MASTER OF SCIENCE
IN ENGINEERING

Advanced Natural Language Processing: AdvNLP

Lesson 1: Introduction

Table of Contents

1. Organization and plan of the course
2. Learning approaches in NLP
3. The state of NLP today
4. NLP data and pre-processing
5. Tokenization methods

1. ORGANIZATION AND PLAN OF THE COURSE

MSE

Teaching Details

- Format:
  - 100% on-site with in-person attendance
  - additional passive streaming on Zoom
    (i.e. real-time watching and listening)
- Resources are available on Moodle
- Schedule: 14 weeks
  - lectures (13:10-13:55 + 14:05-14:50),
    then labs (15:00-15:45 or 15:55-16:40)
  - some labs will be interspersed with lectures
- Course grades
  - exam (two hours): 80%
  - four graded labs: 20%
    - Python/Jupyter, laptop/Colab, 2 people (!)

Teachers:

- Daniele Puccinelli

SUPSI, Manno (TI)
PhD Univ. of Notre-Dame, USA
www.supsi.ch/daniele-puccinelli

- Daniel Perruchoud

FHNW, Windisch (AG)
PhD ETHZ, Switzerland
www.fhnw.ch/en/people/daniel-perruchoud

Plan of the course

Date	Lecturer	Chapter
20.2.2025	Daniel Perruchoud	Introduction
27.2. / 6.3.2025	Daniele Puccinelli	Text classification and sentiment analysis (part 1 / part 2)
13.3.2025	Daniel Perruchoud	Non-contextual word vectors
20.3.2025	Daniele Puccinelli	Statistical models of word sequences
27.3.2025	Daniele Puccinelli	Human-computer interaction
3.4. / 10.4. 2025	Daniel Perruchoud	Language models (part 1 / part 2)
17.4. / 24.4. 2025	Daniele Puccinelli	Foundation models (part 1 / part 2)
8.5.2025	Daniel Perruchoud	Large Language Models (LLMs) in practice
15.5.2025	Daniele Puccinelli	Speech processing – Text-to-Speech
22.5.2025	Daniel Perruchoud	Speech processing – Speech-to-Text
5.6.2025	Daniel Perruchoud	Retrieval Augmented Generation (RAG)

MSE

Textbook

Dan Jurafsky and James H. Martin

Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

⇒ 2nd edition (printed), Prentice-Hall, 2008

⇒ 3rd edition (online draft), https://web.stanford.edu/~jurafsky/slp3/,

regularly updated as the domain evolves very quickly!

2. THE STATE OF NLP TODAY

What is Natural Language Processing?

- **Definition**: NLP focuses on developing systems that *understand and generate human languages*. It leverages computational methods to interpret, process, and produce natural language, *enabling interactions between humans and computers*.
- **Approach**: NLP employs *deep learning techniques* and deals with complexities like the representation of *language and speech*, capturing context, addressing biases.
- **Challenges**: Human language is *ambiguous* and *context-dependent*, making it challenging for machines to fully grasp meaning and intent. Ensuring NLP models do *not propagate existing biases* present in training data is an ongoing challenge.

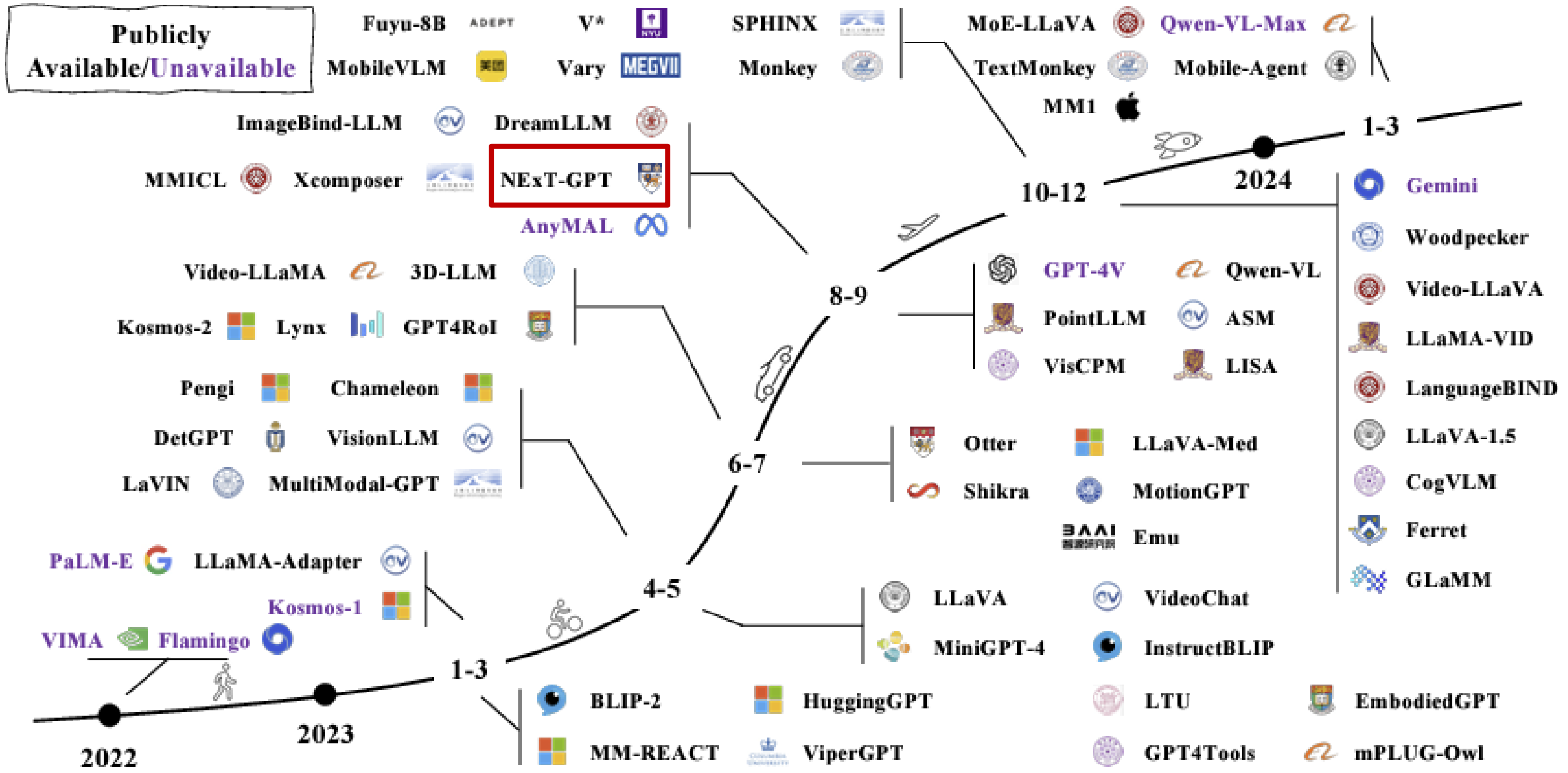
What are essential fields for modern NLP?

- **Machine Learning and Deep Learning**
  - Are essential for training neural network-based models on data
- **Software Engineering**
  - Drives algorithm development, data processing, and real-time deployment of NLP systems
- **Linear Algebra, Calculus, Statistics and Probability**
  - Are foundational to define learning objectives, loss functions and solve optimization methods
- **Linguistics**
  - Helps understanding language structure, syntax, semantics, and pragmatics
- **Cognitive Science**
  - Contributes insights into human language processing, aiding in NLU and NLG

How did NLP evolve?

- **1950s-1960s**: Foundations with rule-based approaches and the Turing Test. Early attempts at machine translation.
- **1970s-1980s**: Shift to symbolic methods with formal grammars. Rise of expert systems for language understanding.
- **1990s**: Introduction of statistical NLP using probabilistic models, n-grams, and Hidden Markov Models (HMMs).
- **2000s**: Dominance of Machine Learning with SVMs, decision trees, and feature-based models like CRFs.
- **2010s**: Rise of Deep Learning with LSTMs, CNNs, and then Transformer-based models like BERT and GPT (2018), revolutionizing NLP tasks.
- **2020s**: Emphasis on large language models (LLMs), contextual embeddings, and applications with LLMs in various industries.

How does NLP evolve now? (1/2)

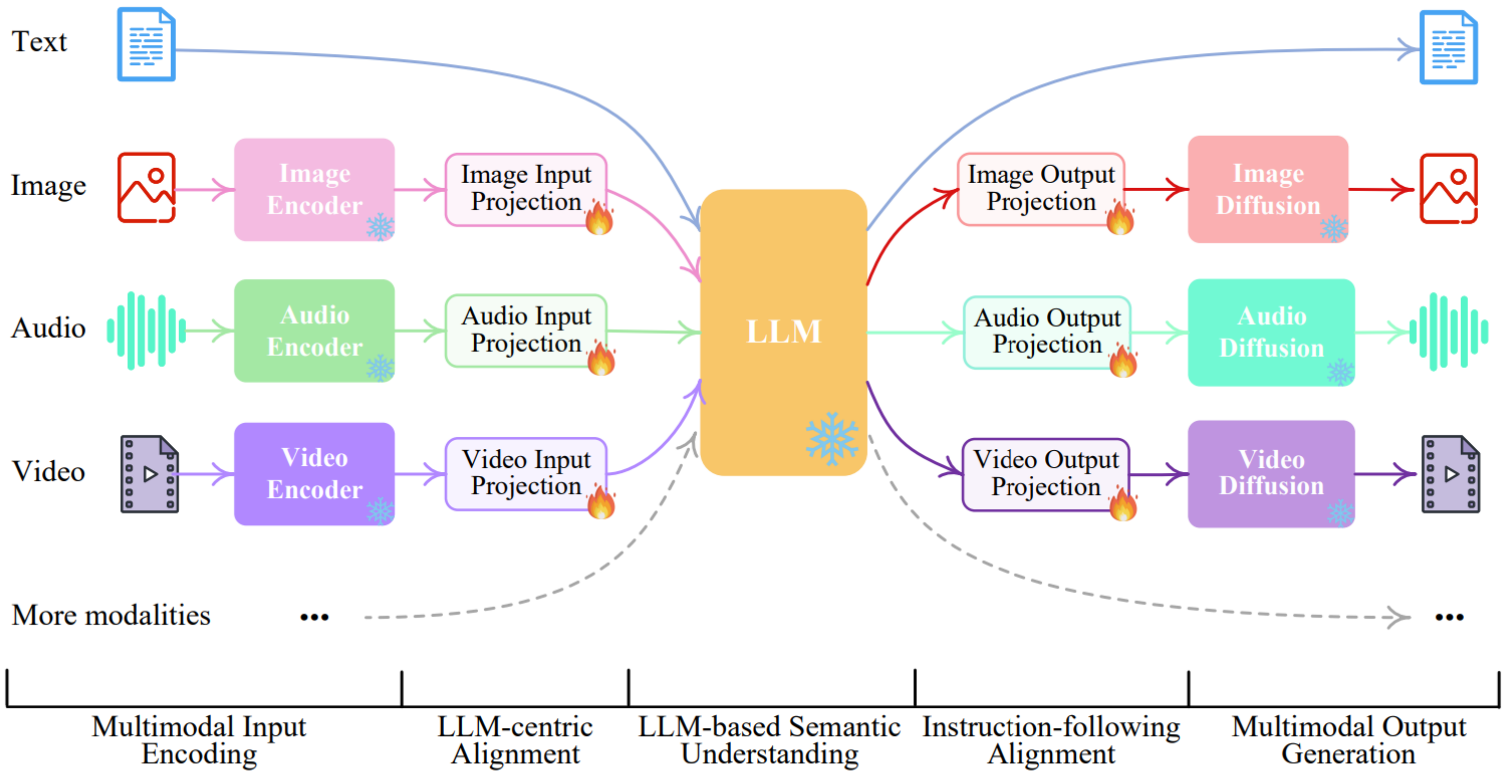


→ *Multimodal models* strive to enable a more comprehensive *understanding of diverse data modalities* including text, tables, images, audios etc.

Source: [Yin et al. (2024) A Survey on Multimodal Large Language Models]

How does NLP evolve now? (2/2)

- **2024**: Development of Multimodal LLMs, i.e., any-to-any systems with multimodal adaptors and modality-specific diffusion decoders



Source: [Wu S. et al. (2024) NExT-GPT: Any-to-Any Multimodal LLM]

What applications does NLP offer?

TEXT & SPEECH ANALYSIS

- Spelling & grammar check
- Document retrieval
- Text classification
- Information extraction
- Sentiment analysis
- Named-entity recognition
- Content-based recommendation
- Speech recognition

GENERATION

- Generate text
- Synthesize speech

ANALYSIS & GENERATION

- Machine translation
- Question answering
- Summarization

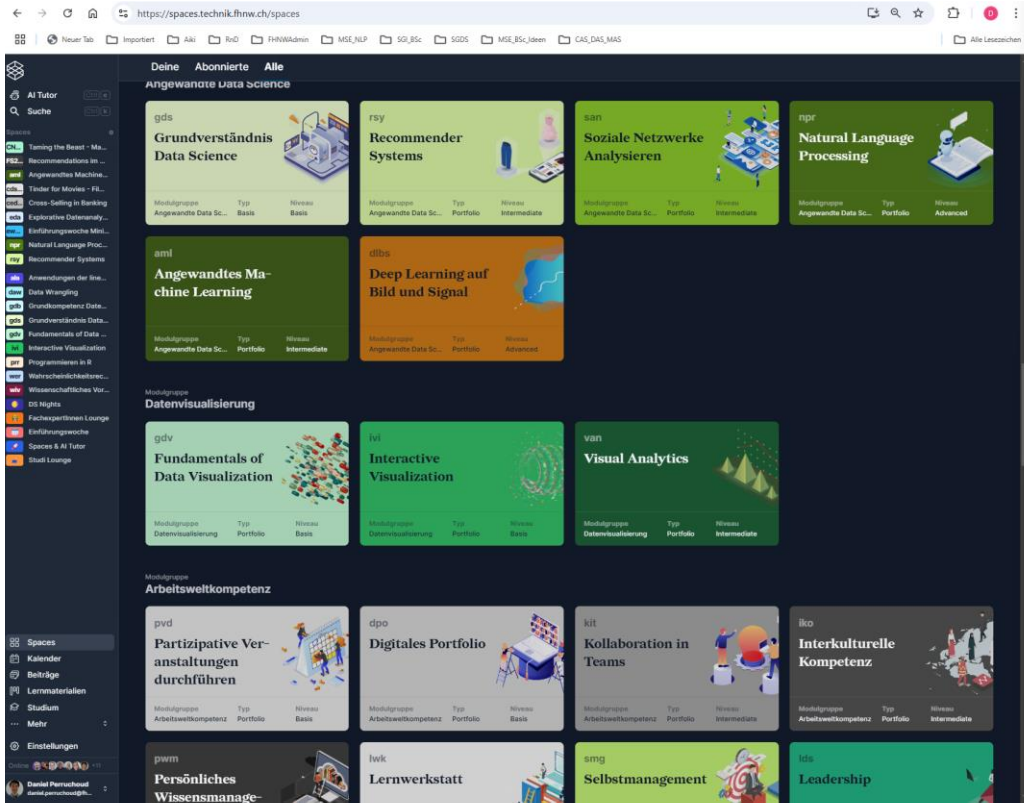
ANALYSIS & GENERATION & INTERACTION

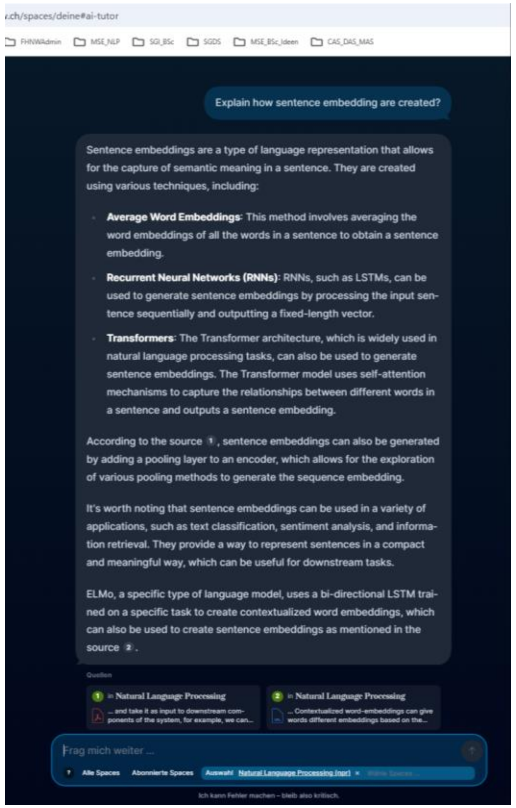
- Dialogue systems / chatbots

Modern NLP in Action

Knowledge Management with Retrieval Augmented Generation

– https://spaces.technik.fhnw.ch/spaces

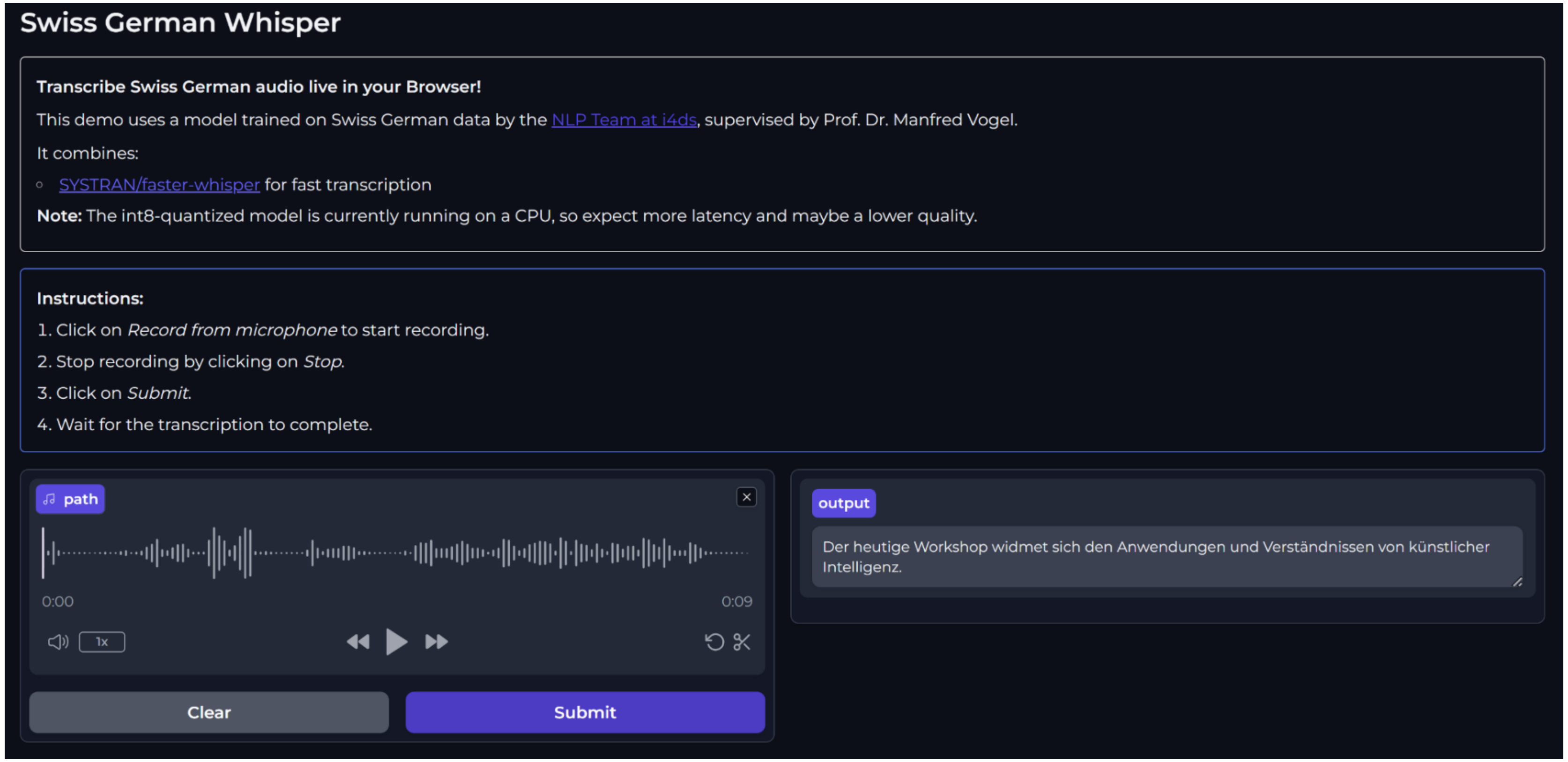




Modern NLP in Action (cont.)

Speech-to-Text transcription of low-resource languages

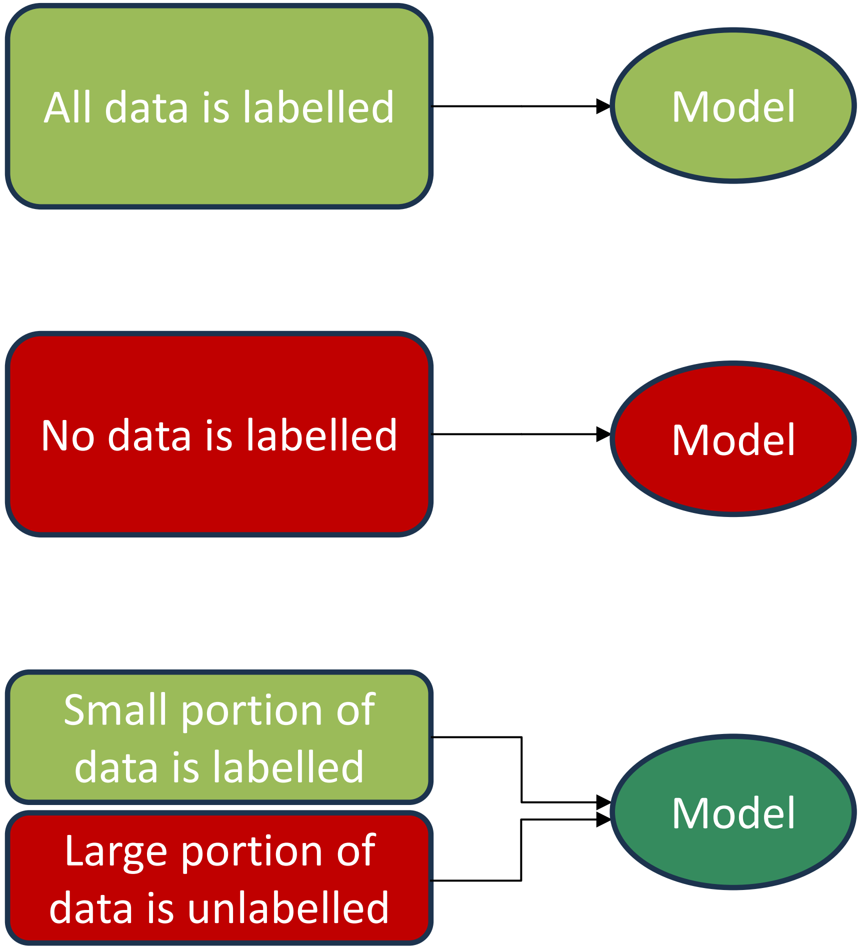
- [FHNW | Speech AI for Swiss German]



3. LEARNING APPROACHES IN NLP

Overview of learning approaches in NLP

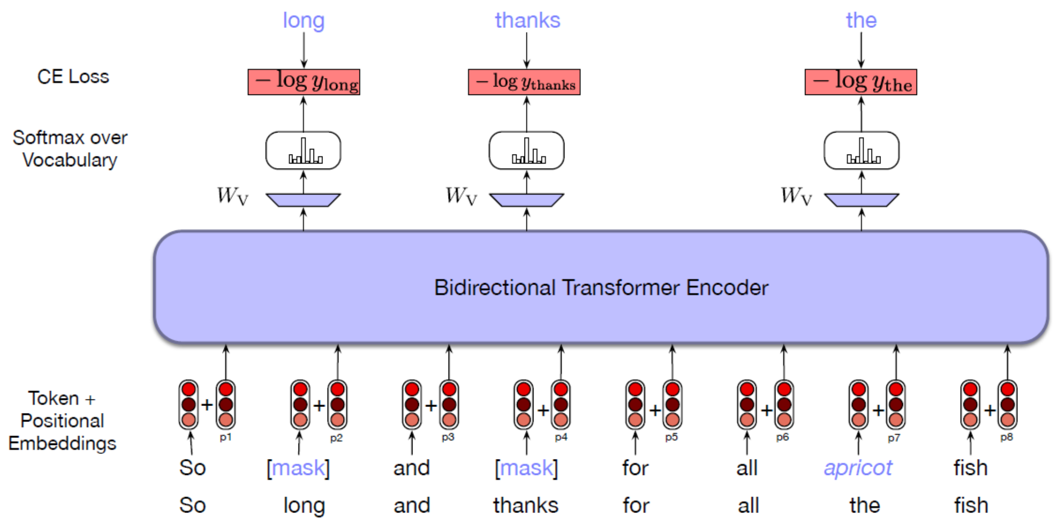
- Supervised learning
  - Learning relations from labeled input-output data-pairs
- Unsupervised / Self-supervised learning
  - Learning hidden patterns from unlabeled data / by creating its own labels from the input data itself
- Semi-supervised learning
  - Leveraging information from labeled data to annotate unlabeled data



Example of self-supervised learning

- Deep neural networks using Transformers
  - input = sequence of words from a text
  - output = one dense low-dimensional vector per word → contextualized word embeddings
- BERT: Pre-trained to guess masked tokens from input text
  - classic backpropagation with cost function:
    $-\log(P_{model}(token_{correct}))$
- “Self-supervised” because training data is simply raw text with 12% masked tokens

BERT (Transformer Encoder)

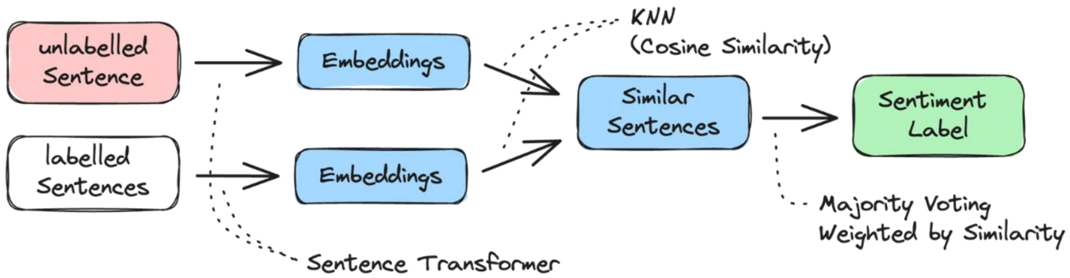


Source: Jurafsky & Martin, SLP3, Ch. 11, Fig. 5

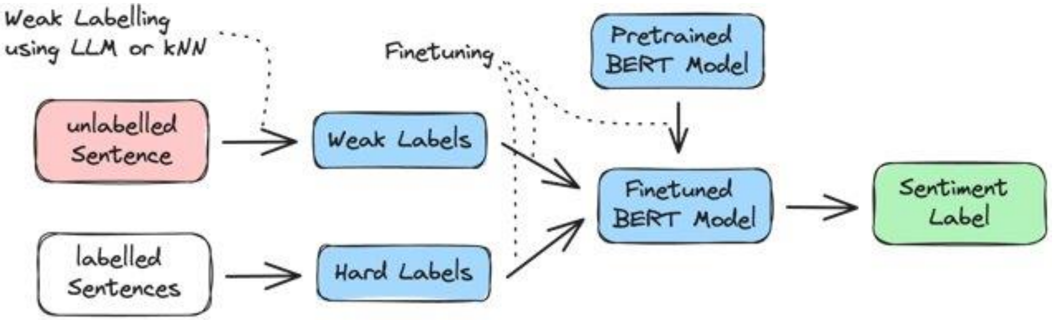
Source: Transformers: Vaswani et al. (2017); BERT pre-trained encoder: Devlin et al. (2018); GPT pre-trained decoder: Radford et al. (2018)

Combining learning approaches

- Semi-supervised learning of synthetic “weak" labels with pre-trained Sentence Transformer and kNN



- Fine-tuning BERT using hard labels and abundant, weak labels to change all model weights



Other variants of learning used in NLP

- Fine-tuning
  - further training a model on a smaller, domain-specific or task-specific dataset using self-supervised learning
- Parameter-efficient fine-tuning (PEFT)
  - fine-tuning large models by modifying only a small subset of parameters
- Preference alignment learning
  - training models to align their outputs with human preferences using human feedback using reinforcement or self-supervised learning
- In-context learning (ICL)
  - performing tasks based on examples in the input prompt, without parameter updates, i.e., no proper machine learning involved

Parameter Efficient Fine-Tuning (PEFT)

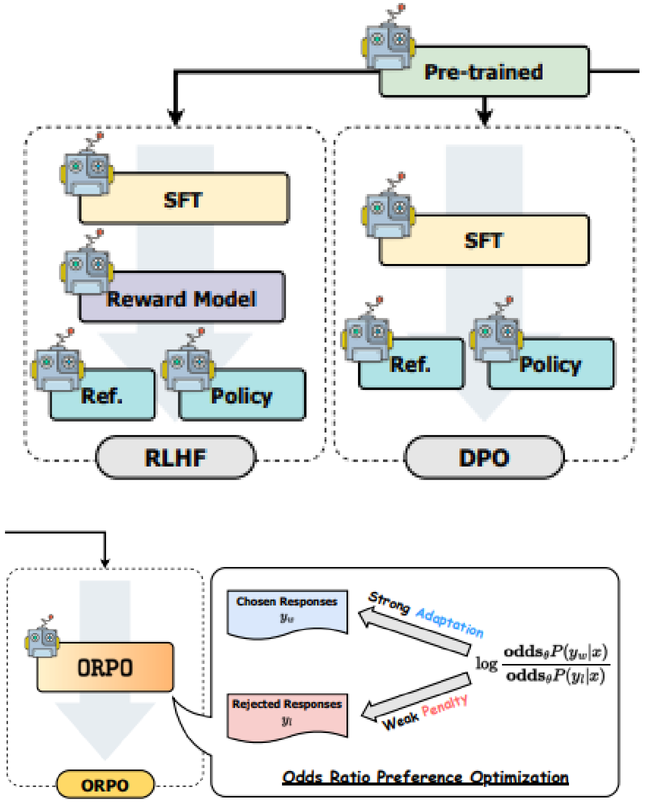
- Reduces inference time and energy consumption, while maintaining good performance

- Low-Rank Adaptation (LoRA)
  - freeze pretrained model weights and inject trainable rank decomposition matrices whose low-rank approximations retain the most important information
- Pruning
  - identify and eliminate redundant or less important parameters based on criteria, such as magnitude-based pruning or structured pruning
- Quantization
  - represent parameters with lower bit precision (e.g., 32-bit floats $\rightarrow$ 8-bit integers)
  - can be implemented during training or after training (post-quantization)

Preference Alignment Learning

- Refining a model's behavior to get it closer to human preferences requires model output samples ranked by humans (typically for safety and helpfulness)

- **Reinforcement Learning from Human Feedback (RLHF)**
  - trains an auxiliary model on human-provided evaluations
  - adjusts main model to maximize rewards based on this model
- **Direct Preference Optimization (DPO)**
  - directly optimizing the model's outputs against human preference data, without explicit reward models
- **Odds Ratio Preference Optimization (ORPO)**
  - fine-tuning language models without requiring a reference model or a separate preference alignment phase

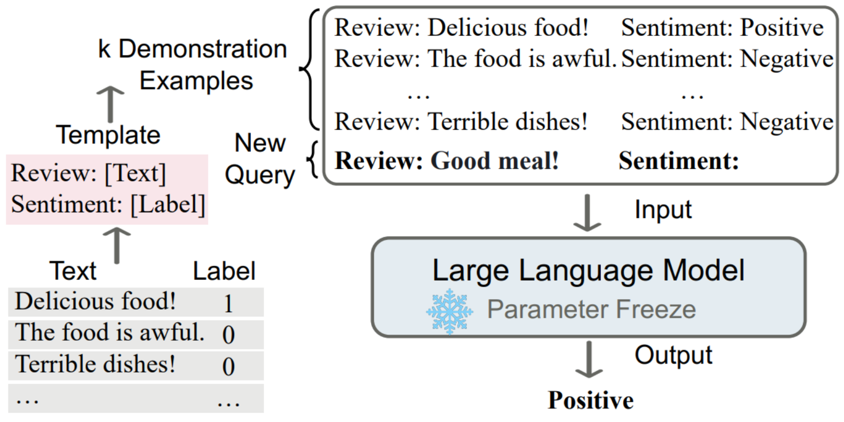


Source: Hong J. et al., ORPO: Monolithic Preference Optimization without Reference Model, 2024.

MSE

In-Context Learning (ICL)

- ICL uses trained LLMs to solve tasks without retraining or fine-tuning
- New tasks are solved by inference of prompted clear, well-structured, and relevant examples
- ICL works better in larger models which support generalization across various contexts
- ICL improves by exposure to diverse data
- ICL depends critically on prompting, but performance drops only marginally when labels in the examples are replaced by random ones



Source: Dong Q. et al., A Survey on In-context Learning, 2023. See also Min S. et al., Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?, 2022.

4. NLP DATA AND PRE-PROCESSING

Characteristics of NLP Data

- NLP models require extensive training $\rightarrow$ characteristics of training data play a crucial role for the model performance
- Training data requires careful selection criteria
  1. Diversity & Balance
     - varied contexts, languages, dialects, and styles for the desired application
     - avoiding overrepresentation of certain classes or topics, to prevent bias
  2. Quality & Quantity
     - accurate, consistently-annotated, and free of ambiguities, errors or biases
     - sufficient data volume to capture underlying patterns and structures
  3. Relevance & Recency
     - aligned with the target task or domain of the specific application
     - including up-to-date data and current knowledge to avoid obsolescence

Examples of benchmarks

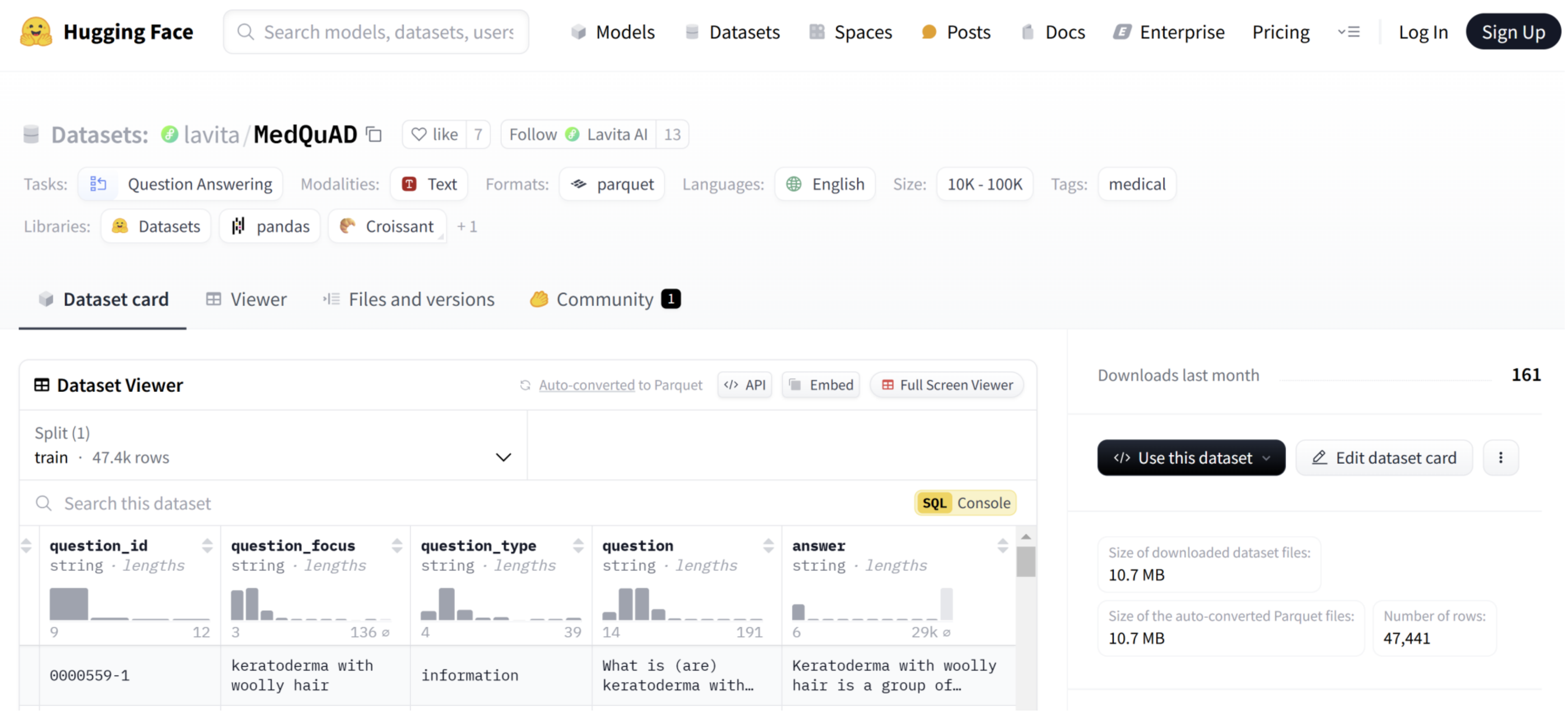
NLP training & test data is selected in relation to the task and application in scope:

GLUE	**General Language Understanding:** A collection of nine diverse natural language understanding tasks, including single-sentence tasks, similarity and paraphrasing tasks, and NLI tasks.
SNLI	**Stanford Natural Language Inference**: A collection of sentence pairs labeled for entailment, contradiction, and neutral relationships, used for training and evaluating NLI models.
SQuAD	**Stanford Question Answering Dataset:** A reading comprehension dataset consisting of questions posed on Wikipedia articles, where the answer to each question is a segment of text from the corresponding passage.
SCROLLS	**Standardized Comparison Over Long Language Sequences:** An NLP benchmark consisting of a suite of tasks that require reasoning over long texts, including summarization, question answering, and NLI across multiple domains.
CoNLL-2003	**CoNLL-2003: CoNLL-2003:** a named entity recognition (NER) dataset released as a part of CoNLL-2003 shared task consisting of eight files covering English and German

Source: https://paperswithcode.com/datasets?mod=texts&page=1

Example: search for data on Hugging Face

NLP data is selected in relation to the task and application in scope:



Source: https://huggingface.co/datasets/lavita/MedQuAD

Leaderboards, model comparison & benchmarks

Model selection via leaderboards needs careful application-specific assessment

Leaderboard for embedding models in French

Model size

Dimensionality

Context window size

Performance averaged over various datasets
(without standard deviation)

Specified metrics

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization

English | Chinese | French | Polish

Overall MTEB French leaderboard (F-MTEB) FR

- **Metric:** Various, refer to task tabs
- **Languages:** French
- **Credits:** Lyon-NLP: Gabriel Sequeira, Imene Kerboua, Wissam Siblini, Mathieu Ciancone, Marion Schaeffer

Rank	Model	Model Size (GB)	Embedding Dimensions	Max Tokens	Average (26 datasets)	Classification Average (6 datasets)	Clustering Average (7 datasets)	Pair Classification Average (2 datasets)
1	mistral-embed		1024		59.69	68.61	44.74	81.01
2	voyage-code-2		1536	16000	58.8	67.44	45.46	80.41
3	sentence-t5-xxl	9.73	768	512	58.04	67.36	43.25	81.99
4	Solon-embeddings-large-0.1	2.24	1024	514	58.03	70.02	39.65	77.1
5	sentence-camembert-large	1.35	1024	512	56.88	65.82	43.17	79.8
6	multilingual-e5-base	1.11	768	514	56.47	66.8	42.66	78.46
7	multilingual-e5-large	2.24	1024	514	56.3	68.39	38.7	79.25

→ **CAUTION**:
data leakage or overfitting may bias leaderboard evaluation!

Source: https://huggingface.co/blog/lyon-nlp-group/mteb-leaderboard-best-practices

5. TOKENIZATION METHODS

Words in neural networks

- Input a word into a neural network = activate a specific input unit = one-hot vector
  - therefore, texts need to be fragmented into words before processing by a neural network
- The input layer cannot be extended excessively: $10^4 - 10^6$ words from training data
- Therefore, many unknown words appear during testing (out of vocabulary, OOV)
  - although the number of lemmas in a language is around 10^5, the number of forms (conjugated verbs, plurals, etc.) is $10 - 100 \times$ larger*
  - Also, numbers, proper names (people, places, companies), etc.
  - novel words created by borrowing, compounding**, derivation
  - people make spelling mistakes or use straaaange spellings on purpose

* This depends on the language : e.g. EN < FR < RU

** E.g., Donaudampfschifffahrts-gesellschaftskapitän

- Possible but non-optimal solutions
  - ignore unknown words; check spelling; normalize words; model numbers with rules; decompose words into root and affixes; use character-based models or hybrid ones (decompose unknown words into characters)

Word- / character- /subword-based tokenization

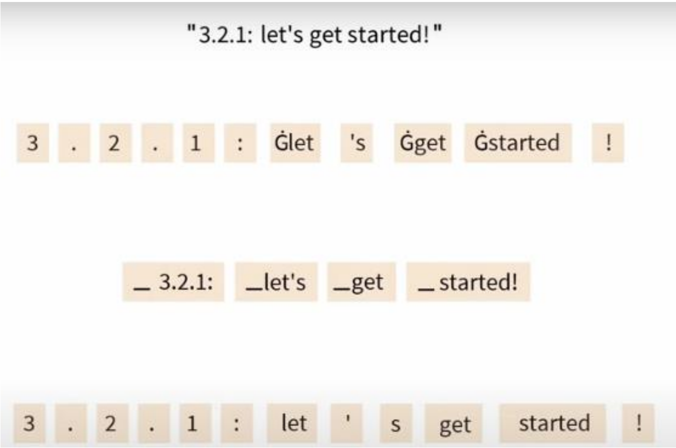
- Word-based: split text into individual words
  - simple and intuitive: use white spaces, deal with punctuation
  - limitations: need to handle many exceptions; resulting vocabulary size ↑↑; OOVs; no modeling of relationships between words (‘dog’ $\neq$ ‘dogs’)
- Character-based: split text into individual characters
  - very simple: no need for specific rules
  - advantage: can handle any word, no OOVs (but possibly unseen characters)
  - limitations: long sequences, much more difficult to learn, hard to capture semantic information
- Subword-based:
  vocabulary made of words, word parts, and characters
  $\Rightarrow$ learned from the training data!

```
['There', 'are', 'eight', 'orbit', '##ers', 'surveying', 'the', 'planet', ':', 'Mars', 'Odyss
ey', ',', 'Mars', 'Express', ',', 'Mars', 'Reconnaissance', 'Or', '##bit', '##er', ',', 'Mars
', 'Or', '##bit', '##er', 'Mission', ',', 'MA', '##VE', '##N', ',', 'the', 'Trace', 'Gas', 'O
r', '##bit', '##er', ',', 'the', 'T', '##ian', '##wen', '-', '1', 'orbit', '##er', ',', 'and
', 'the', 'Hope', 'Mars', 'Mission']
```

Byte-Pair Encoding (BPE): preprocessing

- Byte-Pair Encoding requires input text must be preprocessed for
  - Training, i.e., to build the vocabulary from the training data
  - Testing, i.e., to tokenize any new text using this vocabulary

- Preprocessing includes
  - clean-up newlines and multiple whitespaces, possibly lowercase
  - tokenize into words and punctuations based on whitespaces and/or model-specific rules
  - mark word endings with a specific symbol, e.g. </w> or word continuations, e.g. ## or __ (model-specific rules)
  - split text into individual characters, remove infrequent ones



Source: https://huggingface.co/learn/nlp-course/chapter6/4?fw=pt

Source: Sennrich, R. et al., "Neural Machine Translation of Rare Words with Subword Units", *Proc. of ACL 2016*

Byte-Pair Encoding (BPE): training

- Build the vocabulary through successive applications of “merge” operations
  - training corpus: low (5 times), lower (2 times), newest (6 times), widest (3 times)
  - the initial vocabulary, after pre-processing: `l o w </w> e r n s t i d`
- 1st step: what is the most frequent character bigram?
  - `e+s` occurs 9 times

    → add new symbol ‘`es`’ (merged from `e+s`) to vocabulary

    → resulting vocabulary: `l o w </w> e r n s t i d es`
- 2nd step: what is the most frequent bi-gram of symbols?
  - `es+t` occurs 9 times (note that ‘es’ now participates in the counts)

    → add new symbol ‘`est`’ to vocabulary (still keeping ‘e’, ‘s’ and ‘es’)

    → resulting vocabulary: `l o w </w> e r n s t i d es est`
- After 5 steps, vocabulary is: `l o w </w> e r n s t i d es est est</w> lo low`

See also https://huggingface.co/transformers/tokenizer_summary.html#byte-pair-encoding-bpe

Byte-Pair Encoding (BPE): results & testing

- Result of training consists of the vocabulary and a set of “merge” rules
  - vocabulary contains
    - all characters (10^2 – 10^4), many frequent words (often >60% of all items), frequent character n-grams: some are meaningful (e.g. prefixes or suffixes), but many are not
  - vocabulary size = number of initial characters + number of merges – *X*
    - often between 10k and 32k, more for CJK or large models (up to 256k, [Tao et al. 2024](#))
    - *X*: some subwords are removed if they occur only as parts of larger subwords
- Testing mode: tokenize new text
  - preprocess, including splitting into characters
  - apply the merge operations learned, in the same order
    - no new counts are needed
  - example (unseen): `lowest</w>` → `low` `est</w>`

e s	→	**es**
es t	→	est
est </w>	→	est</w>
l o	→	lo
lo w	→	low

WordPiece: training

- WordPiece builds the vocabulary similarly to BPE through successive applications of ‘merge’ operations starting from a character-based vocabulary
  - Start with a vocabulary of individual characters from the training corpus, adding [UNK] for unknown tokens and the ## prefix for subwords that don’t start a word
  - Split training corpus into words and create all possible subword combinations for each word
  - Count all subword frequencies in the training data (track appearance at start/middle/end)
  - Evaluate possible new word units, i.e. candidate merges $x\ y$ using a scoring function:

    $P(x\ y)\ /\ (P(x)\ P(y))$ *(rationale = proxy for likelihood improvement)*
  - Generate new word unit with the highest score by merging and add it to vocabulary. If it’s not a word-initial subword, add it with the ## prefix, then update frequencies of affected tokens
  - Stop when desired vocabulary size is reached, or highest merge score falls below a threshold

Source: Schuster, M. & Nakajima, K., “Japanese and Korean voice search”, *2012,* see also WordPiece Tokenization: A BPE Variant | by Atharv Yeolekar | Medium

WordPiece: results & testing

- Result of training consists of the vocabulary
  - vocabulary contains
    - all characters, many frequent words, subwords (e.g. ‘quickness’ = ‘qu’ + ‘##ick’ + ‘##ness’), and the UNK symbol for completely new words
  - vocabulary size
    - size varies between 30k (cf. Devlin et al. (2018)) and up to 110k for multilingual BERT (cf. Pires et al. (2018) & HF)
- Testing mode: tokenize new text
  - Preprocess, including splitting into words
  - For each word, match the longest subword from the vocabulary at the start of the word.
  - If a match is found, add it to the output and repeat from the next character, for matches after the first character, use the ## version of the subword.
  - If no match is found, add [UNK] token and move to the next character.

MSE

Comparison of subword tokenization methods

	BPE	WordPiece	UnigramLM
Training	Starts from a *small* vocabulary and learns rules to *merge* tokens	Starts from a *small* vocabulary and learns rules to *merge* tokens	Starts from a *large* vocabulary and learns rules to *remove* tokens
Training steps	Merges the tokens corresponding to the most *frequent* pair	Merges the tokens corresponding to the pair with *the best score* (frequency of the pair, privileging pairs where each individual token is less frequent)	Remove *n%* tokens corresponding to the lowest reduction in likelihood upon removal computed on the whole corpus
Result of training	"Merge" rules and a vocabulary	Just a vocabulary	Vocabulary and score for each token
Encoding a new text	Splits words into characters and applies the merges learned during training	Finds the longest subword (starting from the beginning) that is in the vocabulary, then does the same for the remainder of the word	Finds the most likely split into tokens, using the scores learned during training

Adapted from: https://huggingface.co/learn/nlp-course/chapter6/4?fw=pt, see also Bostrom, K. & Durrett G. , "Byte Pair Encoding is Suboptimal for Language Model Pretraining" *2020*

SentencePiece: training

- SentencePiece directly operates on raw text without pre-tokenization/pre-splitting,
  - Text is considered as a sequence of Unicode characters
  - Whitespace is replaced with the special character "▁" (U+2581)
  - Text is normalized via Unicode NFKC normalization (default) or custom normalization rules
- Token vocabulary of predefined size is created by BPE or UnigramLM on raw text
  - UnigramLM starts with a large set of candidate subwords and prunes them probabilistically using a likelihood model
- Approach
  - Does not require language-specific pre-tokenization
  - Is readily applicable for languages w/o whitespace-separators (e.g., Chinese or Japanese)
  - Improves NMT accuracy and robustness via **subword regularization** using multiple subword segmentations, e.g., "internationalization" → [internation, alization] or [international, ization]

Source: Kudo. T., & Richardson, J.., "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing", *2018*
Kudo, T.. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates", *2018*

SentencePiece: results & testing

- Result of training is a fully self-contained model with no external dependencies
  - pre-compiled finite state transducer for character normalization
  - vocabulary and segmentation parameters contains
    - all characters, many frequent words, subwords, UNK for completely new words
  - vocabulary size
    - is around 32k for T5 (cf. Raffel et al. (2019)) & Llama (cf. Touvron et al. (2023))
- Testing mode: tokenize new text
  - use SentencePiece for tokenization offline or integrate it into the pipeline (see below)
  - segmentation speed of SentencePiece is fast enough for on-the-fly execution (around 10^4—10^5 sentences/sec)

Source: Kudo. T., & Richardson, J.., "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing", *2018*
Kudo, T.. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates", *2018*

Conclusion

- Subword tokenization
  - splits text into subword units and reduces vocabulary size compared to word-based tokenization
  - effectively handles out-of-vocabulary words and captures certain semantic relationships between words and their variations
  - is more complex to implement & produces longer sequences than word-based tokenization

```
]: tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-cased')
```

```
]: print(tokenizer.tokenize('''There are eight orbiters surveying
   the planet: Mars Odyssey, Mars Express, Mars Reconnaissance
   Orbiter, Mars Orbiter Mission, MAVEN, the Trace Gas Orbiter,
   the Tianwen-1 orbiter, and the Hope Mars Mission'''))
```

```
['There', 'are', 'eight', 'orbit', '##ers', 'surveying', 'the', 'planet', ':', 'Mars', 'Odyss
ey', ',', 'Mars', 'Express', ',', 'Mars', 'Reconnaissance', 'Or', '##bit', '##er', ',', 'Mars
', 'Or', '##bit', '##er', 'Mission', ',', 'MA', '##VE', '##N', ',', 'the', 'Trace', 'Gas', 'O
r', '##bit', '##er', ',', 'the', 'T', '##ian', '##wen', '-', '1', 'orbit', '##er', ',', 'and
', 'the', 'Hope', 'Mars', 'Mission']
```