S3PRL

Self-**S**upervised **S**peech **P**re-training and **R**epresentation **L**earning

https://github.com/s3prl/s3prl/

Lead: Leo Yang, Andy T. Liu

Members: Heng-Jui Chang, Haibin Wu, Liang Cheng

Advisor: Prof. Hung-yi Lee

Outline

- S3PRL and its current limit
- Our plan at JSALT
  - Finalize & Test & Release the reconstruction of S3PRL
  - Extend the horizon of S3PRL to Audio SSL
- Timeline

S3PRL and its current limit

Evolution and major usage of S3PRL

Pre-training

Pre-trained model collection

Downstream Benchmarking & SUPERB

2019

Andy T. Liu

Shu-wen Yang

Po-Han Chi

2020

Shu-wen Yang

Andy T. Liu

2021

Shu-wen Yang

Andy T. Liu

Po-Han Chi

Heng-Jui Chang

Xuankai Chang

Yung-Sung Chuang

Zili Huang

Wen-Chin Huang

Tzu-Hsien Huang

Kushal Lakhotia

Yist Lin Y.

Guan-Ting Lin

Jiatong Shi

Hsiang-Sheng Tsai

Wei-Cheng Tseng

Current status

- Many recipes
  - Mockingjay, TERA, APC, …
  - ASR, ASV, SID, …
- But all in a very specific setting
  - A specific loss
  - A specific model
  - A specific data pipeline
- The only reusable components are our upstream collection
  - Which is kind of success that it really simplies things for future research and many paper uses it

More specifically

Pre-training

master ▾ s3prl / s3prl / pretrain /

andi611 Update path in assert msg

..	
apc	
audio_albert	
distiller	
mockingjay	
npc	
spec_augment	
tera	
vq_apc	

Downstream Evaluation

master ▾ s3prl / s3prl / downstream /

yenmeng fix ctc decode

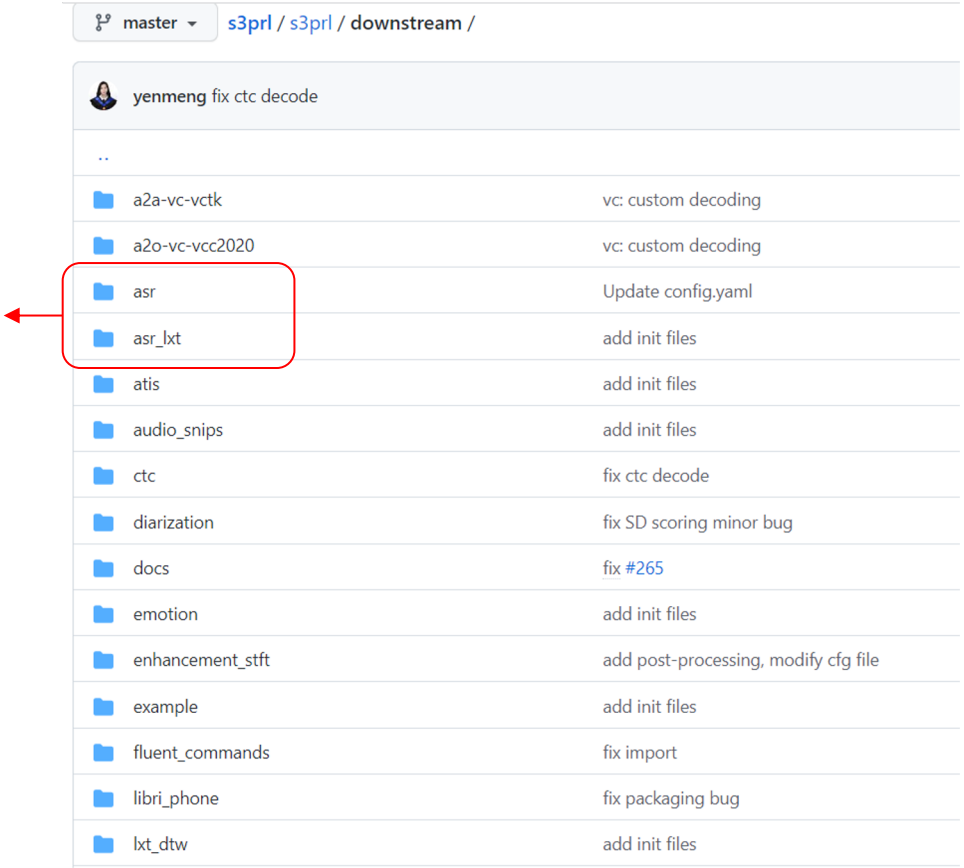
So sooooo many folders…

..	
a2a-vc-vctk	vc: custom decoding
a2o-vc-vcc2020	vc: custom decoding
asr	Update config.yaml
asr_lxt	add init files
atis	add init files
audio_snips	add init files
ctc	fix ctc decode
diarization	fix SD scoring minor bug
docs	fix #265
emotion	add init files
enhancement_stft	add post-processing, modify cfg file
example	add init files
fluent_commands	fix import
libri_phone	fix packaging bug
lxt_dtw	add init files

More specifically

- Each folder entangles lots of things…
  - corpus preprocessing
  - dataloader preparation
  - nn.Module definition
  - Loss / criterion / metric logging
- Any slight setting change leads to a new folder

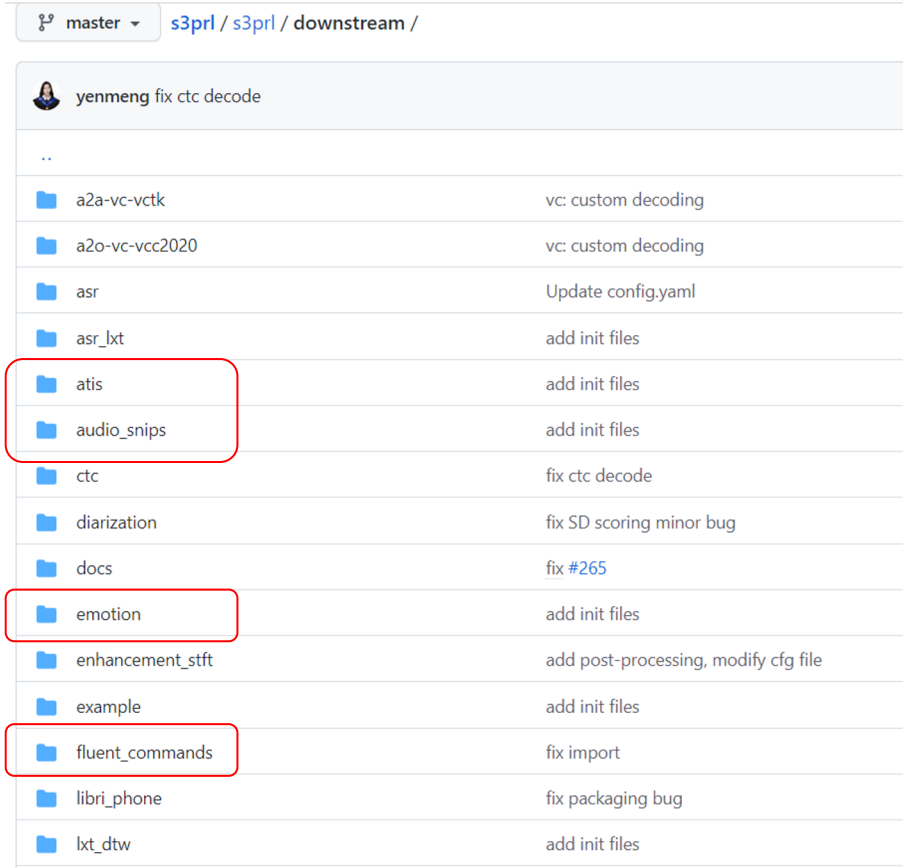
Downstream Evaluation



More specifically

- Duplicated folders for the same utterance classification task (but different corpora)
- This is becoming impossible to maintain by us as the number of folder keep increasing

Downstream Evaluation



More specifically

- Each folder entangles the actual corpus making it hard to change data for the same task and impossible for integration testing

```
18
    You, 11 months ago | 2 authors (tzuhsien and others)
19  class IEMOCAPDataset(Dataset):
20      def __init__(self, data_dir, meta_path, pre_load=True):
21          self.data_dir = data_dir
22          self.pre_load = pre_load
23          with open(meta_path, 'r') as f:
24              self.data = json.load(f)
25          self.class_dict = self.data['labels']
26          self.idx2emotion = {value: key for key, value in self.class_dict.items()}
27          self.class_num = len(self.class_dict)
28          self.meta_data = self.data['meta_data']
29          _, origin_sr = torchaudio.load(
30              path_join(self.data_dir, self.meta_data[0]['path']))
31          self.resampler = Resample(origin_sr, SAMPLE_RATE)
32          if self.pre_load:
33              self.wavs = self._load_all()
```

More essentially...

- It is essentially a collection of recipes
  - You can really easily follow the recipe but hard to adjust the pre-defined setting
- As a pre-training toolkit:
  - Easily change the nn model of TERA? NO
  - Easily pre-train with much more data other than LibriSpeech (Like Voxpopuli)? NO
- As a benchmark toolkit:
  - Easily explore different amount of finetuning paired data? NO
  - Easily change the downstream model for all tasks? NO
  - Easily augment noise for all tasks? NO
  - Easily change the upstream model to benchmark?
    - OK but not so easy

You need to modify every folder one-by-one manually, hoping nothing inconsistent happens

Futhermore

- No any testing: unit-test or integration test
  - From the very begining to now
  - Cool (X)
- Why is this?
  - Originally as a personal (and no-one-care) repository, Andy and I just do anything we like 😅
  - We are always learning 😅 Hence please don't hesitate to let us know we are wrong
- We are fixing this in the new version
  - Including the unit-test on the numerical values of all upstreams

First stage at JSALT

So the first goal at JSALT

- To learn from other famous toolkits (ESPNet, SpeechBrain…): their authors and mantainers, and restructure S3PRL to be more reusable, and more flexible for different pre-training & benchmarking experiment setup
- What we have now

S3PRL 0.4.0 documentation

Search

GETTING STARTED

Install S3PRL

Use S3PRL pre-trained models collection

Use Problem modules to run pre-defined Corpus-Task pairs

HOW TO CONTRIBUTE

Contribute to S3PRL

Internal S3PRL Development

API DOCUMENTATION

nn

base

util

task

metric

corpus

sampler

wrapper

http://140.112.21.12:8000/ (not public yet)

S3PRL

S3PRL is a toolkit targeting for Self-Supervised Learning for speech processing. Its full name is **S**elf-**S**upervised **S**peech **P**re-training and **R**epresentation **L**earning. It supports the following three major features:

- **Pre-training**
  - You can train the following models from scratch:
  - *Mockingjay*, *Audio ALBERT*, *TERA*, *APC*, *VQ-APC*, *NPC*, and *DistilHuBERT*
- **Pre-trained models (Upstream) collection**
  - Easily load most of the existing upstream models with pretrained weights in a unified I/O interface.
  - Pretrained models are registered through torch.hub, which means you can use these models

CONTENTS

What we have now

- The new package design is finalized
- Mockingjay & TERA are migrated
- SUPERB tasks (10) except slot-filling are migrated
  - Not yet completely confirm their backward-compability
  - Net yet friendly documented
  - However, the structure is basically coined, and we are adding details to reproduce the SUPERB setting

The nature of representation benchmarking

- The downstream learning task itself (the pipeline), should not be complicated
  - If it is too complicated, there are too much factors in this benchmarking environment, and can't reflect the direct quality of representation
- The downstream models are also not complicated and have a common set across tasks
  - frame to frame → Linear, RNN, Transformer (might already be overkill)
  - aggregation → Mean / statistic / attentive pooling…
- However, data preparation should be very flexible
  - To test the pretrained model against lots of paired data senario

The nature of representation benchmarking

- The downstream learning task: modularized and reused
- The downstream models: modularized and reused
- Data preparation: should be extenable

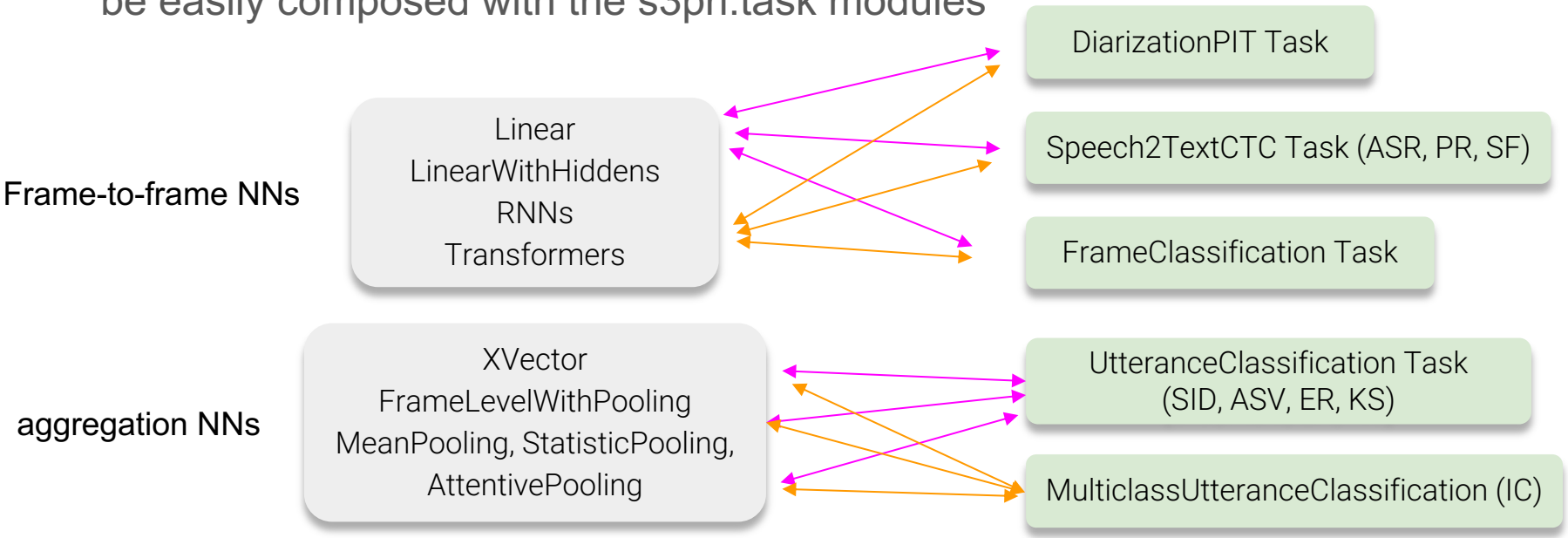
The new package - s3prl.task

- Handles train/valid/test loop
  - loss / metric logging
- And only this one single thing
- No model definition
- No data preparation
- Just interfaces, so that it can be easily reused by any model, any dataset, and different Trainer

```
You, 1 second ago | 1 author (You)
class UtteranceClassificationTask(Task):
    """
    Attributes:
        input_size (int): defined by model.input_size
        output_size (int): defined by len(categories)
    """

    def __init__(self, model: UtteranceClassifierExample, category, **kwargs):
        """
        model.output_size should match len(categories)

        Args:
            model (UtteranceClassifier)
            category:
                encode: str -> int
                decode: int -> str
                __len__: -> int
        """

        super().__init__()
        self.model = model
        self.category = category
        assert self.model.output_size == len(category)
        self._current_best_acc = 0.0
```

The new package - s3prl.nn

- Nothing big deal, just some common nn backbones which are wrapped to be easily composed with the s3prl.task modules



The new package - s3prl.datapipe

- To make dataset preparation extenable, we rely on the nice feature of SpeechBrain: DynamicItemDataset
- Originally each item is just
  - dict(wav="/librispeech/utt1.wav", wrd="Hello how are you")
- After adding a dynamic item: audio_pipeline
  - dict(wav="/librispeech/utt1.wav", wrd="Hello how are you", sig=torch.FloatTensor([.....]))

```
# 2. Define audio pipeline:
@sb.utils.data_pipeline.takes("wav")
@sb.utils.data_pipeline.provides("sig")
def audio_pipeline(wav):
    sig = sb.dataio.dataio.read_audio(wav)
    return sig

sb.dataio.dataset.add_dynamic_item(datasets, audio_pipeline)

# 3. Define text pipeline:
@sb.utils.data_pipeline.takes("wrd")
@sb.utils.data_pipeline.provides(
    "wrd", "tokens_list", "tokens_bos", "tokens_eos", "tokens"
)
def text_pipeline(wrd):
    yield wrd
    tokens_list = tokenizer.encode_as_ids(wrd)
    yield tokens_list
    tokens_bos = torch.LongTensor([hparams["bos_index"]] + (tokens_list))
    yield tokens_bos
    tokens_eos = torch.LongTensor(tokens_list + [hparams["eos_index"]])
    yield tokens_eos
    tokens = torch.LongTensor(tokens_list)
    yield tokens

sb.dataio.dataset.add_dynamic_item(datasets, text_pipeline)

# 4. Set output:
sb.dataio.dataset.set_output_keys(
    datasets, ["id", "sig", "wrd", "tokens_bos", "tokens_eos", "tokens"],
)
```

The new package - s3prl.datapipe

- This kind of logic can appear again and again, and can be modularized and reuse

```
You, 4 days ago | 1 author (You)
@dataclass
class LoadAudio(DataPipe):
    audio_sample_rate: int = 16000
    audio_channel_reduction: str = "first"
    sox_effects: list = None
    crop_segment: bool = False

    wav_path_name: str = "wav_path"
    wav_name: str = "wav"
    start_sec_name: str = "start_sec"
    end_sec_name: str = "end_sec"
```

```
# 2. Define audio pipeline:
@sb.utils.data_pipeline.takes("wav")
@sb.utils.data_pipeline.provides("sig")
def audio_pipeline(wav):
    sig = sb.dataio.dataio.read_audio(wav)
    return sig

sb.dataio.dataset.add_dynamic_item(datasets, audio_pipeline)

# 3. Define text pipeline:
@sb.utils.data_pipeline.takes("wrd")
@sb.utils.data_pipeline.provides(
    "wrd", "tokens_list", "tokens_bos", "tokens_eos", "tokens"
)
def text_pipeline(wrd):
    yield wrd
    tokens_list = tokenizer.encode_as_ids(wrd)
    yield tokens_list
    tokens_bos = torch.LongTensor([hparams["bos_index"]] + (tokens_list))
    yield tokens_bos
    tokens_eos = torch.LongTensor(tokens_list + [hparams["eos_index"]])
    yield tokens_eos
    tokens = torch.LongTensor(tokens_list)
    yield tokens

sb.dataio.dataset.add_dynamic_item(datasets, text_pipeline)

# 4. Set output:
sb.dataio.dataset.set_output_keys(
    datasets, ["id", "sig", "wrd", "tokens_bos", "tokens_eos", "tokens"],
)
```

The new package - s3prl.datapipe

- This kind of logic can appear again and again, and can be modularized and reuse

```
You, 1 second ago | 2 authors (Heng-Jui Chang and others)
@dataclass
class EncodeText(DataPipe):
    text_name: str = "transcription"
    output_text_name: str = "tokenized_text"
    tokenizer_name: str = "tokenizer"

    train_tokenizer: bool = False
```

```
# 2. Define audio pipeline:
@sb.utils.data_pipeline.takes("wav")
@sb.utils.data_pipeline.provides("sig")
def audio_pipeline(wav):
    sig = sb.dataio.dataio.read_audio(wav)
    return sig

sb.dataio.dataset.add_dynamic_item(datasets, audio_pipeline)

# 3. Define text pipeline:
@sb.utils.data_pipeline.takes("wrd")
@sb.utils.data_pipeline.provides(
    "wrd", "tokens_list", "tokens_bos", "tokens_eos", "tokens"
)
def text_pipeline(wrd):
    yield wrd
    tokens_list = tokenizer.encode_as_ids(wrd)
    yield tokens_list
    tokens_bos = torch.LongTensor([hparams["bos_index"]] + (tokens_list))
    yield tokens_bos
    tokens_eos = torch.LongTensor(tokens_list + [hparams["eos_index"]])
    yield tokens_eos
    tokens = torch.LongTensor(tokens_list)
    yield tokens

sb.dataio.dataset.add_dynamic_item(datasets, text_pipeline)

# 4. Set output:
sb.dataio.dataset.set_output_keys(
    datasets, ["id", "sig", "wrd", "tokens_bos", "tokens_eos", "tokens"],
)
```

The new package - s3prl.datapipe

```python
data = {
    "1": {"wav_path": "path1.wav", "text": "hello"},
    "2": {"wav_path": "path2.wav", "text": "good morning"},
}
audio_pipe = LoadAudio(sox_effects=[["gain", "-3.0"]], crop_segment=True)
text_pipe = EncodeText(train_tokenizer=True)
dataset = text_pipe(audio_pipe(data))
# now the dataset can dynamically generate waveforms and tokens

noise_pipe = NoiseAugmentation(snr=[3, 6])
dataset = noise_pipe(dataset)
# now the waveforms are augmented with gaussian noises

chunk_pipe = UnfoldChunkBySec(min_chunk_secs=2, max_chunk_secs=4)
dataset = chunk_pipe(dataset)
# now the waveforms are chunked by 2 to 4 seconds, the __len__ of dataset is increased

dataset2 = SequentialDataPipe(audio_pipe, text_pipe, noise_pipe, chunk_pipe)(data)
# dataset2 is the same as dataset
```

Unified approach to add noises for all task's datasets

Unified approach to crop wavs for all task's datasets

s3prl.corpus → s3prl.datapipe

- **Corpus** parses the corpus directory structure parsing to give a iterable specific for a certain task
- **Datapipe** build the pytorch dataset in a composible way from several common operations
- Disentangle these two make it easy to
  - switch to new data
  - conduct few-shot learning

corpus

(s3prl.corpus)

- s3prl.corpus.base
- s3prl.corpus.fluent_speech_commands
- s3prl.corpus.iemocap
- s3prl.corpus.kaldi
- s3prl.corpus.librispeech
- s3prl.corpus.quesst14
- s3prl.corpus.speech_commands
- s3prl.corpus.voxceleb1sid

dataset (will be renamed to datapipe)

(s3prl.dataset)

- s3prl.dataset.autoregressive_prediction_pipes
- s3prl.dataset.base
- s3prl.dataset.chunking
- s3prl.dataset.common_pipes
- s3prl.dataset.extract_feat_pipes
- s3prl.dataset.masked_reconstruction_pipes
- s3prl.dataset.multiclass_tagging
- s3prl.dataset.noise_augmentation_pipes
- s3prl.dataset.norm_wav_pipes
- s3prl.dataset.pretrain_apc_pipe
- s3prl.dataset.speech2phoneme_pipe
- s3prl.dataset.speech2text_pipe
- s3prl.dataset.utterance_classification_pipe

The new package - s3prl.corpus

```
data = {
    "1": {"wav_path": "path1.wav", "text": "hello"},
    "2": {"wav_path": "path2.wav", "text": "good morning"},
}
audio_pipe = LoadAudio(sox_effects=[["gain", "-3.0"]], crop_segment=True)
text_pipe = EncodeText(train_tokenizer=True)
dataset = text_pipe(audio_pipe(data))
# now the dataset can dynamically generate waveforms and tokens

noise_pipe = NoiseAugmentation(snr=[3, 6])
dataset = noise_pipe(dataset)
# now the waveforms are augmented with gaussian noises

chunk_pipe = UnfoldChunkBySec(min_chunk_secs=2, max_chunk_secs=4)
dataset = chunk_pipe(dataset)
# now the waveforms are chunked by 2 to 4 seconds, the __len__ of dataset is increased

dataset2 = SequentialDataPipe(audio_pipe, text_pipe, noise_pipe, chunk_pipe)(data)
# dataset2 is the same as dataset
```

Can be provided by s3prl.corpus for common corppra

Unified approach to add noises for all task's datasets

Unified approach to crop wavs for all task's datasets

Timeline

- **6/13 ~ 7/1:** Try to finalize SUPERB 10 tasks in the new structure and make sure it is backward compatible with the existing recipe (or with very small variance)
  - Pre-release the PyPi package for people to run SUPERB without cloning S3PRL
  - Install it and use it in their own repository without switching repositories back and forth
- **7/1 ~ 7/8:** Add support to use different Trainer
  - Ideally Speechbrain, PytorchLightning, and Huggingface
  - At least SpeechBrain

Second Stage at JSALT

Audio SSL v.s. Speech SSL

- Two lines of work developed independently but closely related

- **Pretrained on:** all sounds v.s. speech
- **Typical granularity:** segment v.s. frame
- **Mostly evaluated with:** sound event detection v.s. ASR

- They are both SSL on signals just different content

Audio SSL & HEAR Challenge

- There should be a single model understanding all kinds of sound
  - Speech, music, animal sounds, environment sounds…
- Usually only deals with the simple speech classification tasks
  - Google speech commands
  - Voxceleb1 SID
  - FSC intent classification
- Good to be universal beyond speech, but ignoring the most popular speech recognition task

Holistic Evaluation of Audio Representations

Speech SSL > Audio SSL for speech related tasks

	KS Acc ↑	SID Acc ↑
FBANK	8.63	8.5E-4
PASE+ [16]	82.37	35.84
APC [7]	91.04	59.79
VQ-APC [32]	90.52	49.57
NPC [33]	88.54	50.77
Mockingjay [8]	82.67	34.50
TERA [9]	88.09	58.67
modified CPC [34]	92.02	42.29
wav2vec [12]	94.09	44.88
vq-wav2vec [13]	92.28	39.04
wav2vec 2.0 Base [14]	92.31	45.62
HuBERT Base [35]	**95.98**	64.84
HuBERT Large [35]	93.15	**66.40**

SUPERB

Method	Dim.	VC1	SPCV2/12
TRILL [13]		17.9%	74.9%
COLA [14]		29.9%	71.7%
OpenL3 [20] [1]		N/A	N/A
COALA [19] [2]		N/A	N/A
COLA'	512-d	25.9%	59.1%
COLA'	1024-d	31.2%	71.9%
COLA'	2048-d	30.4%	76.7%
BYOL-A	512-d	33.4%	86.5%
BYOL-A	1024-d	38.0%	90.1%
BYOL-A	2048-d	**40.1%**	**91.0%**

BYOL-A

Table 5: ***Generalization to other tasks***. *We show test accuracy (%) on different downstream tasks trained with a linear classifier on top of the frozen features outputed from our pre-trained network, comparing to supervsied and unsupervsied baselines. All methods are based on EfficientNet-B0.*

Task	COLA [14]	Ours	Sup. [39]
Speaker Id. (Librispeech)	**100.0**	99.6	-
Speech commands (V1)	71.7	**80.5**	93.4
Speech commands (V2)	62.4	**82.2**	-
Acoustic scenes	**94.0**	90.4	99.1
Speaker Id. (VoxCeleb)	29.9	**38.2**	33.1
Birdsong detection	77.0	**80.0**	81.4
Music, speech & noise	99.1	**99.6**	-
Language Id.	71.3	**79.0**	86.0
Music instrument	63.4	**68.3**	72.0
Average	74.3	**79.8**	-

Multimodal

Audio SSL > Speech SSL for audio related tasks

TABLE V
FSD50K LINEAR EVALUATION PERFORMANCES.

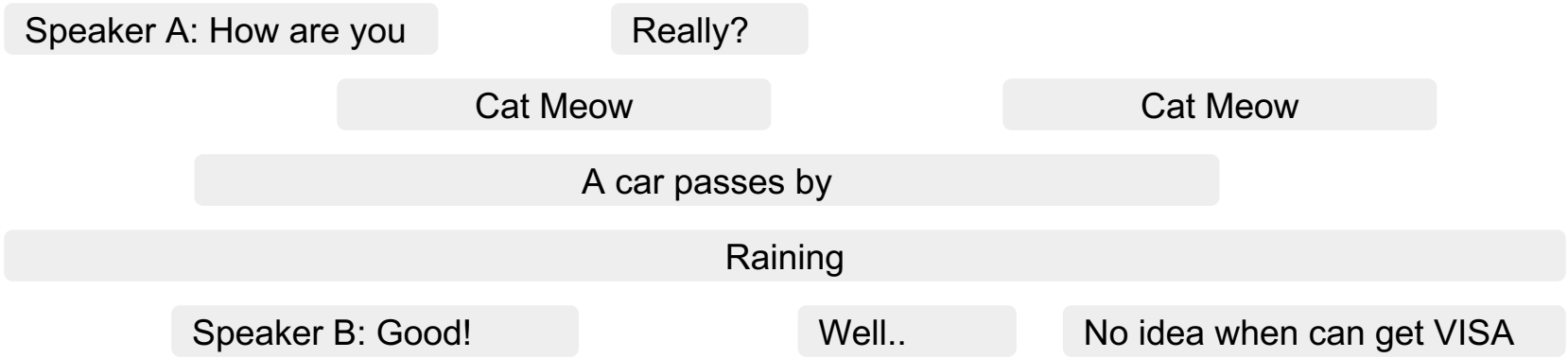
	All classes		Char. subsets (mAP)		
Representation	(mAP)	(AUC)	Single	Seq.	Scene
[Sas] PANNs *	0.494	0.904	0.466	0.424	0.511
[Sas] ESResNeXt *	0.524	0.911	0.490	0.443	0.611
[Sas] AST *	0.589	0.928	0.561	0.528	0.609
[S] VGGish	0.320	0.886	0.287	0.260	**0.513**
[S] VGGish-4K	0.352	0.835	0.342	0.267	0.445
[Ux] COALA	0.310	0.854	0.326	0.230	0.404
[Ux] OpenL3-E	0.420	0.879	0.433	0.305	0.486
[Ux] OpenL3-M	0.429	0.880	0.447	0.302	0.498
[U] TRILL	0.330	0.841	0.336	0.241	0.455
[U] Wav2Vec2-F	0.273	0.859	0.292	0.179	0.379
[U] Wav2Vec2-C	0.240	0.843	0.252	0.200	0.332
[U] BYOL-A	**0.448**	**0.896**	**0.473**	**0.313**	0.496

* Reference results for models pre-trained with AudioSet labels which is a super-set of FSD50K labels.

Current status

- No single model performs well for all sounds
- It is a problem clearly far from solved
  - Compared to SSL for more other speech tasks
  - Hence, might be important to keep S3PRL updated
- In the current universal sound recognition:
  - speech tasks are simple
  - sounds are usually independently classified

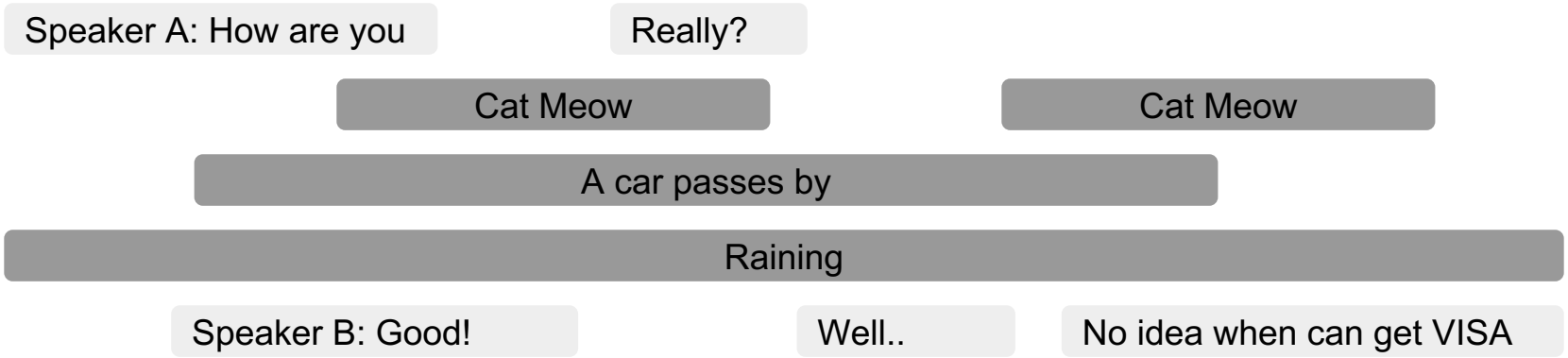
The ultimate goal of an universal sound recognizer



Can a single model recognize them all best?

Not exitsts yet (at least to my best knowledge)

For speech SSL, audio is usually considered as noises



But we as the human can recognize them all,

not just ignoring all the non-speech part

For speech SSL, audio is usually considered as noises

- So, can a model know all the sounds and when they are occuring?
- Speaker diarization: Who spoken when and be noise-robust
- How about:
  - What sound happens when and recognize both speech and audio
- When a hubert-msg being more noise-robust on speech, it might be more far away from an universal sound recognizer (or the opposite suprisingly)

Timeline

- **7/8 ~ 7/15:** Adding Audio pre-trained models to S3PRL
- **7/15 ~:** Exploring more details on the speech + audio task design
    - Might not finish in the workshop period

Evolution and major usage of S3PRL