
DiViMe Documentation

Release 1.0

ACLEW

Jul 03, 2019

Contents:

1	Before starting	1
1.1	What is the ACLEW DiViMe?	1
1.2	Who is the ACLEW DiViMe for?	1
1.3	What exactly is inside the ACLEW DiViMe?	1
1.4	How should I cite ACLEW DiViMe?	2
2	Installing DiViMe	3
2.1	First installation	3
2.2	Checking your installation	4
2.3	Common installation errors and fixes	4
2.4	Updating DiViMe	5
2.5	Uninstallation	5
2.6	References	5
3	Using DiViMe	7
3.1	Overview	7
3.2	Further information on Step 1, putting your data into the <code>data/</code> directory	7
3.3	Further information on Step 2, going to the DiViMe folder	8
3.4	Further information on Step 3, Waking the machine up	8
3.5	Further information on Step 4, Using tools on data	8
3.6	An alternative for Step 4: using recipes	13
3.7	Further information on Step 5, putting DiViMe back to sleep	13
4	More information about DiViMe	15
4.1	Pipeline Structure	15
4.2	Building a virtual machine	15
4.3	Folder Structure	16
5	Formats	17
5.1	Input: TextGrid	17
5.2	Input: Eaf	17
5.3	Speech or Voice activity detection output	17
5.4	Diarization style (diarization or role assignment) output	18
6	Speech or Voice activity detection tools	19
6.1	NoisemesSad	19
6.2	OpenSmile SAD	22

6.3	TOCombo SAD	23
7	Talker diarization tools	25
7.1	DiarTK	25
8	Other tools	27
8.1	Yunitator	27
8.2	VCM	28
9	Evaluation	29
9.1	Speech/voice activity detection and diarization evaluation	29
10	Extra tools	31
10.1	Getting sample data	31
10.2	Using scripts in the Utilities	31
11	Troubleshooting	33
11.1	Installation issues	33
11.2	Problems with some of the Tools	34
12	Instructions For Contributors	35
12.1	Overview	35
12.2	Before You Start	35
12.3	Understanding the general structure of DiViMe	36
12.4	Adapting Your Tool to the VM Environment	36
12.5	Write a Wrapper	37
12.6	Document Your Tool	39
12.7	Create a Reproducible Test for Your Tool	39
12.8	Check reproducibility of your version of the VM by reprovisioning	40
12.9	Integrate Your Tool Into the Public Version of DiViMe	40
13	References	41
14	Indices and tables	43

1.1 What is the ACLEW DiViMe?

It is a collection of speech processing tools allowing users to automatically add annotations onto a raw audio recording. At present, we have tools to do the following types of annotation:

1. Speech activity detection (*when is someone talking?*)
2. Talker diarization (*who is talking?*)
3. Role diarization (*what kind of person is talking?*)
4. Vocal type classification (*what kind of vocalization is this one?*)

We are hoping to add more tools in the future, including register detection, and syllable quantification.

1.2 Who is the ACLEW DiViMe for?

ACLEW DiViMe is for researchers dealing with speech recorded in naturalistic environments (speech in the wild), typically in daylong recording settings. The population recorded may be sensitive, therefore researchers may not be able to share their audio recordings. Our primary test case involves language acquisition in children 0-3 years of age.

We are trying to make the use of these tools as easy as possible, but some command line programming/scripting is unavoidable. If you are worried when reading this, we can recommend the Software Carpentry programming courses for researchers, and particularly their [unix bash](#) and [version control](#) bootcamps.

1.3 What exactly is inside the ACLEW DiViMe?

A virtual machine (VM) is actually a mini-computer that gets set up inside your computer. This creates a virtual environment within which we can be sure that our tools run, and run in the same way across all computers (Windows, Mac, Linux).

Inside this mini-computer, we have tried to put several tools for each one of our three questions. Please note that some of the tools are developed by fellow researchers and programmers, and since we do not control them, we cannot be absolutely certain they will work. Therefore, we provide a general introduction to the contents in the usage section, and a specific list of tools in dedicated Detailed instructions sections.

1.4 How should I cite ACLEW DiViMe?

The main citation is this paper, which explains the structure and idea, and provides some evaluation:

Adrien Le Franc, Eric Riebling, Julien Karadayi, Yun Wang, Camila Scaff, Florian Metze, and Alejandrina Cristia. *The ACLEW DiViMe: An easy-to-use diarization tool*. In Proc. INTERSPEECH, Hyderabad; India, September 2018.

The idea of using virtual machines to package speech tools comes from this work: Florian Metze, Eric Fosler-Lussier, and Rebecca Bates. The speech recognition virtual kitchen. In Proc. INTERSPEECH, Lyon; France, August 2013. <https://github.com/srvk>.

2.1 First installation

Try the following first:

1. Install [Vagrant](#): Click on the download link and follow the prompted instructions
2. Install [VirtualBox](#): When we last checked, the links for download for all operating systems were under the header “VirtualBox 5.2.18 platform packages”, so look for a title like that one.
3. Clone the present repository: To do this, you must use a terminal. If you don’t know what this means, we recommend that you first follow the [Software Carpentry Shell Tutorial](#) (up to 00:45, namely “Introducing the shell”, and “Navigating files and directories”). Next, navigate to the directory in which you want the VM to be hosted and type in: `$ git clone https://github.com/srvk/DiViMe`
4. Change into this folder: `$ cd DiViMe`
5. Install HTK. HTK is used by some of these tools (until we find and implement an open-source replacement). We are not allowed to distribute HTK, so unfortunately you have to get it yourself.
 - Go to the [HTK download page](#)
 - Register by following the instructions on the left (under “Getting HTK”: Register)
 - Check that you have received your password via email; you will need it for the next step.
 - Find the link that reads “HTK source code” under your system (if you have a mac, it will be under “Linux/unix downloads”). Notice that you will need your username and password (from the previous step). You must download the version 3.4.1.
 - Move the HTK-3.4.1.tar.gz file into the root folder of this repository (alongside Vagrantfile).
6. Type `$ vagrant up`

The first time you do this, it will take at least 20 minutes to install all the packages that are needed to build the virtual machine. Once the virtual machine will be installed, it will stay stuck at “installation finished” for few minutes. However, the tools are not yet installed at this step. You will need to wait for the tools to be installed, and to take back the control of the terminal to run the tools.

The instructions above make the simplest assumptions as to your environment. If you have Amazon Web Services, an ubuntu system, or you do not have admin rights in your computer, you might need to read the [instructions to the eesen-transcriber](#) for fancier options. Or you can just open an issue [here](#), describing your situation.

We are working on [Installing With Docker](#), but this option is not yet functional.

2.2 Checking your installation

The very first time you use DiViMe, it is a good idea to run a quick start test, which will be performed using data from the [VanDam Public Daylong HomeBank](#) corpus (VanDam et al., 2015):

1. Open a terminal
2. Navigate inside the DiViMe folder
3. Do `$ vagrant up`
4. Do `$ vagrant ssh -c "launcher/test.sh"`

This should produce the output:

```
Testing Speech Activity Detection Using Noisemes...
Noisemes passed the test.

Testing OpenSmile SAD...
OpenSmile SAD passed the test.

Testing Threshold Optimized Combo SAD...
Threshold Optimized Combo SAD passed the test.

Testing DiarTK...
DiarTK passed the test.

Testing Yunitator...
Yunitator passed the test.

Testing DScore...
Yunitator passed the test.

Congratulations, everything is OK!
```

2.3 Common installation errors and fixes

- For noisemesSad, and diartk, you may get an error “failed the test because a dependency was missing. Please re-read the README for DiViMe installation, Step number 4 (HTK installation).” This means that your HTK installation was not successful. Please re-download the

If something else fails, please open an issue [here](#). Please paste the complete output there, so we can better provide you with a solution.

2.4 Updating DiViMe

If there is a new version of DiViMe, you will need to perform the following 3 steps from within the DiViME folder on your terminal:

```
$ vagrant destroy  
$ git pull  
$ vagrant up
```

2.5 Uninstallation

If you want to get rid of the files completely, you should perform the following 3 steps from within the DiViME folder on your terminal:

```
$ vagrant destroy  
$ cd ..  
$ rm -r -f divime
```

2.6 References

VanDam, M., De Palma, P., Strong, W. E. (2015, May). Fundamental frequency of speech directed to children who have hearing loss. Poster presented at the 169th Meeting of the Acoustical Society of America, Pittsburgh, PA.

3.1 Overview

This is an overview of the full tool presentation found in the next stage, recapping the main steps:

1. Put your data in the `data` shared directory.
2. Do `$ vagrant up` to “wake the machine up”

Next we provide instructions for all tools. More detailed information about each tool can be found in separate readme’s.

Assuming the installation of the virtual machine is complete and some of the tests have passed, you can now use at least some of the tools in the virtual machine. We explain more about each step below, but in a nutshell, the steps to use DiViMe are always the same:

1. Put the data you want to process in the `data/` directory (or any subdirectory within `data/`)
2. Go to the DiViMe folder
3. Do `$ vagrant up` to “wake the machine up”
4. Use tools on data, typically by doing `vagrant ssh -c "script.sh [arguments]"`. You can also run a recipe.
5. Finally, remember to put DiViMe back to sleep with `$ vagrant halt`

3.2 Further information on Step 1, putting your data into the `data/` directory

Put the sound files that you want analyzed (and annotations, if you have any) inside the shared `data` folder. It is probably safer to make a copy of your files (rather than moving them), in case you later decide to delete the whole folder.

You can drop a whole folder into `data`. You will provide the path to the specific folder to be analyzed when running the tools (as per instructions below). All `.wav` files in that folder will be analyzed.

If your files aren't .wav some of the tools may not work. Please consider converting them into wav with some other program, such as [ffmpeg](#). It is probably safer to make a copy (rather than moving your files into the data folder), in case you later decide to delete the whole folder.

If you have any annotations, put them also in the same “data” folder. Annotations can be in .eaf, .textgrid, or .rttm format, and *they should be named exactly as your wav files*. For more information on these formats, see the Format section.

IMPORTANT: If you already analyzed a file with a given tool, re-running the tool will result in the previous analysis being overwritten.

3.3 Further information on Step 2, going to the DiViMe folder

To interact with the virtual machine, you must use a terminal. If you don't know what this means, we recommend that you first follow the [Software Carpentry Shell Tutorial](#) (up to 00:45, namely “Introducing the shell”, and “Navigating files and directories”).

Next, navigate in the terminal window to the DiViMe directory that was created when you did `git clone https://github.com/srvk/DiViMe` when installing DiViMe.

3.4 Further information on Step 3, Waking the machine up

Remember that you will be using a mini-computer within your computer. Typically, the machine will be down - i.e., it will not be running. This is good, because when it is running, it will use memory and other resources from your computer (which we call “the host”, because it is hosting the other computer). With this step, you launch the virtual machine:

```
$ vagrant up
```

3.5 Further information on Step 4, Using tools on data

3.5.1 Overview of tools

If all tools passed the test, then you'll be able to automatically add the following types of annotation to your audio files:

1. Speech activity detection (*when is someone talking?*): The tools available for this task are the following: noisemesSad, tocomboSad, opensmileSad
2. Talker diarization (*who is talking?*) The tools available for this task are the following: diartk
3. Role diarization (*what kind of person is talking?*) The tools available for this task are the following: yunitator
4. Vocal type classification (*what kind of vocalization is this one?*) The tools available for this task are the following: vcm
5. Evaluation (*how good is the automatic annotation?*) There is an evaluation available for the following tools: noisemesSad, tocomboSad, opensmileSad, diartk, yunitator

3.5.2 Overview of “pipelines”

DiViMe is a platform for tools to analyze naturalistic, unannotated audiorecordings. We consider this process to involve three kinds of processes:

- speech activity detection and voice activity detection = “detecting vocalizations”,
- diarization = “deciding to whom the vocalizations belong”, and
- “additional annotations”

Some tools actually combine two of these stages (e.g. a tool may do both speech activity detection and role attribution in one fell swoop). This [flowchart](#) may help.

We call a *pipeline* a sequence of those processes; i.e., it involves using one tool after another. For example, you may do *speech activity detection* + *talker diarization* + *vocal type classification*

Starting from an audio file with no annotation, typically, you may want to run a *speech activity detection* tool followed by a *talker diarization* tool; then you will end up with an annotation showing who spoke when. However, you may not know who “talker0” and “talker1” are. (You could decide this by listening to some samples of each, and mapping them to different roles. However, we do not provide tools to do this.)

Alternatively, we provide a *role diarization* tool that directly segments recordings into 3 main roles, namely child, male adult, female adult; and these separated from silence.

In both cases, you may want to classify each vocalizations into different types with the *vocal type classification* tool.

3.5.3 How to run a Speech or Voice activity detection tool

For these tools, type a command like this one:

```
$ vagrant ssh -c "noisemesSad.sh data/mydata/"
```

You can read that command as follows:

vagrant ssh -c: This tells DiViMe that it needs to run a tool.

noisemesSad.sh: This first argument tells DiViMe which tool to run. The options are: *noisemesSad.sh*, *tocomboSad.sh*, *opensmileSad.sh*

data/mydata/: This second argument tells DiViMe where are the sound files to analyze. Note that the directory containing the input files should be located in the *data/* directory (or it can be *data/* itself). The directory does not need to be called *mydata* - you can choose any name.

For each input wav file, there will be one rttm file created in the same directory, with the name of the tool added at the beginning. For example, imagine you have put a single file called *participant23.wav* into *data/*, and you decided to run two SADs:

```
$ vagrant ssh -c "opensmileSad.sh data/"
$ vagrant ssh -c "noisemesSad.sh data/"
```

This will result in your having the following three files in your *data/* folder:

- *participant23.wav*
- *opensmileSad_participant23.rttm*
- *noisemesSad_participant23.rttm*

If you look inside one of these .rttm’s, say the *opensmileSad* one, it will look as follows:

SPEAKER	participant23	1	0.00	0.77	<NA>	<NA>
→	speech	<NA>				
SPEAKER	participant23	1	1.38	2.14	<NA>	<NA>
→	speech	<NA>				

This means that opensmileSad considered that the first 770 milliseconds of the audio were speech; followed by 610 milliseconds of non-speech, followed by 2.14 seconds of speech; etc.

3.5.4 How to run a Talker diarization tool

For these tools, type a command like this one:

```
$ vagrant ssh -c "diartk.sh data/mydata/ noisesmesSad"
```

You can read that command as follows:

vagrant ssh -c: This tells DiViMe that it needs to run a tool.

diartk.sh: This first argument tells DiViMe which tool to run. The options are: *diartk.sh*.

data/mydata/: This second argument tells DiViMe where are the sound files to analyze. Note that the directory containing the input files should be located in the *data/* directory (or it can be *data/* itself). The directory does not need to be called *mydata* - you can choose any name.

noisemesSad: Remember that this tool does “talker diarization”: Given some speech, attribute it to a speaker. Therefore, this type of tool necessitates speech/voice activity detection. This third argument tells DiViMe what file contains information about which sections of the sound file contain speech.

You can only use one of the following options: *textgrid*, *eaf*, *rttm*, *opensmileSad*, *tocomboSad*, *noisemesSad*. We explain each of these options next.

You can provide annotations done by a human or in some other way, and encoded as one of the following three file types:

- *textgrid*: this means you want the system to use your *textgrid* annotations. Notice that all tiers count, so if you have some tiers that are non-speech, you should remove them from your *textgrids* before you start. Please note that the system will convert your *textgrids* into *.rttm* in the process.
- *eaf*: this means you want the system to use your *eaf* annotations. Notice that we only know how to properly process *.eaf* files that follow the [ACLEW Annotation Scheme](#). Please note that the system will convert your *eafs* into *.rttm* in the process.
- *rttm*: this means you want the system to use your *rttm* annotations. Notice that all annotations that say “speech” in the eight column count as such.

Alternatively, you can provide use automatic annotations generated by DiViMe’s speech/voice activity detection systems, encoded in *rttm* files. In this case, you would pass one of the following options:

- *this system before*: this means you want the system to use the output of the *noisemesSad* system. If you have not run *noisemesSad*, the system will fail.
- *opensmileSad*: this means you want the system to use the output of the *opensmile* system. If you have not run this system before, the system will fail.
- *tocomboSad*: this means you want the system to use the output of the *tocomboSad* system. If you have not ran this system before, the system will fail.

If the third parameter is not provided, the system will give an error.

If all three parameters are provided, then the system will first find all the annotation files matching the third parameter (e.g., all the files *.TextGrid*; or all the *tocomboSad_.rttm* files), and then find the corresponding sound files. For example, imagine you have put into your *data/mydata/* folder the following files:

- participant23.wav
- opensmileSad_participant23.rttm
- participant24.wav
- participant24.TextGrid

If you run:

```
$ vagrant ssh -c "diartk.sh data/mydata/ opensmileSad"
```

then only participant23.wav will be analyzed.

If you run:

```
$ vagrant ssh -c "diartk.sh data/mydata/ textgrid"
```

then only participant24.wav will be analyzed.

At the end of the process, there will be an added rttm file for each analyzed file. For instance, if you have just one sound file (participant23.wav) at the beginning and you run opensmileSad followed by diartk, then you will end up with the following three files:

- participant23.wav: your original sound file
- opensmileSad_participant23.rttm: the output of opensmileSad, which states where there is speech
- diartk_opensmileSad_participant23.rttm: the output of opensmileSad followed by diartk, which states which speech sections belong to which speakers.

If you look inside a diartk_*.rttm file, it will look as follows:

SPEAKER	file17	1	4.2	0.4	<NA>	talker0	<NA>
SPEAKER	file17	1	4.6	1.2	<NA>	talker0	<NA>
SPEAKER	file17	1	5.8	1.1	<NA>	talker1	<NA>
SPEAKER	file17	1	6.9	1.2	<NA>	talker0	<NA>
SPEAKER	file17	1	8.1	0.7	<NA>	talker1	<NA>

This means that diartk considered that one talker spoke starting at 4.2 seconds for .4 seconds; starting at 4.6 for 1.2 seconds; then someone else spoke starting at 5.8 seconds and for 1.1 seconds; etc.

3.5.5 How to run a Role assignment tool

For these tools, type a command like this one:

```
$ vagrant ssh -c "yunitator.sh data/mydata/"
```

You can read that command as follows:

vagrant ssh -c: This tells DiViMe that it needs to run a tool.

yunitator.sh: This first argument tells DiViMe which tool to run. The options are: yunitator.

data/mydata/: This second argument tells DiViMe where are the sound files to analyze. Note that the directory containing the input files should be located in the *data/* directory (or it can be *data/* itself). The directory does not need to be called *mydata* - you can choose any name.

It returns one rttm per sound file, with an estimation of where there are vocalizations by children, female adults, and male adults.

3.5.6 How to run a Vocalization classification tool

vcm

NO INFORMATION YET

3.5.7 How to run an Evaluation

If you have some annotations that you have made, you probably want to know how well our tools did - how close they were to your hard-earned human annotations.

Evaluating Speech/Voice activity detection

Type a command like the one below:

```
$ vagrant ssh -c "evalSAD.sh data/ noisesemesSad"
```

You can read that command as follows:

vagrant ssh -c: This tells DiViMe that it needs to run a tool.

evalSAD.sh: This first argument tells DiViMe which tool to run. The options are: evalSAD.sh.

data/mydata/: This second argument tells DiViMe where are the sound files to analyze. Note that the directory containing the input files should be located in the *data/* directory (or it can be *data/* itself). The directory does not need to be called *mydata* - you can choose any name.

noisemesSad: The third argument indicates which tool's output to evaluate (in this case, *noisemesSad*). All of our Speech/Voice activity detection tools can be evaluated with this.

For the evaluation process, YOU MUST PROVIDE files that have the coding by humans. For each sound file that has been analyzed with that tool (e.g., in the example, for each file called *noisemesSad*.rttm*), the system will generate the name of the sound file (by removing “*noisemesSad*” and “*.rttm*”). Then it will look for *.rttm* annotations; for instance, in our running example, it will look for a file called *participant23.rttm*. If this does not exist, it will look for *.eaf* files (i.e., *participant23.eaf*). Finally, if those don't exist, it will check for *.textgrid* ones (i.e., *participant23.TextGrid*).

Evaluating Speech/Voice activity detection

Type a command like the one below:

```
$ vagrant ssh -c "evalDiar.sh data/ diartk_noisemesSad"
```

You can read that command as follows:

vagrant ssh -c: This tells DiViMe that it needs to run a tool.

evalSAD.sh: This first argument tells DiViMe which tool to run. The options are: evalSAD.sh.

data/mydata/: This second argument tells DiViMe where are the sound files to analyze. Note that the directory containing the input files should be located in the *data/* directory (or it can be *data/* itself). The directory does not need to be called *mydata* - you can choose any name.

- *diartk_noisemesSad**: The third argument indicates which output to evaluate.

THIS NEEDS WORK

For the evaluation process, YOU MUST PROVIDE files that have the coding by humans. For each sound file that has been analyzed with that tool (e.g., in the example, for each file called *noisemesSad*.rttm*), the system will generate the name of the sound file (by removing “*noisemesSad*” and “*.rttm*”). Then it will look for *.rttm* annotations; for instance,

in our running example, it will look for a file called participant23.rttm. If this does not exist, it will look for .eaf files (i.e., participant23.eaf). Finally, if those don't exist, it will check for .textgrid ones (i.e., participant23.TextGrid).

3.6 An alternative for Step 4: using recipes

THIS NEEDS WORK

3.7 Further information on Step 5, putting DiViMe back to sleep

Last but not least, you should **remember to halt the virtual machine**. If you don't, it will continue running in the background, taking up useful resources! To do so, simply navigate to the DiViMe folder on your terminal and type in:

```
$ vagrant halt
```

More information about DiViMe

DiViMe is a virtual machine, whose characteristics and contents are almost completely determined by provisioning scripts found in the DiViMe folder created when you did `git clone https://github.com/srvk/DiViMe/`. This section provides information to help you understand the DiViMe structure in conceptual terms (the “pipeline”). We also explain the process of “bootstrapping” or creation of this virtual machine. Finally, we describe the folder structure. If you just want to use the tools in DiViMe, you can probably skip this whole section, or read just the `Pipeline structure` section.

4.1 Pipeline Structure

DiViMe is a platform for tools to analyze naturalistic, unannotated audiorecordings. We consider this process to involve three kinds of processes:

- speech activity detection and voice activity detection = “detecting vocalizations”,
- diarization = “deciding to whom the vocalizations belong”, and
- “additional annotations”

Some tools actually combine two of these stages (e.g. a tool may do both speech activity detection and role attribution in one fell swoop).

4.2 Building a virtual machine

The structures and contents of the VM are actually built from reproducible directives which are executed when you do `vagrant up`. In technical terms, this is called VM is *provisioning*. In a nutshell, a file called “Vagrantfile” builds the machine (e.g., creates a virtual machine, allocates memory, installs an operating systems). Another file (`conf/bootstrap.sh`) installs tools within this virtual machine. For detailed documentation, see <https://www.vagrantup.com/docs/vagrantfile/>.

4.3 Folder Structure

By virtue of how DiViMe is provisioned, there is some information that is accessible both from the host machine (i.e., your computer) and the virtual machine (i.e., the minicomputer built inside your computer). These are called `shared folders`.

The following folders are shared between the host and the virtual machine, and thus accessible from both:

- `utils/` contains ancillary files, typically used across more than one tool
- `launcher/` contains files that target users will use to launch different tools
- `conf/` is for configuration files for two potential reasons: they are shared across more than one tool, or to make them more easily editable from outside the VM
- `data/` folder where users put the data to be analyzed, and recover the automatically annotated files

From within the virtual machine, these are inside `/vagrant`: `/vagrant/utils`, `/vagrant/launcher`, etc. Some of these are also accessible from the home of the default user, through links, ie, `~/utils` and `~/launcher`. (the default user is `vagrant` for `virtualbox`)

The following folders can only be found within the virtual machine:

- `repos/` contains any repository that is cloned into the machine, typically tools for the user, but also tools used by other tools to do e.g. feature extraction.

From within the virtual machine, `~/repos` is inside the `vagrant` user's home folder, so you can access it as `~/repos/` or `/home/vagrant/repos`.

This section explains the input and output formats. Each type of tool returns a different type of output, depending on the key information.

5.1 Input: TextGrid

TextGrid is a standard format for speech annotation, used by the Praat software. Notice that all tiers count, so if you have some tiers that are non-speech, you should remove them from your textgrids before you start. Please note that the system will convert your textgrids into .rttm in the process.

5.2 Input: Eaf

Eaf is a standard format for speech annotation, that allows for rich annotation, used by the Elan software. Notice that we only know how to properly process .eaf files that follow the [ACLEW Annotation Scheme](#). Please note that the system will convert your eafs into .rttm in the process.

5.3 Speech or Voice activity detection output

RTTM is an annotation format for audio files well designed for diarization. Explanations about how to write and read .rttm files can be found [here](#). This format is used by the [DiViMe](#).

Tools that are of the SAD type (SAD or VAD) return one rttm per audio file, named toolnameSad_filename.rttm, which looks like this:

SPEAKER	file17	1	0.00	0.77	<NA>	<NA>	speech	<NA>
SPEAKER	file17	1	1.38	2.14	<NA>	<NA>	speech	<NA>

The fourth column indicates the onset of a speech region; the forth column the duration of that speech region. All other columns may be ignored. Regions of non-speech are all the others (e.g., in this example, between .77 and 1.38).

5.4 Diarization style (diarization or role assignment) output

RTTM is an annotation format for audio files well designed for diarization. Explanations about how to write and read .rttm files can be found [here](#). This format is used by the [DiViMe](#).

Diarization-type tools return one rttm per audio file, named `toolnameDiar_filename.rttm`, which looks like this:

SPEAKER	file17	1	4.2	0.4	<NA>	talker0	<NA>
SPEAKER	file17	1	4.6	1.2	<NA>	talker0	<NA>
SPEAKER	file17	1	5.8	1.1	<NA>	talker1	<NA>
SPEAKER	file17	1	6.9	1.2	<NA>	talker0	<NA>
SPEAKER	file17	1	8.1	0.7	<NA>	talker1	<NA>

The fourth column indicates the onset of a region, the identity being indicated in ; the forth column the duration of that speech region. All other columns may be ignored. Regions of non-speech are all the others (e.g., in this example, between .77 and 1.38).

Speech or Voice activity detection tools

This section contains documentation from the different Speech or Voice activity detection tools.

6.1 NoisemesSad

6.1.1 General intro

Noiseme SAD was actually not specifically built as a SAD but rather as a broader “noiseme classifier”. It is a neural network that can predict frame-level probabilities of 17 types of sound events (called “noisemes”), including speech, singing, engine noise, etc. The network consists of one single bidirectional LSTM layer with 400 hidden units in each direction. It was trained on 10h of basically web videos data (Strassel et al., 2012), with the Theano toolkit. The OpenSMILE toolkit (Eyben et al., 2013) is used to extract 6,669 low-level acoustic features, which are reduced to 50 dimensions with PCA. For our purposes, we summed the probabilities of the classes “speech” and “speech non-english” and labeled a region as speech if this probability was higher than all others. The original method in Wang et al. (2016) used 983 features selected using information gain criterion, but we used an updated version from authors Y. Wang and F. Metze in this paper.

6.1.2 Instructions for direct use (ATTENTION, MAY BE OUTDATED)

You can analyze just one file as follows. Imagine that <\$MYFILE> is the name of the file you want to analyze, which you’ve put inside the `data/` folder in the current working directory.

```
$ vagrant ssh -c "OpenSAT/runOpenSAT.sh data/<$MYFILE>"
```

You can also analyze a group of files as follows:

```
$ vagrant ssh -c "OpenSAT/runDiarNoisemes.sh data/"
```

This will analyze all .wav’s inside the “data” folder.

Created annotations will be stored inside the same “data” folder.

This system will classify slices of the audio recording into one of 17 noise classes:

- background
- speech
- speech non English
- mumble
- singing alone
- music + singing
- music alone
- human sounds
- cheer
- crowd sounds
- animal sounds
- engine
- noise_ongoing
- noise_pulse
- noise_tone
- noise_nature
- white_noise
- radio

Some more technical details

For more fine grained control, you can log into the VM and from a command line, and play around from inside the “Diarization with noises” directory, called “OpenSAT”:

```
$ vagrant ssh
$ cd OpenSAT
```

The main script is `runOpenSAT.sh` and takes one argument: an audio file in `.wav` format. Upon successful completion, output will be in the folder (relative to `~/OpenSAT`) `SSSF/data/hyp/<input audiofile basename>/confidence.pkl.gz`

The system will first create features for all the `.wav` files it found, then will place those features in a subfolder `feature`. Then it will load a model, and process all the features generated, producing output in a subfolder `hyp/` two files per input: `<inputfile>.confidence.mat` and `<inputfile>.confidence.pkl.gz` - a confidence matrix in Matlab v5 mat-file format, and a Python compressed data ‘pickle’ file. Now, as well, in the `hyp/` folder, `<inputfile>.rttm` with labels found from a config file `noisemeclasses.txt`

-More details on output format-

The 18 classes are as follows:

```
0      background
1      speech
2      speech_ne
3      mumble
```

(continues on next page)

(continued from previous page)

```

4      singing
5      musicSing
6      music
7      human
8      cheer
9      crowd
10     animal
11     engine
12     noise_ongoing
13     noise_pulse
14     noise_tone
15     noise_nature
16     white_noise
17     radio

```

The frame length is 0.1s. The system also uses a 2-second window, so the i -th frame starts at $(0.1 * i - 2)$ seconds and finishes at $(0.1 * i)$ seconds. That's why 60 seconds become 620 frames. 'speech_ne' means non-English speech

-Sample RTTM output snippet-

SPEAKER	family	1	4.2	0.4	noise_ongoing	<NA>	<NA>	0.37730383873
SPEAKER	family	1	4.6	1.2	background	<NA>	<NA>	0.327808111906
SPEAKER	family	1	5.8	1.1	speech	<NA>	<NA>	0.430758684874
SPEAKER	family	1	6.9	1.2	background	<NA>	<NA>	0.401730179787
SPEAKER	family	1	8.1	0.7	speech	<NA>	<NA>	0.407463937998
SPEAKER	family	1	8.8	1.1	background	<NA>	<NA>	0.37258502841
SPEAKER	family	1	9.9	1.7	noise_ongoing	<NA>	<NA>	0.315185159445

The script `runClasses.sh` works like `runDiarNoisemes.sh`, but produces the more detailed results as seen above.

6.1.3 Main references:

Wang, Y., Neves, L., & Metze, F. (2016, March). Audio-based multimedia event detection using deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP)*, 2016 IEEE International Conference on (pp. 2742-2746). IEEE. [pdf](#)

6.1.4 Associated references:

S.Burger,Q.Jin,P.F.Schulam,andF.Metze,“Noisemes:Manual annotation of environmental noise in audio streams,” Carnegie Mellon University, Pittsburgh, PA; U.S.A., Tech. Rep. CMU-LTI- 12-07, 2012. S.Strassel,A.Morris,J.G.Fiscus,C.Caruso,H.Lee,P.D.Over, J. Fiumara, B. L. Shaw, B. Antonishek, and M. Michel, “Creating havic: Heterogeneous audio visual internet collection,” in *Proc. LREC*. Istanbul, Turkey: ELRA, May 2012. F. Eyben, F. Wenginger, F. Gross, and B. Schuller, “Recent developments in opensmile, the munich open-source multimedia feature extractor,” in *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 2013, pp. 835–838.

6.1.5 Questions and bug reports

<http://github.com/srvk/OpenSAT/issues>

6.2 OpenSmile SAD

6.2.1 General intro

openSMILE SAD relies on openSMILE (Eyben et al., 2013a) to generate an 18-coefficient RASTA-PLP plus first order delta features. It then uses a long short-term memory recurrent neural network (see details in Eyben et al., 2013b) that has been pre-trained on two corpora of read and spontaneous speech by adults recorded in laboratory conditions, augmented with various noise types.

6.2.2 Some more technical details

These are the parameters that are being used with values that depart from the openSMILE default settings (quoted material comes from either of the openSMILE manuals):

monoMixdown = 1, means “mix down all recorded channels to 1 mono channel” noHeader = 0, means read the RIFF header (don’t need to specify the parameters ‘sampleRate’, ‘channels’, and possibly ‘sampleSize’) preSil = 0.1 “Specifies the amount of silence at the turn beginning in seconds, i.e. the lag of the turn detector. This is the length of the data that will be added to the current segment prior to the turn start time received in the message from the turn detector component”; we use a tighter criterion than the default (.2) postSil = 0.1 “Specifies the amount of silence at the turn end in seconds. This is the length of the data that will be added to the current segment after to the turn end time received in the message from the turn detector component.”; we use a tighter criterion than the default (.3)

You can change these parameters locally by doing:

```
$ vagrant ssh
$ nano /vagrant/conf/vad/vadSegmenter_aclew.conf
```

openSMILE manuals consulted:

- Eyben, F., Woellmer, M., & Schuller, B. (2013). openSMILE: The Munich open Speech and Music Interpretation by Large space Extraction toolkit. Institute for Human-Machine Communication, version 2.0. http://download2.nust.na/pub4/sourceforge/o/project/op/opensmile/openSMILE_book_2.0-rc1.pdf
- Eyben, F., Woellmer, M., & Schuller, B. (2016). openSMILE: open Source Media Interpretation by Large fecture-space Extraction toolkit. Institute for Human-Machine Communication, version 2.3. <https://www.audeering.com/research-and-open-source/files/openSMILE-book-latest.pdf>

6.2.3 Main references for this tool:

Eyben, F., Weninger, F., Gross, F., & Schuller, B. (2013a). Recent developments in OpenSmile, the Munich open-source multimedia feature extractor. Proceedings of the 21st ACM international conference on Multimedia, 835–838.

Eyben, F., Weninger, F., Squartini, S., & Schuller, B. (2013b). Real-life voice activity detection with lstm recurrent neural networks and an application to hollywood movies. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 483-487). IEEE.

6.2.4 Questions and bug reports

<https://www.audeering.com/technology/opensmile/#support>

6.3 TOCombo SAD

6.3.1 General intro

This tool's name stands for "Threshold-Optimized" "combo" SAD; we explain each part in turn. It is a SAD because the goal is to extract speech activity. It is called "combo" because it combines linearly 4 different aspects of voicing (harmonicity, clarity, prediction gain, periodicity) in addition to one perceptual spectral flux feature (see details in Sadjadi & Hansen, 2013). These are extracted in 32-ms frames (with a 10 ms stride). The specific version included here corresponds to the Combo SAD introduced in Ziaei et al. (2014) and used further in Ziaei et al (2016). In this work, a threshold was optimized for daylong recordings, which typically have long silent periods, in order to avoid the usual overly large false alarm rates found in typical SAD systems provided with these data.

6.3.2 Main references:

Ziaei, A., Sangwan, A., & Hansen, J. H. (2014). A speech system for estimating daily word counts. In Fifteenth Annual Conference of the International Speech Communication Association. <http://193.6.4.39/~czap/letoltes/IS14/IS2014/PDF/AUTHOR/IS141028.PDF> A. Ziaei, A. Sangwan, J.H.L. Hansen, "Effective word count estimation for long duration daily naturalistic audio recordings," Speech Communication, vol. 84, pp. 15-23, Nov. 2016. S.O. Sadjadi, J.H.L. Hansen, "Unsupervised Speech Activity Detection using Voicing Measures and Perceptual Spectral Flux," IEEE Signal Processing Letters, vol. 20, no. 3, pp. 197-200, March 2013.

6.3.3 Questions and bug reports

Not available

Talker diarization tools

This section contains documentation from the different Talker diarization tools (i.e., given a speech segment, decide who speaks).

7.1 DiarTK

7.1.1 General intro

This tool performs diarization, requiring as input not only .wav audio, but also speech/nonspeech in .rttm format, from human annotation, or potentially from one of the SAD or VAD tools included in this VM.

The DiarTK model imported in the VM is a C++ open source toolkit by Vijayasenan & Valente (2012). The algorithm first extracts MFCC features, then performs non-parametric clustering of the frames using agglomerative information bottleneck clustering. At the end of the process, the resulting clusters correspond to a set of speakers. The most likely Diarization sequence between those speakers is computed by Viterbi realignment.

We use this tool with the following parameter values:

- weight MFCC = 1 (default)
- Maximum Segment Duration 250 (default)
- Maximum number of clusters possible: 10 (default)
- Normalized Mutual Information threshold: 0.5 (default)
- Beta value: 10 (passed as parameter)
- Number of threads: 3 (passed as parameter)

7.1.2 Main references:

D. Vijayasenan and F. Valente, “DiarTK: An open source toolkit for research in multistream speaker diarization and its application to meetings recordings,” in Thirteenth Annual Conference of the International Speech Communication

Association, 2012. <https://pdfs.semanticscholar.org/71e3/9d42aadd9ec44a42aa5cd21202fedb5eae5.pdf>

7.1.3 Questions and bug reports

<http://htk.eng.cam.ac.uk/bugs/buglist.shtml>

This section contains documentation from other tools.

8.1 Yunitator

8.1.1 General intro

Given that there is no reference for this tool, we provide a more extensive introduction based on a presentation Florian Metze gave on 2018-08-13 in an ACLEW Meeting.

The data used for training were:

- ACLEW Starter dataset
- VanDam public 5-min dataset (about 13h; <https://homebank.talkbank.org/access/Public/VanDam-5minute.html>); noiseme-sad used to detect and remove intraturn silences

Talker identity annotations collapsed into the following 4 types:

- children (including both the child wearing the device and other children; class prior: .13)
- female adults (class prior .09)
- male adults (class prior .03)
- non-vocalizations (class prior .75)

The features were MED (multimedia event detection) feature, extracted with OpenSMILE. They were extracted in 2s windows moving 100ms each step. There were 6,669 dims at first, PCA-ed down to 50 dims

The model was a RNN, with 1 bidirectional GRU layer and 200 units in each direction. There was a softmax output layer, which therefore doesn't predict overlaps..

The training regime used 5-fold cross-validation, with 5 models trained on 4/5 of the data and tested on the remainder. The outputs are pooled together to measure performance. The final model was trained on all the data.

The loss function was cross entropy with classes weighted by 1/prior. The batch size was 5 sequences of 500 frames. The optimizer was SGD with Nesterov momentum=.9, the initial LR was .01 and the LR schedule was *=0.8 if frame accuracy doesn't reach new best in 4 epochs

The resulting F1 for the key classes were:

- Child .55 (Precision .55, recall .55)
- Male adult .43 (P .31, R .61)
- Female adult .55 (P .5, R .62)

8.1.2 Main references:

There is no official reference for this tool.

8.1.3 Questions and bug reports

Not available

8.2 VCM

8.2.1 General intro

Two independent models: one (modelLing) to predicts linguistic vs. non-linguistic infant vocalisations; the other one (modelSyll) predicts canonical vs. non-canonical syllables if given a linguistic infant vocalization.

Specifically, the modelLing was trained on an infant linguistic dataset (refer to this paper: https://static1.squarespace.com/static/591f756559cc68d09fc2e308/t/5b3a94cb758d4645603085db/1530565836736/ZhangEtAl_2018.pdf) and modelSyll was trained on another infant syllable vocalisation dataset (refer to this paper: <https://pdfs.semanticscholar.org/2b25/bc84d2c4668e6d17f4f9343106f726198cd0.pdf>).

Feature set: 88 eGeMAPS extracted by openSMILE-2.3.0 on the segment level.

Model: two hidden layers feed-forward neural networks with 1024 hidden nodes per each hidden layer. A log_softmax layer is stacked as an output layer. The optimiser was set to SGD with a learning rate 0.01, and the batch size is 64.

Setups: Both the infant linguistic and syllable vocalisation datasets were split into train, development, and test partitions following a speaker independent strategy.

Results: The results are 67.5% UAR and 76.6% WAR on the test set for the linguistic voc classification; and are 70.4% UAR and 69.2% WAR for the syllable voc classification.

8.2.2 Main references:

There is no official reference for this tool.

8.2.3 Questions and bug reports

<https://github.com/MilesICL/vcm/issues/>

9.1 Speech/voice activity detection and diarization evaluation

For SAD, we employ an evaluation, which returns the false alarm (FA) rate (proportion of frames labeled as speech that were non-speech in the gold annotation) and missed speech rate (proportion of frames labeled as non-speech that were speech in the gold annotation). For TD, we employ the evaluation developed for the DiHARD Challenge, which returns a Diarization error rate (DER), which sums percentage of speaker error (mismatch in speaker IDs), false alarm speech (non-speech segments assigned to a speaker) and missed speech (unassigned speech).

One important consideration is in order: What to do with files that have no speech to begin with, or where the system does not return any speech at the SAD stage or any labels at the TD stage. This is not a case that is often discussed in the literature because recordings are typically targeted at moments where there is speech. However, in naturalistic recordings, some extracts may not contain any speech activity, and thus one must adopt a coherent framework for the evaluation of such instances. We opted for the following decisions.

If the gold annotation was empty, and the SAD system returned no speech labels, then the $FA = 0$ and $M = 0$; but if the SAD system returned some speech labels, then $FA = 100$ and $M = 0$. Also, if the gold annotation was not empty and the system did not find any speech, then this was treated as $FA = 0$ and $M=100$.

As for the TD evaluation, the same decisions were used above for FA and M, and the following decisions were made for mismatch. If the gold annotation was empty, regardless of what the system returned, the mismatch rate was treated as 0. If the gold annotation was empty but a pipeline returned no TD labels (either because the SAD in that system did not detect any speech, or because the diarization failed), then this was penalized via a miss of 100 (as above), but not further penalized in terms of talker mismatch, which was set at 0.

10.1 Getting sample data

10.1.1 ACLEW Starter Dataset

The ACLEW Starter dataset is freely available, and can be downloaded in order to test the tools. To download it, using your terminal, as explained before, go in the DiViMe folder and do: `$ vagrant ssh -c "launcher/get_aclewStarter.sh data/aclewStarter/"`

This will create a folder called `aclewStarter` inside `data`, in which you will find the audio files from the public dataset and their corresponding `.rttm` annotations. At the present time, there are only two matched annotation-wav files, totalling 10 minutes of running recordings

You can then use the tools mentioned before, by replacing the “data/” folder in the command given in the previous paragraph by “`aclewStarter/`”, E.G for `noisemes`:

```
$ vagrant ssh -c "launcher/noisemesSad.sh /"
```

Reference for the ACLEW Starter dataset:

Bergelson, E., Warlaumont, A., Cristia, A., Casillas, M., Rosemberg, C., Soderstrom, M., Rowland, C., Durrant, S. & Bunce, J. (2017). Starter-ACLEW. Databrary. Retrieved August 15, 2018 from <http://doi.org/10.17910/B7.390>.

10.2 Using scripts in the Utilities

10.2.1 `elan2rttm.py`

Convert annotations made using the ELAN tool, in `.eaf` format, into `rttm` transcriptions. Please note that, using this script, some information is lost, notably the vocal maturity annotation (coded in tier `vcm@CHI`), does not appear in the RTTM format. These information could be retrieved and put in the `rttm`. If you need this information in the RTTM, please raise an issue on [github](https://github.com).

10.2.2 textgrid2rttm.py

Convert annotations made using Praat, in .TextGrid format, into rttm transcriptions. Requires:

- `pympi`
- `tgt`

We provide code to translate annotations from other formats into RTTM:

ELAN .eaf format

WARNING: the current version does not handle subtypes when parsing annotations e.g. TIER_ID ‘CHI’ would be written in the RTTM output file but ‘vmc@CHI’ would not. This is due to the time references being based on other TIER_ID’s annotations for subtypes.

From within the machine, you would run the script as follows:

```
python utils/elan2rttm.py -i my_file.eaf -o my_output_folder
```

Praat TextGrid format

From within the machine, you would run the script as follows:

```
python utils/textgrid2rttm.py my_input_folder
```

10.2.3 adjust_timestamps.py

This script is specific to the data in ACLEW, with the ACLEW annotations conventions. It takes as input a daylong recording in wav format (available on databrary), and a transcription in .eaf format that contains annotated segment coded in an “on_off” tier (or “code” tier for some corpora that were annotated before the new convention). It then takes each annotated segment of 2 minutes, extract it from the daylong recording to output a small wav file of 2 minutes, with the name: `corpus_id_onset_offset.wav` where `corpus` is the name of the original corpus, `id` is the name of the daylong recording (which is itself the id given to the recorded child), `onset` is where the segment starts in the daylong recording (in seconds, with 6 digits padded with 0’s if necessary), `offset` is where the segment ends in the daylong recording (with the same convention). For each of these segments extracted, it also writes the annotations in rttm format, with the timestamps adapted to correspond to the small wav, and with the same name as the small wav previously written.

10.2.4 remove_overlap_rttm.py

Take a transcription in RTTM format, and convert it to a SAD annotation in RTTM format. The SAD annotation contains less information, as it only indicated “speech” segment (i.e. the talker is written as “speech”, no matter who the original talker is), and there are no overlap between speech segments.

```
### make_big_corpus.sh
```

This script is called to treat all the daylong recording with their annotations, using the previous `adjust_timestamps.py` script. It also creates gold SAD rttm using the `remove_overlap_rttm.py` script previously described.

11.1 Installation issues

11.1.1 Virtual Machine creation

If your computer freezes after `vagrant up`, it may be due to several things. If your OS is ubuntu 16.04, there's a known incompatibility between VirtualBox and the 4.13 Linux kernel on ubuntu 16.04. What you may do is to install a previous version of the kernel, for example the 4.10, following [these instructions](#), or install the latest version of virtualbox which should fix the problem. If you are not on ubuntu 16.04, or if the previous fix didn't work, it may also be due to the fact that Vagrant is trying to create a Virtual Machine that asks for too much resources. Please ensure that you have enough space on your computer (you should have at least 15Gb of free space) and check that the memory asked for is okay. If not, you can lower the memory of the VM by changing line 25 of the VagrantFile,

```
vbox.memory = 3072
```

to a lower number, such as

```
vbox.memory = 2048
```

11.1.2 Resuming the Virtual Machine

If you already used the VM once, shut down your computer, turned it back on and can't seem to be able to do `vagrant up` again, you can simply do

```
vagrant destroy
```

and recreate the VM using

```
vagrant up
```

If you don't want to destroy it, you can try opening the VirtualBox GUI, go to File -> Settings or Preferences -> Network, click on the Host-only Networks tab, then click the network card icon with

the green plus sign in the right, if there are no networks yet listed. The resulting new default network should appear with the name 'vboxnet0'. You can now try again with `vagrant up`

11.2 Problems with some of the Tools

11.2.1 OpenSmile, DiarTK

If OpenSmile, DiarTK don't seem to work after `vagrant up`, first, please check that you indeed have the htk archive in your folder. If you don't, please put it there and launch:

```
vagrant up --provision
```

This step will install HTK inside the VM, which is used by several tools.

11.2.2 Noisemes

If you use the `noisemesSad` or the `noisemes_full` tool, one problem you may encounter is that it doesn't treat all of your files and gives you an error that looks like this:

```
Traceback (most recent call last):
  File "SSSF/code/predict/1-confidence-vm5.py", line 59, in <module>
    feature = pca(readHtk(os.path.join(INPUT_DIR, filename))).astype('float32')
  File "/home/vagrant/G/coconut/fileutils/htk.py", line 16, in readHtk
    data = struct.unpack(">%df" % (nSamples * sampSize / 4), f.read(nSamples *
→sampSize))
MemoryError
```

If this happens to you, it's because you are trying to treat more data than the system/your computer can handle. What you can do is simply put the remaining files that weren't treated in a separate folder and treat this folder separately (and do this until all of your files are treated if it happens again on very big datasets). After that, you can put back all of your data in the same folder.

11.2.3 Input Format For Transcriptions

If your transcriptions are in TextGrid format but the conversion doesn't seem to work, it's probably because it isn't in the right TextGrid format. The input TextGrid the system allows is a TextGrid in which all the tiers have speech segments (so remove tiers with no speech segments) and all the annotated segments for each tiers is indeed speech (so remove segments that are noises or other non-speech type).

Instructions For Contributors

Temporary instructions: We have reorganized DiViMe to try to facilitate tool incorporation and VM use, but these changes have not yet made it to the main branch. Therefore, all of the following instructions are NOT in the master branch, but in the `major_reorganization` branch. This should make no difference to you, except that you need to check out the right branch to view the VM fitting with these instructions. We provide the code for doing this below.

12.1 Overview

This detailed guide provides you with step-by-step, specific instructions for adapting a tool into the DiViMe environment. The following is a Summary of the steps:

1. Install the VM for yourself
2. Adapt & test your tools in the VM environment, and build the necessary links to other modules.
3. Put your tools and all your custom scripts somewhere where they can be downloaded (e.g., GitHub repo(s))
4. Create a bash script to download and install your tools on the VM (the same steps will be added to a Vagrantfile that controls the entire VM installation)
5. ...

12.2 Before You Start

Before you start, you'll need to get the VM up and running to establish the environment in which your tool will run.

1. Install DiViMe as per the [installation instructions](#), including the `vagrant up` that wakes the machine up.

Temporary instructions: IMPORTANT: After you've cloned DiViMe, you should check out the `major_reorganization` branch, as follows:

```
$ git checkout remotes/origin/major_reorganization
```

If ever you want to go back to the master branch, you'd do:

```
$ git checkout master
```

If you’ve already built one version of the VM, you’ll need to do:

```
vagrant destroy  
vagrant up
```

1. Run the `test.sh` routine. It downloads audio recordings and annotations that you will need to make sure your tool is compatible with the rest of the workflow. Additionally, it will help you spot any issues there may be in your local environment.

12.3 Understanding the general structure of DiViMe

DiViMe is a virtual machine, whose characteristics and contents are almost completely determined by provisioning scripts found at the top level when you clone DiViMe. That means that if you want to contribute to DiViMe, you need to understand its structure, and bootstrapping of the virtual machine.

In the next section, we will walk you through some stages of modifying the contents of the VM, which you will do initially “by hand”. However, remember the contents of the VM are actually built from reproducible directives ran when you do `vagrant up`. (In technical terms, when the VM is *provisioned*.) Therefore, any changes you make by hand when you are logged into the VM will be lost when you run `vagrant destroy`. Please bear that in mind when trying out the steps in the next section.

Additionally, by the end of these instructions you will be ready to propose a revised version of the VM - i.e., a new recipe to *provision* the VM. Therefore, any changes made by hand that you wish to make permanent in the VM should eventually be added to a file called `util/bootstrap.sh`. So you may want to have a copy of `bootstrap.sh` open to make notes in there of the steps you take, which should be reproduced when provisioning the VM.

12.4 Adapting Your Tool to the VM Environment

1. For this section, you’ll be working almost entirely within the virtual machine. So start by doing `vagrant ssh` to log in. This logs you in as the `vagrant` user, which also has `sudo` privileges to do things like install software, change permissions, etc. in the virtual machine.
2. Decide where your tool is made available so that it can be copied into the VM. Ideally, it will be somewhere public, such as GitHub, so that anyone rebuilding the machine will get access to it. Alternatively, your tool might be stored on a server in a location you control, and then pulled into the virtual machine. Please note that the latter solution only preserves the anonymity of your code temporarily; if you get to the end of this document, when you are proposing a new version of the VM including your tool, then the URL needs to be known to the world, and accessible to everyone. Please note that neither alternative forces you to make your code open source. You can also share a binary executable (ask us for instructions if you need help).
3. Import your tool into the VM. Once you have decided where to put your tool, install your code by hand in the location where it will be within the machine: `/home/vagrant/repos`. For example if your code is in a public GitHub repository `https://github.com/srvk/OpenSAT`, you would type into the terminal:

```
cd /home/vagrant/repos  
git clone http://github.com/srvk/OpenSAT
```

If your tool is accessible via URL `https://toolnameurl.com/tool.zip`, you would type into the terminal:

```
cd /home/vagrant/repos
wget -q -N https://toolnameurl.com/toolname.zip
unzip -q -o toolname.zip
```

In `bootstrap.sh`, add the same code to make this step permanently reproducible.

1. If your code requires certain linux packages that are not installed, first install them ‘by hand’ in the VM with a command like `sudo apt-get install <packagename>`. Any packages installed this way should be added similarly to one of the `apt-get` lines in `bootstrap.sh` like:

```
sudo apt-get install -y libxt-dev libx11-xcb1
```

1. The next step is to install additional dependencies, if any are needed. The VM already includes code to install OpenSmile 2.1, matlab v93, python 3, and anaconda 2.3.0. As in the two previous steps, you can do this by hand within the terminal, but you need to add the step to `bootstrap.sh` to make it permanent. For instance, if you need the python package `pympl-ling`, you would type `pip install pympl-ling` by hand into the terminal. Additionally, to make this change permanent (and have the dependencies installed when you reprovision the VM or when someone else rebuilds the VM), you need to add it to the Vagrantfile section where python packages are installed, with code like this:

```
su ${user} -c "/home/${user}/anaconda/bin/pip install pympl-ling"
```

12.5 Write a Wrapper

In this section, we provide detailed instructions for writing a wrapper that allows users to run your tool in a standardized way. The wrapper should be written in bash. Please look at the other launcher wrappers to have an idea of how to structure it, and read on for important considerations.

12.5.1 Naming Conventions

- You choose your own tool’s name. Use anything you want except the other names already in use. We refer to your tool name later as ‘`TOOLNAME`’
- It may be useful to you to decide at what “stage” of diarization your tool operates. A few things will be clearer if you have a good notion of when this is, such as the number of arguments and whether there is already an evaluation/scoring tool that can be re-used with your tool. We explain these things in more detail below.
- To facilitate end users’ understanding of what tool they are using, we have systematically added the stage name to the wrapper’s name, but you don’t have to follow this procedure if you think it will not be useful.
- We have identified three stages with a fixed name: `Sad` (for both speech activity detection and voice activity detection), `Diar` (for speaker diarization and role assignment), and `Add` (for adding annotation dependent on role assignment).
- Other stages or stage combinations do not have fixed names. But please feel free to use these stage names. For instance, if your tool only requires audio files as input, then you can use `Sad`; if it operates on both audio and speech activity detection, then use `Diar`; and if it is specific to one talker role as input, use `Add`.

12.5.2 Tool Autonomy

Tools should be self-aware of where they have been installed, and they should use this awareness to find their dependencies. Said differently, a tool should run “in place” independent of the absolute path it’s installed in. Tool dependency paths should be relative to the tool home folder, which should serve as the working directory. Again, please look at the

other launcher wrappers to reuse the code present at the top of the wrappers, which correctly reconstructs the absolute path of this folder.

12.5.3 Input, Output, and Parameters

Your wrapper should take at least one argument, namely the name of a folder containing data. This folder is appended to “/vagrant” so from your script and the VM’s perspective, data appears in /vagrant/data/. This is actually a shared folder, coming from the host computer working directory. Everything in data/ on the host will in /vagrant/data in the VM, and vice-versa. The default wrapper argument, then, is typically “data/”, but users could also supply “data/mystudy/” or “data/mystudy/baby1/” as the data folder argument. This supports the notion of having multiple datasets in different folders on the host. You can see how other wrappers use this, typically setting the variable `$audio_dir` to /vagrant/data. For the rest of this explanation, we’ll be referring to this folder as DATAFOLDER.

- The wrapper should process all .wav files inside DATAFOLDER and, optionally, associated annotation files, which are in rttm format. (For more information on the rttm output, read [NIST’s 2009 eval plan](#))
- Your wrapper should support processing long sound files. If you have no better way of achieving this, look to `utils/chunk.sh` as an example; it breaks up long files into smaller 5 minute chunks, then iteratively calls a tool (in this case, yours), and concatenates the results.
- Your tool must process many sound files. This may require some optimization, for example loading very large model files into memory for each sound file is less optimal than loading once, then iterating over many files.
- The wrapper should write its *main output* into DATAFOLDER. Typically, this will be one annotation file for each .wav file. If your tool does VAD, SAD, or diarization, this annotation file should respect the rttm format. Additionally, the output file it should be named according to the pattern: `toolname_file_original_name.rttm`. If your tool does something other than VAD, SAD, or diarization, use a format that seems reasonable to you. If your tool returns results on individual files (and not e.g., summary statistics over multiple files), we still ask you to name the file `toolnameStage_file_original_name.ext` – where “ext” is any extension that is reasonable for the annotation format you have chosen.
- You probably also generate two types of ancillary files: Intermediary representation files, such as features extracted from the wav files; and log files, with detailed information of what was done and how. Both should be initially stored in the DATAFOLDER/temp/TOOLNAME folder.
- Intermediary representation files should be deleted – with one exception, introduced below.
- It is a good idea to print out a short log accompanying each batch of files analyzed, stating the key parameters, version number, and preferred citation. You may want to write out a lot more information. However, our target user may not be technically advanced, and thus including long technical logs may do more to confuse than to help. Log files should also be deleted if they are large (>5MB) – with one exception, introduced next.
- Your wrapper should expect an optional argument “--keep-temp”, which is optionally passed in final position. If the last argument to the wrapper is not “--keep-temp”, then ancillary files should be deleted. Code snippet example:

```
KEEPTEMP=false
if [ $BASH_ARGV == "--keep-temp" ]; then
    KEEPTEMP=true
fi
...
if ! $KEEPTEMP; then
    rm -rf $MYTEMP
fi
```

See any script in `launcher/` for examples.

12.6 Document Your Tool

Add a documentation file to the `DiViMe/docs/` folder on your host, in markdown format, containing at least the following three pieces of information: A one-paragraph explanation of what your tool does A reference or citation that people using your tool must employ A short section explaining how to use it. This will typically include a description of input & output formats, which can be replaced with references to the Format section of the docs. You should also include an example command line of how to run the tool. Often, this will be `vagrant ssh 'toolname.sh data/`. Please include any other information that would be useful to users, such as what parameters are available, how to access them, a tiny example input and output, further technical information on how the tool was built, additional references for work using this tool or on which the tool was built, etc.

For an example, see the `tocomboSad` section in the [tools documentation](#).

12.7 Create a Reproducible Test for Your Tool

DiViMe comes equipped with a test script that downloads a publicly available daylong audio file and transcript, which all tools within DiViMe can process (and many can be evaluated). In this section, we provide instructions for you to add your tool to the `test.sh` routine.

By default, all launchers are read-only to avoid accidental editing by newbie users. For the next step, you need to change file permissions so as to be able to edit `test.sh`. From the host machine, type into the terminal:

```
chmod +rw test.sh
```

Open the file `test.sh`, which is inside the launcher folder, and add a section testing your tool, modeling it on the other tests present there. Typically, you will need these lines:

Under “# Paths to Tools” add a line with the path to your tool, eg: `TOOLNAMEDIR=$REPOS/TOOLNAME`

b) Before “# test finished”, add a section like the following:

Example for a Sad type tool:

```
echo "Testing TOOLNAME..."
cd $TOOLNAMEDIR
TESTDIR=$WORKDIR/TOOLNAME-test
rm -rf $TESTDIR; mkdir -p $TESTDIR
ln -fs $TEST_WAV $TESTDIR
$LAUNCHERS/toolnameStage.sh $DATADIR/TOOLNAME-test >$TESTDIR/TOOLNAME-test.log || {
  echo "    TOOLNAME failed - dependencies"; FAILURES=true;}

if [ -s $TESTDIR/toolnameStage_$BASETEST.rttm ]; then
  echo "TOOLNAME passed the test."
else
  FAILURES=true
  echo "    TOOLNAME failed - no RTTM output"
fi
```

In the above example: `DATADIR` is predefined as the test 5 minute data folder `data/VanDam-Daylong/BN32` `TOOLNAME` is whatever human readable name you have given your tool `toolnameStage` is the pattern for the system/launcher name for your tool, for example `opensmileSad` `BASETEST` is the basename of a test input file e.g. `BN32_010007_test` for the 5 minute input file `BN32_010007_test.wav`

Example for a Diar type tool: `** todo: complete**`

Example for an Add type tool: `** todo: complete**`

Run `test.sh`. Only proceed to the next phase if your tool passes the test.

12.8 Check reproducibility of your version of the VM by reprovisioning

Throughout the steps above, you have modified `Vagrantfile/bootstrap.sh` to automatically install your code, any required packages, and any required dependencies. If you have been keeping your `Vagrantfile/bootstrap.sh` in a good state, you should be able to rebuild your version of the virtual machine from scratch.

If necessary, log out from the virtual machine with `control+D`. Then, from the host machine, run the following code to destroy, re-build, and re-run the test:

```
vagrant destroy
vagrant up
vagrant ssh -c "test.sh"
```

WARNING: any changes you made by hand when you were logged into the VM will be lost when you run `vagrant destroy`: to make sure they show up automatically with `vagrant up`, all such dependencies need to be automated (added to `Vagrantfile/bootstrap.sh`). If your tool passes the test in this condition, you are ready to integrate your tool to DiViMe for real.

12.9 Integrate Your Tool Into the Public Version of DiViMe

1. Fork the DiViMe repo
2. Replace `Vagrantfile/bootstrap.sh` with your version of `Vagrantfile/bootstrap.sh`
3. Add in the docs/ your tool's doc
4. Add your wrapper to launcher/
5. Replace `test.sh` with your version containing an additional test case specific to your tool
6. Create a pull request to DiViMe requesting your additions be incorporated

References

Our work builds directly on that of others. The main references for tools currently included and/or data currently used to perform tests are:

- Bergelson, E., Warlaumont, A., Cristia, A., Casillas, M., Rosenberg, C., Soderstrom, M., Rowland, C., Durrant, S. & Bunce, J. (2017). Starter-ACLEW. Databrary. Retrieved October 1, 2018 from <http://doi.org/10.17910/B7.390>.
- Eyben, F. Weninger, F., Gross, F. & B. Schuller. (2013). Recent developments in opensmile, the munich open-source multimedia feature extractor. Proceedings of the 21st ACM international conference on Multimedia, 835–838.
- Eyben, F., Weninger, F., Squartini, S., & Schuller, B. (2013, May). Real-life voice activity detection with lstm recurrent neural networks and an application to hollywood movies. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 483-487). IEEE.
- Räsänen, O., Seshadri, S., & Casillas, M. (2018, June). Comparison of Syllabification Algorithms and Training Strategies for Robust Word Count Estimation across Different Languages and Recording Conditions. In Interspeech 2018.
- Sadjadi, S.O. & Hansen, J.H.L. (2013). Unsupervised Speech Activity Detection using Voicing Measures and Perceptual Spectral Flux. IEEE Signal Processing Letters, 20(3), 197-200.
- VanDam, M., & Tully, T. (2016, May). Quantity of mothers' and fathers' speech to sons and daughters. Talk presented at the 171st Meeting of the Acoustical Society of America, Salt Lake City, UT.
- Vijayasanen, D. & Valente, F. (2012) Diartk: An open source toolkit for research in multistream speaker diarization and its application to meetings recordings. Thirteenth Annual Conference of the International Speech Communication Association, 2012.
- Wang, Y., Neves, L., & Metze, F. (2016, March). Audio-based multimedia event detection using deep recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on (pp. 2742-2746). IEEE. [pdf](#)
- Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D. et al. (2002) The HTK book. Cambridge University Engineering Department.

- Ziaei, A. Sangwan, A., & Hansen, J.H.L. (2016). Effective word count estimation for long duration daily naturalistic audio recordings. *Speech Communication*, 84, 15-23.
- Ryant, N. (2018). Diarization evaluation. <https://github.com/nryant/dscore>, accessed: 2018-06-17.

CHAPTER 14

Indices and tables

- `genindex`
- `modindex`
- `search`