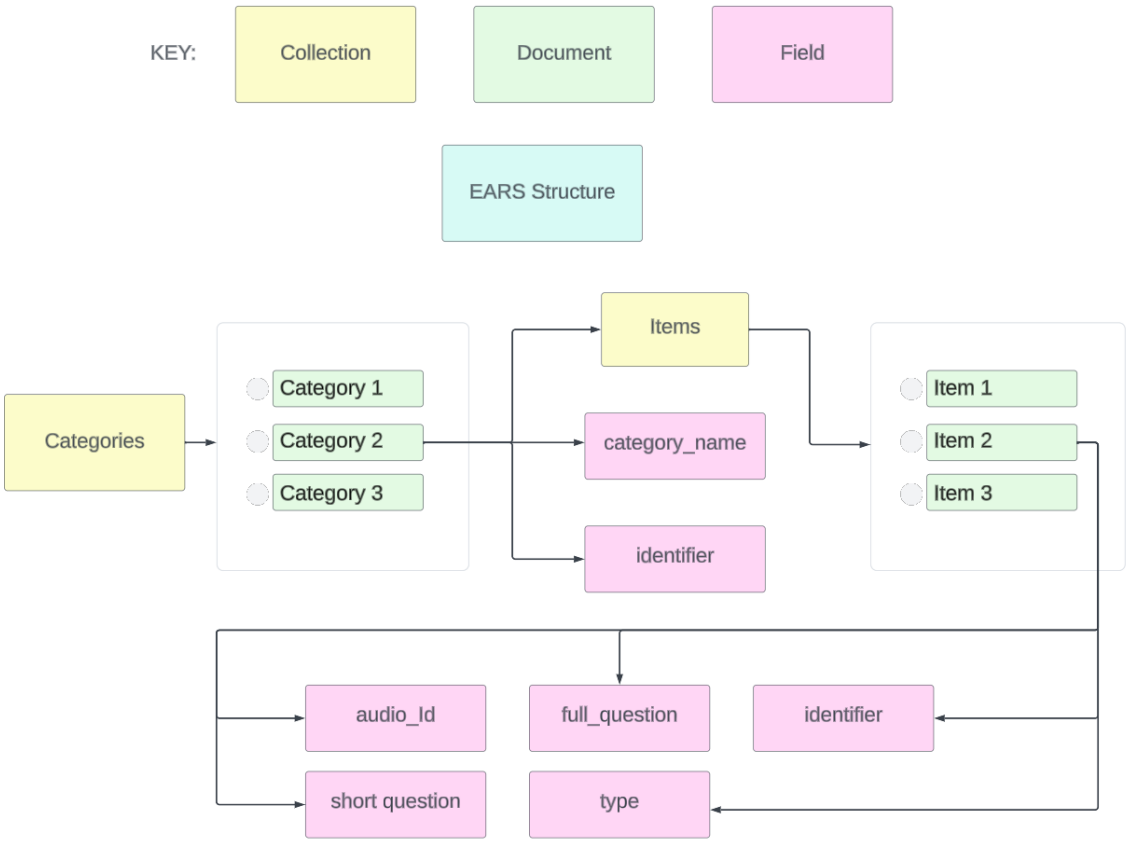
Audio Manager
In the audio manager tab, users can either upload audio from the device, or select a language from the list of languages. Selecting a language will take the user to a list of each audio file within that language where users can choose to delete specific files.

2. Database Structure

EARS uses Firebase for all facets of database storage. It uses Firebase to store questions & categories data, as well as audio data for all languages. These are managed through the secondary application or through the Firebase web interface.

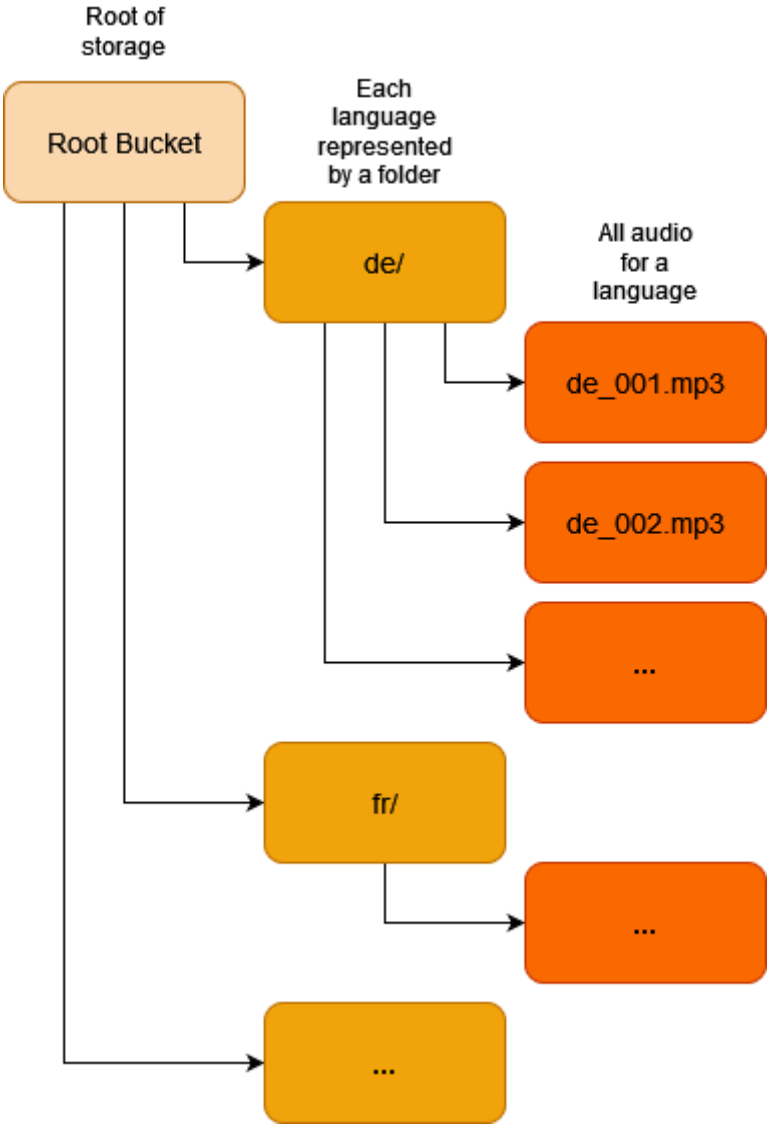
2.1. Firebase Firestore

Firebase Firestore is a NoSQL database. Data is stored as JSON and synchronised in realtime to every connected client. The EARS project uses Firestore to store data on the questions and statements in the application. Firestore uses collections, documents and fields to organise its data. Categories are like folders, documents are like the files within a folder and the fields are the relevant information stored on those files. The EARS structure is as follows:



2.2. Firebase Cloud Storage

Firebase Cloud Storage is a service for storing large pieces of data, such as audio or images, on the cloud. EARS uses this service to store all audio for the application. Files are stored in buckets, and conceptually organised into folders. All data is stored under the root bucket for the project, and then organised by named folders.


Root of storage
Root Bucket
Each language represented by a folder
de/
All audio for a language
de_001.mp3
de_002.mp3
...
fr/
...
...

3. Source Code

3.1. Application Source Code

Filename	main.dart
Purpose	The main dart file contains the driver and main widget for the application, which is a tab view. This is the start point for the application.
Classes, Functions & Content	***main()*** Drives the app and starts runtime, also initialises Firebase as well as some utilities. **MyApp** Builds *MyHomePage* with the various providers needed. **MyHomePage** The main application widget contains the tab bar controller and initialises state for user preferences. **NavigationTab** A custom styled tab for the navigation tabs in the top bar. **BurgerMenu** Creates a burger menu widget to be placed at the top of the tab bar, leading to various utilities in the application.

3.1.1. Main Pages

Filename	invitation_tab.dart
Purpose	The invitation tab file contains the main widget for this screen. It is used to display an animation and record audio, then predict the language spoken.
Classes & Content	**InvitationTab** Contains all state for the invitation to speak page, and renders an *AudioRecorder* widget as its primary view.

Filename	questions_tab.dart
Purpose	The questions tab dart file contains all relevant widgets to the questions and statements screen, including question widgets, the main screen, and all dialogs. It is used to access a large array of questions to play audio for.
Classes & Content	**QuestionsTab** Contains all state for the questions & statements page, rendering two lists displaying all questions and statements from the categories model. Also contains state and logic for playing audio from the questions list. **YesNoDialog** Renders a dialog to display the options to press yes or no in response to a question. **ScaleRatingDialog**

	Renders a dialog to display a scale rating from 1 - 10 in response to a question. **MultipleChoiceDialog** Renders a dialog to display a multiple choice answer in response to a question. **CustomSliderThumbCircle** Custom styled slider thumb used for the scale rating dialog. **ConfirmationDialog** Renders a dialog for confirming audio playback, manages some state for the question pressed.

Filename	answers_tab.dart
Purpose	This file contains all relevant widgets to the answers history screen, including FABs, and the screen itself. It is used to view a history of answers after asking questions.
Classes & Content	**AnswersTab** Contains all state for the answers history page, and renders the list of answers from the provider. **FabWithIcons** Creates an extended floating action button which is used for various utilities on the screen. **SeatPositionDialog** Renders a dialog for modifying the answers provider and selecting a seat position.

Filename	settings_screen.dart
Purpose	This dart file contains all relevant widgets to the settings screen, including dialogs and the screen itself. It is used to manage personal settings throughout the app and access utility features.
Classes & Content	**DownloadButton** A small styled button widget for downloading items. **Section** A custom section for a settings list, styled as needed. This extends an AbstractSettingsSection from the *settings_ui* package. **SettingsWidget** The entire settings screen contains a list of settings and all buttons and functions associated with updating user preferences.

3.1.2. Utility & Miscellaneous

Filename	audio_recorder.dart
Purpose	The audio recorder file contains widgets for managing the audio recorder used in *invitation_tab.dart*. It is used to record audio and then save it to the

	device.
Classes & Content	**AudioRecorder** The widget for the audio recorder, containing all states for both the animation and the audio recording.

Filename	log.dart
Purpose	Small utility file extending a custom Logger for debugging.
Classes & Content	**Log** Extends a debugger with some custom styling.

Filename	pdf_viewer.dart
Purpose	The PDF viewer contains a widget for managing a pdf viewer, primarily used for viewing the user manual in the application.
Classes & Content	**PDFViewerFromAsset** Loads a PDF from assets then displays it in a reader on screen as a widget.

3.1.3. Language Inference

Filename	ml_inference.dart
Purpose	This file contains no widgets, but instead a number of functions used for language inference in *invitation_tab.dart*.
Functions & Content	***predictLanguage(String)*** Loads audio, transforms it into a spectrogram, then completes inference. ***loadAudio(String)*** Loads audio given a path, deletes the file, then returns the signal. ***inference(Matrix)*** Loads the TensorFlow interpreter and then completes inference given an input signal matrix.

Filename	spectrogram.dart
Purpose	This file contains no widgets, but instead a large number of functions for processing audio into spectrograms ready for language inference in *ml_inference.dart*.
Functions & Content	***msFrames(int, int)*** Converts ms to number of frames given some sample rate and ms ***hertzToMel(double[])*** Maps hertz frequencies to a mel scale. ***cmvn(Matrix)*** Computes the central mean variance normalisation of a given matrix.

	stft(double[], fn(), int, int, int, float[]) Computes the short time fourier transform for a given input signal. ***computeMelWeightsMatrix(int, int, int, double, double)*** Computes a mel weights matrix given some parameters. ***melSpectrogram(double[])*** Creates and returns a log mel spectrogram given some signal. ***removeSilence(double[], int)*** Removes silence from a signal using framewise VAD. ***nonOverlapFrame(double[], int)*** Transforms an input signal into non overlapping frames. ***rootMeanSquare(double[][], int)*** Computes the root mean square of an input. ***invertShortConesecutive(bool[], int)*** Used by VAD decisions to invert binary decisions. ***framewiseVAD(double[], int, int, int, int, double, double, int)*** Computes framewise VAD decisions for an input signal for noise filtering.

3.1.4. Providers

The following are all files that contain change notifier providers, used to maintain some global state across the application.

Filename	answers.dart
Purpose	The answers provider maintains state on the answers history and any answers to question dialogs.
Classes & Content	**Answer** Abstracts information for an answer, including the question asked and the answer given. **AnswersModel** The provider for the answers model.

Filename	audio_downloader.dart
Purpose	The audio downloader provider is used to maintain downloading state while downloading audio from the database.
Classes & Content	**AudioDownloader** The provider for the audio downloader.

Filename	category.dart
Purpose	The category provider manages the list of categories and questions, as well as any state associated with loading these, from the database or from local files.
Classes &	**Category**

Content	Abstracts information for a category, including the name and list of questions. **Question** Abstracts information for a question, including the name, id, and question itself. **CategoriesModel** The provider for the category model.

Filename	themes.dart
Purpose	The themes provider manages state on the colour themes used throughout the application, as well as the currently selected theme.
Classes & Content	**ThemeModel** The provider for the theme model. **AppThemeData** Wrapper of ThemeData class to give themes a name.

Filename	language.dart
Purpose	The language provider manages state and interfaces with the language list used in the application. It also contains widgets for the language dialog.
Classes & Content	**LanguageModel** The provider for the language model. **LanguageDialog** The dialog for changing the language.

3.2. Machine Learning Source Code

Filename	main.py
Purpose	Begins training a model using the language list folder specified in its first command line argument.
Functions & Content	Requires one command line argument: the path to the folder containing the training data **tsv_to_lang_dataframe()** Loads language .tsv files, organises them so that we have the targeted split (eg 70% train+dev, 30% test), and loads them into a dataframe for use. **batch_extract_features()** Creates a dictionary containing the log mel spectrograms to be passed into the model. **pipeline_from_metadata()** Begins most of the dataset processing, ending by extracting features in a

	dictionary with batch_extract_features().

Filename	reduce_size.py
Purpose	Reduces a collection of datasets down to the size specified inside the file.
Functions & Content	Target directory to be trimmed is found on like 168 ('datadir=...'). Amount to trim the collection down to is found on line 198 ('target size = ...')

Filename	convert.py
Purpose	Converts a completed model into the tflite format. This lets it run on the tablet.
Notes	Requires two command line arguments: 1) Path to the training dataset 2) Path to the trained model

Filename	model-test.py
Purpose	Outputs summary charts showing dataset statistics, and then prints a table showing the per-language accuracy of the specified model.
Notes	Requires one command line argument: the path to the training dataset folder. Note: this script assumes that the model to be loaded is in cv-workdir/cache/model/

3.3. Daughter Application Source Code

3.3.1. Main Pages

Filename	main.dart
Purpose	The main dart file contains the driver and main widget for the application, which is a bottom tab view. This is the start point for the application.
Classes, Functions & Content	***main()*** Drives the app and starts runtime, also initialises Firebase as well as some utilities. **MyApp** Builds *MyHomePage* with the various providers needed. **MyHomePage** The main application widget contains the tab bar and initialises state for the

	app. **CategoriesList** Contains a list view of all the categories from the database. **ButtonAddGeneric** Styled button for creating new elements in the database.

Filename	questions_list.dart
Purpose	The questions list file contains all widgets associated with the questions list view, including dialogs and the screen itself.
Classes & Content	**QuestionList** Contains all state for the questions list, rendering a single list view as well as some editable fields for changing the category name and adding new questions.

Filename	add_question.dart
Purpose	This file contains the widget for adding new questions to a category. It will interface with the category provider to update the database.
Classes & Content	**AddQuestion** Contains all editable controllers and fields for adding or editing a question to the list of questions & statements in a category.

Filename	add_category.dart
Purpose	This file contains the widget for adding new categories to the database It will interface with the category provider to update the database.
Classes & Content	**AddCategory** Contains all editable controllers and fields for adding or editing a category to the list of categories

Filename	audio_file_picker.dart
Purpose	This file contains widgets for viewing the data in the audio model. It renders a list view.
Classes & Content	**AudioFilePicker** Contains all state for the audio list, rendering a list view of all the languages, as well as displaying audio files underneath them, interfacing with the audio model. It also implements a file picker for uploading audio.

Filename	audio_list.dart
Purpose	This file contains widgets for viewing the data in the audio model. It renders a list view and allows for manipulation of the model.

Classes & Content	**AudioList** Contains all state for an audio list for all audio under a language. Allows for deletion of audio and interfaces with the model for communication with the database.

3.1.1 Providers

Filename	category.dart
Purpose	The category provider manages the list of categories and questions, as well as any state associated with loading these from the database.
Classes & Content	**Category** Abstracts information for a category, including the name and list of questions. **Question** Abstracts information for a question, including the name, id, and question itself. **CategoriesModel** The provider for the category model.

Filename	audio_model.dart
Purpose	The audio model provider manages state for the list of audio, as well as state associated with loading audio from the database.
Classes & Content	**AudioModel** The provider for the audio model.

4. Development Tools & Dependencies

4.1. Development Tools

Name	Python
Type	Programming Language
Version	3.8.17/ 3.9.17
Purpose	Python is the primary language used for machine learning, it is object-oriented and popular for this use case.

Name	Flutter
Type	Software Development Kit
Version	>=3.7.0
Purpose	Flutter is a software development framework created by Google used to create cross-platform applications.

Name	Dart
Type	Programming Language
Version	>=2.19.5 < 3.0.0
Purpose	Dart is the primary language used by the Flutter framework.

Name	Visual Studio Code
Type	Text Editor
Version	1.82.3
Purpose	Visual Studio Code, created by Microsoft, is the primary code editor used for the Flutter & Python development.

Name	Android Studio
Type	Integrated Development Environment
Version	Electric Eel | 2022.1.1 Patch 2
Purpose	Android Studio is the official development environment for Android development and was used by some team members for Flutter development.

4.2. Python Dependencies

Name	Numpy
Type	Python Package
Version	1.24.3
Purpose	General purpose high level number manipulation.

Name	Tensorflow
Type	Python Package
Version	2.12.1
Purpose	Primary framework used for training the model.

Name	Matplotlib
Type	Python Package
Version	3.7.2
Purpose	Used for graphing and displaying results from testing.

Name	Pandas
Type	Python Package
Version	1.1.5
Purpose	Table and data manipulation.

Name	Seaborn
Type	Python Package
Version	0.12.2
Purpose	Visual representations of data.

Name	Miniaudio
Type	Python Package
Version	1.59

Purpose	Audio processing and reading.

4.3. Flutter Dependencies

Name	Logger
Type	Flutter Plugin
Version	1.4.0
Purpose	A logging framework used to debug the application during development.

Name	Share Plus
Type	Flutter Plugin
Version	7.1.0
Purpose	A simple plugin for creating dialogs for sharing information across devices.

Name	Cloud Firestore
Type	Flutter Plugin
Version	4.8.2
Purpose	A plugin used to maintain, connect to, and communicate with cloud storage through Firebase.

Name	Firebase Storage
Type	Flutter Plugin
Version	11.2.6
Purpose	A plugin used to maintain, connect to, and communicate with a Firebase database.

Name	Flutter Sound
Type	Flutter Plugin
Version	9.2.13
Purpose	Plugin used for audio recording and manipulation.

Name	Audioplayers
Type	Flutter Plugin
Version	4.1.0
Purpose	Plugin used for audio playback.

Name	FFTea
Type	Flutter Plugin
Version	1.3.1
Purpose	FFTea is a plugin which runs optimised Fast Fourier Transforms, functions used when converting the audio for inference.

Name	Ml Linalg
Type	Flutter Plugin
Version	13.11.31
Purpose	Utility package used for its linear algebra functions.

Name	Scidart
Type	Flutter Plugin
Version	0.0.2-dev.12
Purpose	Utility package used for its linear algebra, as well as FFT helper functions.

Name	Wav
Type	Flutter Plugin
Version	1.3.0
Purpose	Simple package which allows for WAV file manipulation.

Name	Tflite Flutter
Type	Flutter Plugin
Version	0.10.1
Purpose	Allows for loading and usage of TFLite models for running machine learning

	inference on devices.

Name	Settings UI
Type	Flutter Plugin
Version	2.0.2
Purpose	A modular settings page builder plugin, used for building a custom settings list.

Name	Flutter PDFview
Type	Flutter Plugin
Version	1.3.1
Purpose	Simple implementation of a PDF viewer within a Flutter application.

Name	Permission Handler
Type	Flutter Plugin
Version	10.3.0
Purpose	Handles permissions for audio recording and file handling.

Name	Rive
Type	Flutter Plugin
Version	0.11.4
Purpose	An animation framework, used primarily for the invitation to speak screen animation.

Name	Flutter Launcher Icons
Type	Flutter Plugin
Version	0.13.1
Purpose	Used to create custom icons for the application.

Name	Shared Preferences
Type	Flutter Plugin

Version	2.2.1
Purpose	State management and user preferences plugin used to remember settings across uses.

5. Updating the Application

The scale of the project is large and has many components to it. The following is some general guidelines on updating the application and modifying it for future use.

5.1. Application Updates

The application is built with the Flutter SDK, all relevant versions for Dart and Flutter can be found in Section 4.1. The project can be set up as standard for Flutter projects.
The app will not run on web builds, due to the drivers used for machine learning. It has been primarily tested and run on a Nexus 9 API 34 emulator.
When the project is set up, it can be built and run on the device, and changes can be made.

Changes to the questions & statements, and audio, can be managed through the daughter application, or directly through the Firebase database web interface. When files are downloaded through settings they are loaded into a local cache, these can be deleted or removed as needed. Questions & statements are loaded as json files, and audio as mp3.

The available languages are managed through the *assets/ml/label_maps.json* file, and loaded in *language.dart*, which is simply an array of languages with their code and full title. Note that the order of these languages does matter, as the language model will use a simple index to identify the language from the list, so it must match the order of the languages the model has been trained on.

Additionally, under the *assets/ml/* folder is the currently used language model. Any number of models can be added here, however only the one specified in *ml_inference.dart* will be used.

The user manual is found under *assets/pdf/user_manual.pdf*. It is specified in *pdf_viewer.dart*, this can be changed there.

5.1.1. Tensorflow Updates

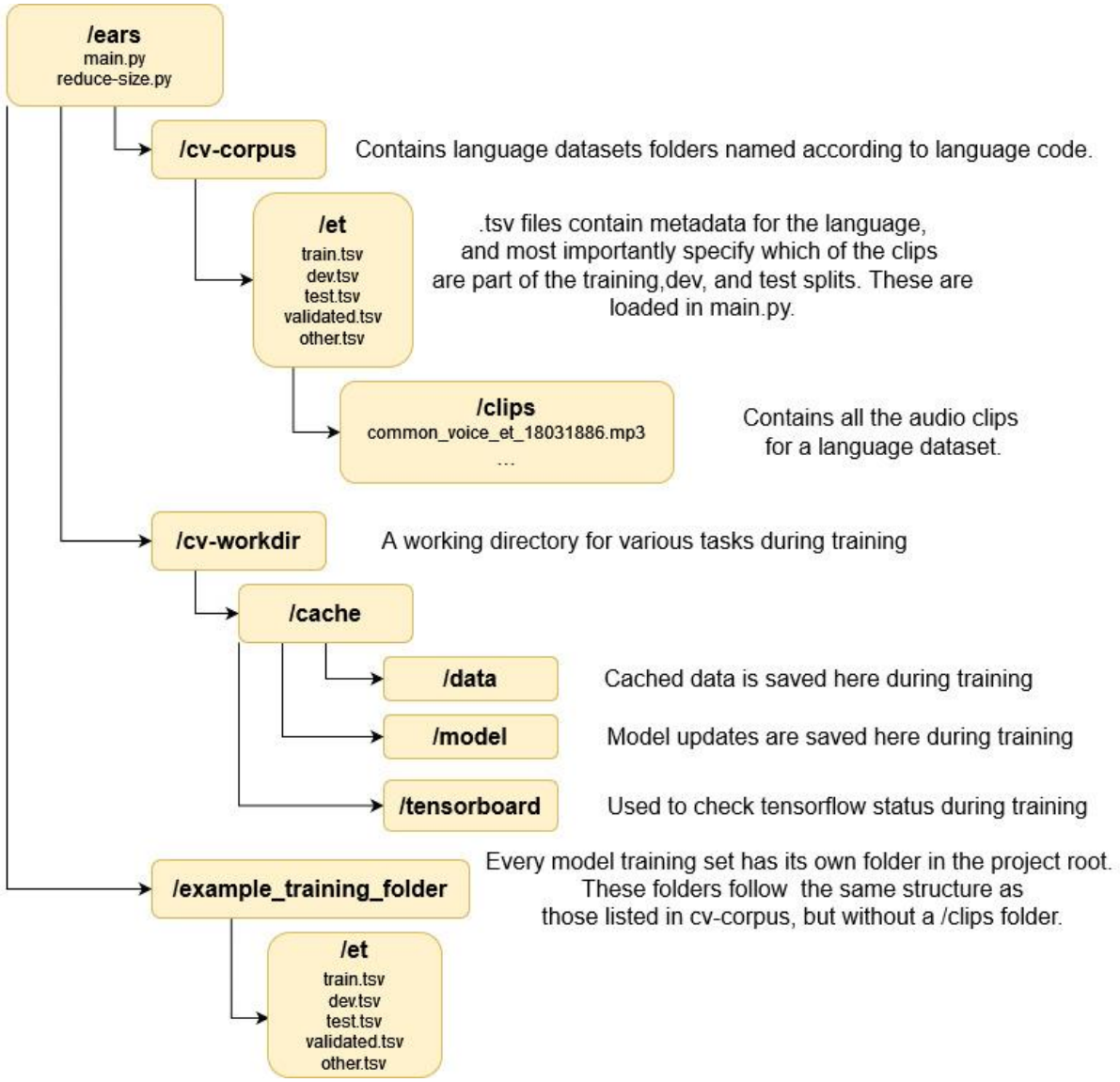
Note that the application uses machine learning on-device using its GPU. This requires special installation of drivers to ensure the device can correctly use Tensorflow. In the project, the drivers should be seen under *android/app/src/main/jniLibs/*. If they are not, then run the installation script found under the root of the project.

5.2. Language Model

1. Running the training script.
2. Converting the resulting model to a format usable on tablets.

5.2.1. Directory Structure

The project structure is based on the setup used in the Lidbox documentation (https://py-lidbox.github.io/common-voice-small/main.html), with the inclusion of separate folders for training sets for ease of testing multiple model setups.



5.2.2 Installing Project Dependencies

Step 1: Install tensorflow using the steps in the tensorflow docs

Found here: https://www.tensorflow.org/install/pip. Note: make sure to use Python 3.8 when setting up the Conda virtual environment. Development took place on Ubuntu 20.04.6 LTS, but we still needed to follow the additional steps relating to Ubuntu 22.04 at the bottom of the tensorflow docs. Ensure that the virtual environment you created here is activated *every time* you attempt to run the project, because all project dependencies will be installed into the environment and will not exist outside it.

Step 2: Install the python dependencies used in the training script.

a) Either install these using the list in section 4.2 of this document, or optionally let python tell you what needs to be installed: making sure you have the virtual

environment from step 1) activated, and ensuring that main.py is in the project root, navigate in the terminal to that directory and run ‘python3 main.py test’.

b) You will receive an error about a package not being recognised. In the terminal, run ‘pip install <package name>==<version specified in section>’
c) Eg: ‘pip install numpy==1.24.3’
d) Rerun the script: ‘python3 main.py test’
e) If you get a module error saying that yaml was not found: 'pip install pyyaml'
f) Repeat until you receive an error that the test directory was not found - this means that all dependencies have been installed and the script has now begun attempting to open a training folder named ‘test’, which does not exist.

5.2.3. Preprocessing data

The data used to train a model was from the open source Mozilla common voice project. Not all of this data is the same size, with some languages having much more data than others. The purpose of the ‘reduce_size.py’ script is to level out the sizes of each language, thus reducing the effect this discrepancy has on the model.

‘reduce_size.py’ finds all of the languages included in a given directory and then gets the size of the ‘clips’ folder in each language’s directory. It starts by removing all clips listed in the ‘invalidated.tsv’ and ‘other.tsv’ files, getting the new size of each ‘clips’ folder afterwards. The size of each ‘clips’ folder is then compared with a set target size to determine how much needs to be removed. Each language is then loaded into a dataframe and a dictionary is created with the ids of individual speakers and how often they contributed. Clips are then deleted one at a time, each time removing a clip from the individual with the most number of contributions.

5.2.4. Training a model

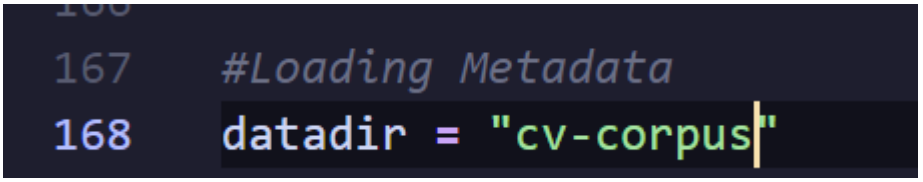
Download the Datasets and set up Project Directory Structure

Model training largely follows the process laid out in the Lidbox example found at https://github.com/py-lidbox/examples. See common-voice-small (https://py-lidbox.github.io/common-voice-small/main.html) for the closest analogue to main.py. To train a model, ensure that you have a training folder set up in the project root. The rest of these steps assume you have made a folder called ‘test_example’. It also assumes that you have downloaded several Common Voice datasets from the Mozilla Common Voice project. Those datasets should be extracted into the ‘cv-corpus’ directory so that it has the structure seen in section 5.2.1. If you want to store your datasets in a folder with a different name, you must edit line 35 of main.py as seen below. It must contain the path of the directory relative to main.py.

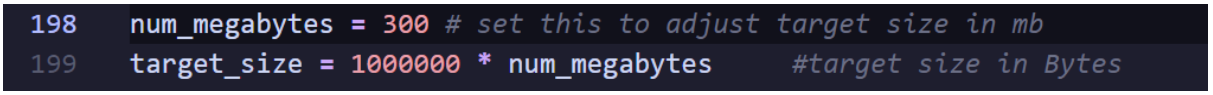
```
33  '''Set up the 3 directories'''
34  workdir = os.getcwd()+"/cv-workdir" # place to save the cached data during training
35  datadir = os.getcwd()+"/cv-corpus" # source of the audio clips
36  tsvdir = os.getcwd()+ "/" + sys.argv[1] # source of the metadata files
```

Trim the Datasets

Next, trim the datasets to your desired size using reduce_size.py. As with main.py, if you changed the name of the data directory or want to manage multiple data directories (say, for different dataset sizes), make sure to set the path of the directory (relative to the script location) on line 168:



Make sure to set the target size for each language here:



With datadir and num_megabytes set correctly, run 'python3 reduce_size.py'. Each language folder with be trimmed to the specified size, and files that haven't been validated by Common Voice will be removed.

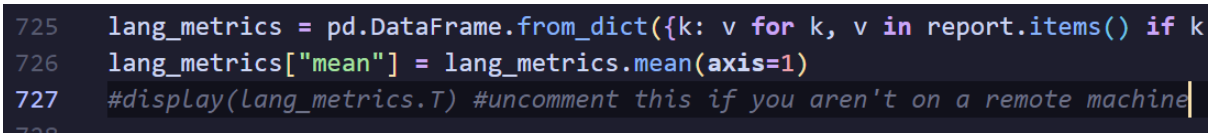
Set up a Training Folder

Next, populate test_example with language folders for each language you want to test. These should have the same name as languages in cv-corpus (ie the name of each folder must be a language code). Inside each folder should be the *.tsv files for that language, which you can source from the language folders in cv-corpus.

Begin Training

IMPORTANT: each time you train a model in the following step, files will be cached in /cv-workdir/cache. You MUST delete the cache folder between each model test if the datasets being used (eg the language list or dataset size) change in any way. If you don't delete the cache, the model training will complete but will produce completely incorrect values.

If you're running this locally on a machine with a desktop, uncomment the final line of main.py so that you can see the accuracy statistics when training has concluded. If you're working remotely, you can run the model-test.py script to get the model stats after training.



It's time to make the model: run 'python3 main.py test_example'. When training has completed, the model files will be saved to test_example/model.

[Optional] Get Model Stats

If you need to, ensure model-test.py is in the project root, and then run 'python3 model-test.py test_example' to get the model statistics. This will need to reprocess the training splits, which takes substantial time.

Convert Model to TFLite Format

Finally, convert the model to the tflite format to run it on a tablet device. Move the model folder from /test_example/model into the project root. Ensure conver.py is in your project root, then run 'python3 convert.py test_example model'. To change where the script expects each folder, edit the following lines in convert.py:

```
7   tsvdir = os.getcwd()+ "/" + sys.argv[1] #path of the training directory
8   model_dir = os.getcwd()+ "/" + sys.argv[2] #path of the trained model
```

The model will be saved to the project root. If you provide a more complex path (ie if the model directory isn't in root), the model will be saved along that path, in the parent directory of the model directory.

You now have a converted model to port into the application.

5.3. Database

The database can be updated through the web interface on Firebase. Content in the database is also intended to be managed through the daughter application, however this is high level management and only allows for modification of data, not the structure of the database itself.

5.3.1. Daughter Application

The daughter application offers the questions manager and the audio manager for modifying and uploading data in the database.

The questions manager allows users to view, create, and remove categories. Within the categories is the list of questions associated. All of these can be edited or removed. New questions can also be added from this screen.

The audio manager lists all conceptual folders under the root bucket in the Firebase Storage. New audio can be uploaded from here. As long as the audio follows the correct format (*lang_id.mp3*), it will automatically be assigned to the correct folder on upload, creating a new one if necessary. The file uploader supports batch uploading.
Within each folder is a list of every single audio file under it. These can be individually removed. When all audio is removed from a folder, the folder itself is removed.

5.3.2. Firebase Web Interface

The login details for the Google account used for Firebase management can be found as part of the deliverable for the project, under *login_details.txt*. Logging into the Firebase console allows users to view and manage all data under both the Firestore and Cloud Storage databases.

As Firestore is a NoSQL database, there is no structure to maintain. However, accessing the database allows for granular control over data, particularly in cases where something breaks. Similarly, the Cloud Storage is simply a storage solution for audio files, there is no storage to maintain. However, accessing the web interface allows users to manually delete or upload files in the case that something is wrong.