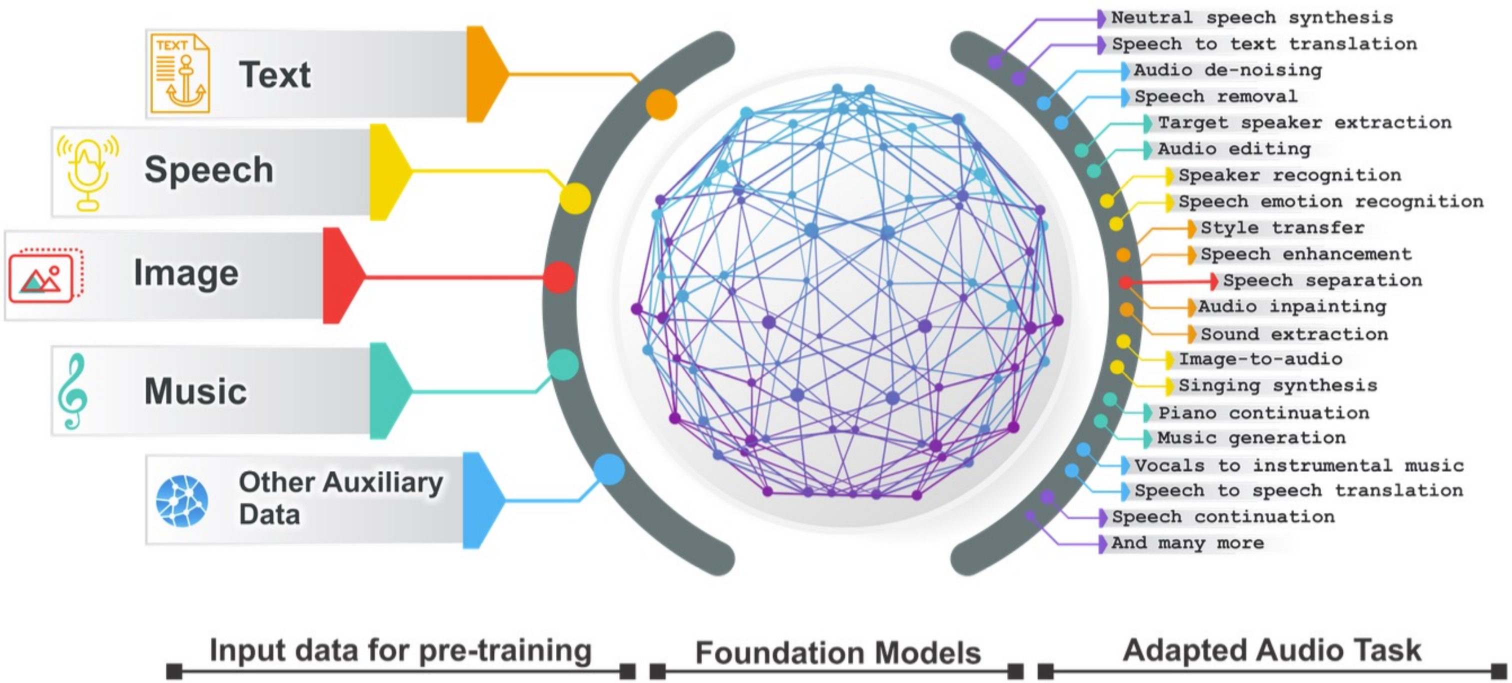
Bonus presentation on audio LLMs and BPE algorithms

AUDIO LLMs

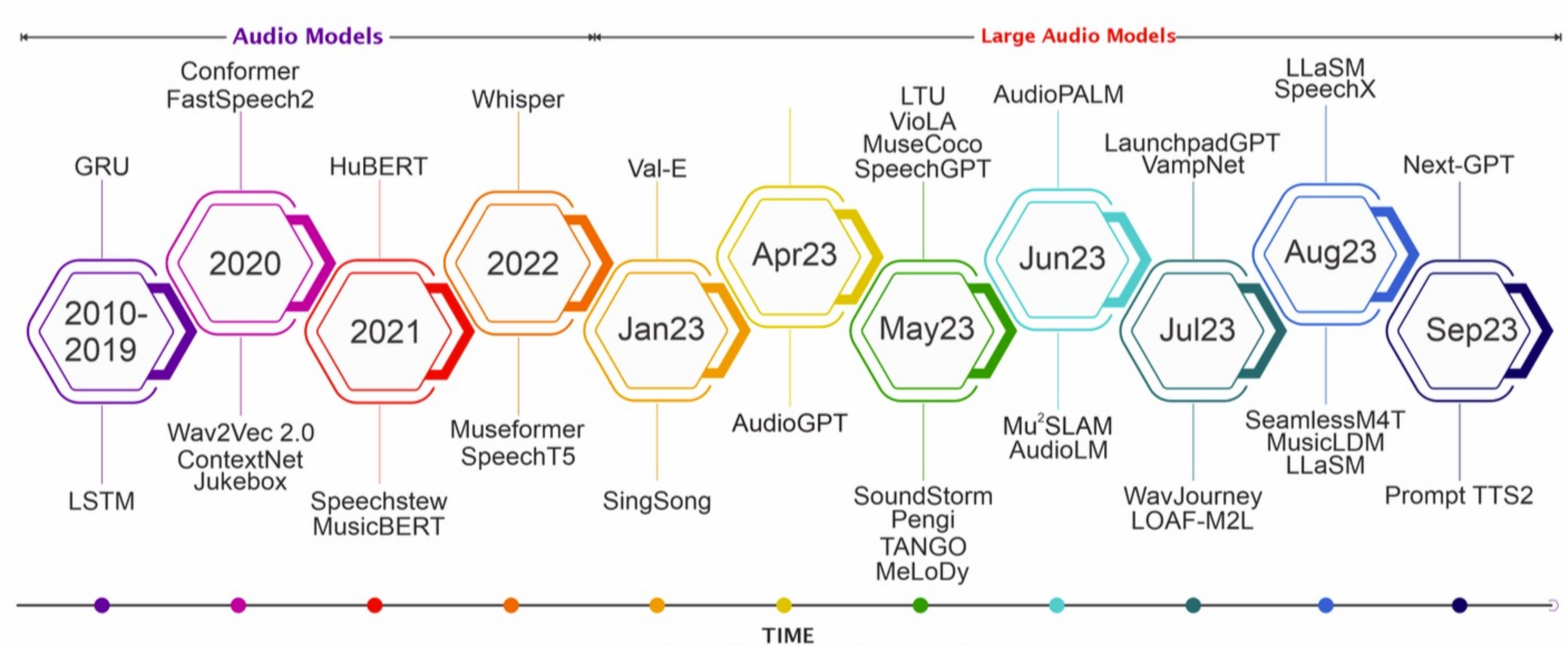
- Overview
- Time-line

Overview of Foundational Audio Models:
(Latif et al., 2023)



https://arxiv.org/abs/2308.12792

A fast-evolving world
(Latif et al., 2023)



Time line of Large Audio Models

https://arxiv.org/abs/2308.12792

Three main (sub)tokenisation algorithms

- Byte Pair Encoding (BPE) (GPTmodels, Whisper)
- WordPiece (BERT models)
- SentencePiece (neural machine translation)

byte pair encoding (BPE)

- We adapt *byte pair encoding* (BPE) (Gage,1994), a compression algorithm, to the task of word segmentation. BPE allows for the representation of an open vocabulary through a fixed-size vocabulary of variable-length character sequences, making it a very suitable word segmentation strategy for neural network models

Sennrich et al. **Neural Machine Translation of Rare Words with Subword Units**

The algorithm

- Firstly, we initialize the symbol vocabulary with the character vocabulary, and represent each word as a sequence of characters, plus a special end-of-word symbol '·', which allows us to restore the original tokenization after translation. We iteratively count all symbol pairs and replace each occurrence of the most frequent pair ('A', 'B') with a new symbol 'AB'. Each merge operation produces a new symbol which represents a character n-gram. Frequent character n-grams (or whole words) are eventually merged into a single symbol, thus BPE requires no shortlist. The final symbol vocabulary size is equal to the size of the initial vocabulary, plus the number of merge operations
- – the latter is the only hyperparameter of the algorithm.

- For efficiency, we do not consider pairs that cross word boundaries.

Sennrich et al. **Neural Machine Translation of Rare Words with Subword Units**

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

r ·	$\rightarrow$	r·
l o	$\rightarrow$	lo
lo w	$\rightarrow$	low
e r·	$\rightarrow$	er·

Figure 1: BPE merge operations learned from dictionary {'low', 'lowest', 'newer', 'wider'}.

Sennrich et al. **Neural Machine Translation of Rare Words with Subword Units**

Subtokenisation depends on the vocabulary size

system	sentence
source	health research institutes
reference	Gesundheitsforschungsinstitute
WDict	Forschungsinstitute
C2-50k	Fo|rs|ch|un|gs|in|st|it|ut|io|ne|n
BPE-60k	Gesundheits|forsch|ungsinstitu|ten
BPE-J90k	Gesundheits|forsch|ungsin|stitute
source	asinine situation
reference	dumme Situation
WDict	asinine situation → UNK → asinine
C2-50k	as|in|in|e situation → As|in|en|si|tu|at|io|n
BPE-60k	as|in|ine situation → A|in|line-|Situation
BPE-J90K	as|in|ine situation → As|in|in-|Situation

Table 4: English→German translation example. "\|" marks subword boundaries.

Byte Pair Encoding (BPE)

- Approach: Bottom-up, iteratively merging most frequent character pairs
- Merge criterion: Based on frequency of symbol pairs
- Used in: GPT models
- Tokenization: Places '@@' at the end of (sub)tokens
- Lossless: Fully lossless, preserving consecutive spaces

WordPiece

- Approach: Bottom-up, similar to BPE but with different selection criteria
- Merge criterion: Maximizes likelihood of training data
- Used in: BERT models
- Tokenization: Places '##' at the beginning of (sub)tokens
- Lossless: Lossy, does not preserve spaces

SentencePiece

- Approach: Configurable, can use BPE or Unigram algorithm
- Input: Works on raw, unsegmented text, including spaces
- Language agnostic: Suitable for languages without clear word boundaries
- Lossless: Partially lossless, preserves one space for multiple consecutive spaces
- Tokenization: Uses '▁' to represent spaces

Key Differences

- Merge criteria: BPE uses frequency, WordPiece uses likelihood, SentencePiece depends on the chosen algorithm
- Pre-tokenization: BPE and WordPiece require pre-tokenization, SentencePiece does not
- Language support: SentencePiece is more language-agnostic, making it suitable for multilingual models
- Complexity: BPE has low complexity, WordPiece medium, and SentencePiece medium to high
- Subword regularization: Only SentencePiece supports this feature

In practice, the choice between these tokenizers often depends on the specific task, language, and model architecture being used.