EMNLP2026 BUDAPEST

EPFL UNIVERSITY OF CAMBRIDGE ETH zürich

LangMAP: A Language-Adaptive Approach to Tokenization

Clara Meister[1], Suchir Salhan[2], Andrzej Szablewksi[2], Pietro Lesci[2], Paula Buttery[2] & Tiago Pimentel[3]

[1]EPFL [2]University of Cambridge [3]ETH Zurich

Slides prepared by Suchir Salhan and Clara Meister.

Motivation

Tokenization: From Strings to Tokens

Tokenizer = map from strings to token sequences

Under the same fixed vocabulary $\mathcal{V}$, the same string usually has several valid segmentations.

$\mathcal{V}$ = { R, a, t, h, u, s, Ra, ha, us, at, aus, Rat, haus, Rathaus, … }

Tokenization: From Strings to Tokens

Tokenizer = map from strings to token sequences

Under the same fixed vocabulary $\mathcal{V}$, the same string usually has several valid segmentations.

$\mathcal{V}$ = { R, a, t, h, u, s, Ra, ha, us, at, aus, Rat, haus, Rathaus, … }

⟨ Rathaus ⟩	⟨ Rat, haus ⟩	⟨ Ra, th, aus ⟩
1 token	2 tokens	3 tokens

Tokenization: From Strings to Tokens

Tokenizer = map from strings to token sequences

Under the same fixed vocabulary $\mathcal{V}$, the same string usually has several valid segmentations.

$\mathcal{V}$ = { R, a, t, h, u, s, Ra, ha, us, at, aus, Rat, haus, Rathaus, ... }

⟨ Rathaus ⟩	⟨ Rat, haus ⟩	⟨ Ra, th, aus ⟩
1 token	2 tokens	3 tokens

Every segmentation concatenates back to "Rathaus"!

A tokenizer has a fixed rule that chooses one of these segmentations.
Different methods = different rules: BPE applies a learned merge order, WordPiece takes the longest matching token, **UnigramLM** uses token probabilities...

PROBLEM

Tokenizers are not optimised for most languages



Oversegmentation → longer token sequences



Poor token boundaries can obscure meaningful morphological structure. Inhibiting downstream performance



Higher per-request costs for APIs

Tuned for high-resource languages. Applied without adjustment to low-resource and dialects.

LangMAP

Tokenization: From Strings to Tokens

Tokenizer = map from strings to token sequences

Under the same fixed vocabulary $\mathcal{V}$, the same string usually has several valid segmentations.

$\mathcal{V}$ = { R, a, t, h, u, s, Ra, ha, us, at, aus, Rat, haus, Rathaus, ... }

⟨ Rathaus ⟩	⟨ Rat, haus ⟩	⟨ Ra, th, aus ⟩
1 token	2 tokens	3 tokens

Every segmentation concatenates back to "Rathaus"!

A tokenizer has a fixed rule that chooses one of these segmentations.
Different methods = different rules: BPE applies a learned merge order, WordPiece takes the longest matching token, **UnigramLM** uses token probabilities...

Our Paper

Sometimes the tokenizer has all the right tokens — it just chooses the wrong ones.

Can we make a pretrained tokenizer more language-adaptive simply by changing how we score existing tokens, rather than changing the vocabulary itself?

Introducing LangMAP



Most modern multilingual models have enormous vocabularies that contain pieces from a mixture of languages.

Tokenizers typically apply a single global segmentation policy.

Changing vocabulary is expensive, but changing a segmentation policy is cheap!

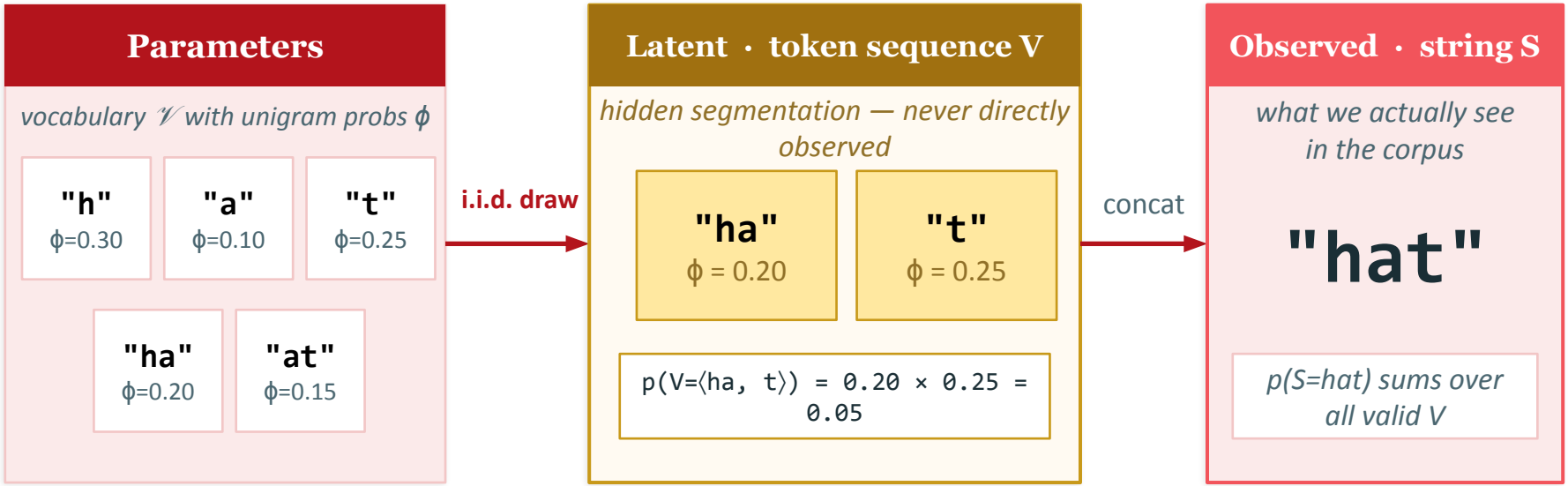
Our Paper

Sometimes the tokenizer has all the right tokens — it just chooses the wrong ones.

LangMAP adapts a pretrained tokenizer to a new language simply by changing how we score existing tokens, rather than changing the vocabulary itself!

UnigramLM: The Generative Model

Strings arise from a latent token sequence drawn i.i.d. from $\mathcal{V}$ (never observed directly)



Why this matters: V is latent. We never observe which segmentation produced S. This is why learning ϕ requires EM and inference needs the posterior p(V | S).

Multiple Valid Segmentations

The string "hat" has three valid segmentations under this $\mathcal{V}$.
The posterior probability distribution ranks them:

⟨ ha, t ⟩

48.8%

posterior probability

★ Viterbi choice

⟨ h, at ⟩

43.9%

posterior probability

⟨ h, a, t ⟩

7.3%

posterior probability

Inference picks the highest-posterior segmentation via Viterbi (lowest-cost path). Can be done with an efficient DP

$$\tau(s; \phi) = \text{argmax } p(v \mid s; \phi)$$

LangMAP

Unigram LM's Global allocation of probability mass over *V* is not representative of per language distributions

LangMAP takes shared vocabulary, and learns one unigram distribution per language → language-specific segmentations without inference-time language labels

LangMAP: Language-adaptive Maximum a Posteriori Tokenization

1. Keep base model + V
2. Learn language-specific token preferences
3. Change inference-time segmentation

From Tokenization to LangMAP

Fix shared vocabulary $\mathcal{V}$ →

Shared Vocabulary $\mathcal{V}$ (e.g., 100k byte-level tokens)

The same vocabulary is used for every language. LangMAP does not add or remove tokens.

The problem is that having a token in the vocabulary does not guarantee that the tokenizer will prefer it. That preference depends on the probability assigned to each token.

From Tokenization to LangMAP

Fix shared vocabulary $\mathcal{V}$ →

Shared Vocabulary $\mathcal{V}$ (e.g., 100k byte-level tokens)

▼

German Corpus C^{de}

```
"Das Rathaus ist geschlossen"
"Wir treffen uns am Rathaus"
"Ein Haus mit Garten"
```

English Corpus C^{en}

```
"The rat ran across the yard"
"Meet us at the town hall"
"He sat in the house"
```

For each language ℓ, we have a language-specific corpus C_ℓ and learn a corresponding probability distribution ϕ_ℓ

From Tokenization to LangMAP

Fix shared vocabulary $\mathcal{V}$ →

Shared Vocabulary $\mathcal{V}$ (e.g., 100k byte-level tokens)

▼

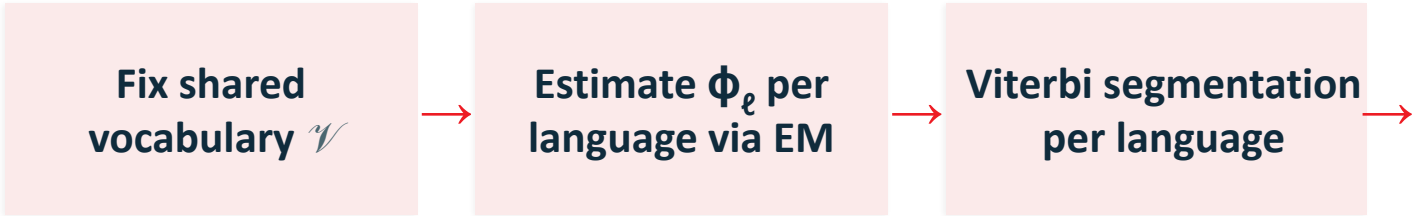
Each language gets its own token frequency profile $\hat{\phi}_\ell$ → each language gets its own segmentations

Because tokens that are common or useful in one language may be rare or less useful in another!

English	**English corpus C_{en}** Learn English token probabilities ϕ_{en}
French	**French corpus C_{fr}** Learn French token probabilities ϕ_{fr}
Spanish	**Spanish corpus C_{es}** Learn Spanish token probabilities ϕ_{es}

All three distributions use the same vocabulary V. Only their token probabilities differ.

From Tokenization to LangMAP



Parameter Estimation (EM)

Iterate until ~convergence:

E-step: compute expected token counts via forward–backward over all segmentations

M-step: normalize counts ⟶ new ϕ

Vocabulary pruning: remove tokens that hurt likelihood least

Learning Per-Language Distributions

Shared Vocabulary $\mathcal{V}$ (e.g., 100k byte-level tokens)

German Corpus C^{de}

"Das Rathaus ist geschlossen"

"Wir treffen uns am Rathaus"

"Ein Haus mit Garten"

EM (20 rounds) → ϕ_{de}

Rathaus 0.0054 **haus** 0.0018

der 0.0231 **ein** 0.0142

English Corpus C^{en}

"The rat ran across the yard"

"Meet us at the town hall"

"He sat in the house"

EM (20 rounds) → ϕ_{en}

Rathaus ≈ 0 **haus** 0.0003

th 0.0198 **us** 0.0089

Same vocabulary, same EM algorithm — but different corpora → different distributions
→ different optimal segmentations

Language-Adaptive Inference

Learning language-specific distributions is not enough!

At inference time, LangMAP decides which distribution to use for a new input string.

Find best segmentation under each language

$$v_\ell = \arg\max_{v \in T_V(s)} P(v; \phi_\ell)$$

Compare Candidates across languages

$$v^* = \arg\max_{v_\ell : \ell \in \Omega} P(v_\ell; \phi_\ell)$$

Language-Specific Segmentation

Input string: "Rathaus"

German ϕ_{de}

`Rathaus`

→ Single high-probability token
log p = −3.2
✓ **Single frequent token in German**

English ϕ_{en}

`Ra` `th` `aus`

→ Fragmented: unfamiliar word
log p = −8.7
✗ **Low probability → not this language…**

Different ϕ_ℓ → different segmentations → different likelihoods → signal for string's language

Experiments & Results

EXPERIMENTS

Experimental Setup

Natural Languages

7 languages: German, Finnish, Hungarian, Turkish, Hindi, Arabic, and Tamil

Code Languages

9 languages: Python, Java, JavaScript, C, C++, and TypeScript

EXPERIMENTS

Experimental Setup

Model and Setup

Experimental setup: Apply LangMAP to model vocabularies, estimating language-specific token probabilities from 25M tokens over 10 EM iterations; compare against the original tokenizer, language-agnostic UnigramLM.

- **Base Models:** Qwen2.5-0.5B, Llama-3.2-1B; Qwen, Llama, Gemma, Mistral tokenizers.
- **Method:**
  - Train LangMAP tokenizers (trained on 25M language tokens), compare against original base model tokenizer and language-agnostic UnigramLM.
  - Finetune models with LangMAP v base tokenizer.

EXPERIMENTS

Experimental Setup

Metrics

- **MorphScore:** compares token boundaries with gold-standard morphological boundaries
- **Code:** Alignment with Code Boundaries (e.g., abstract syntax tree leaf boundaries and identifier fragmentation

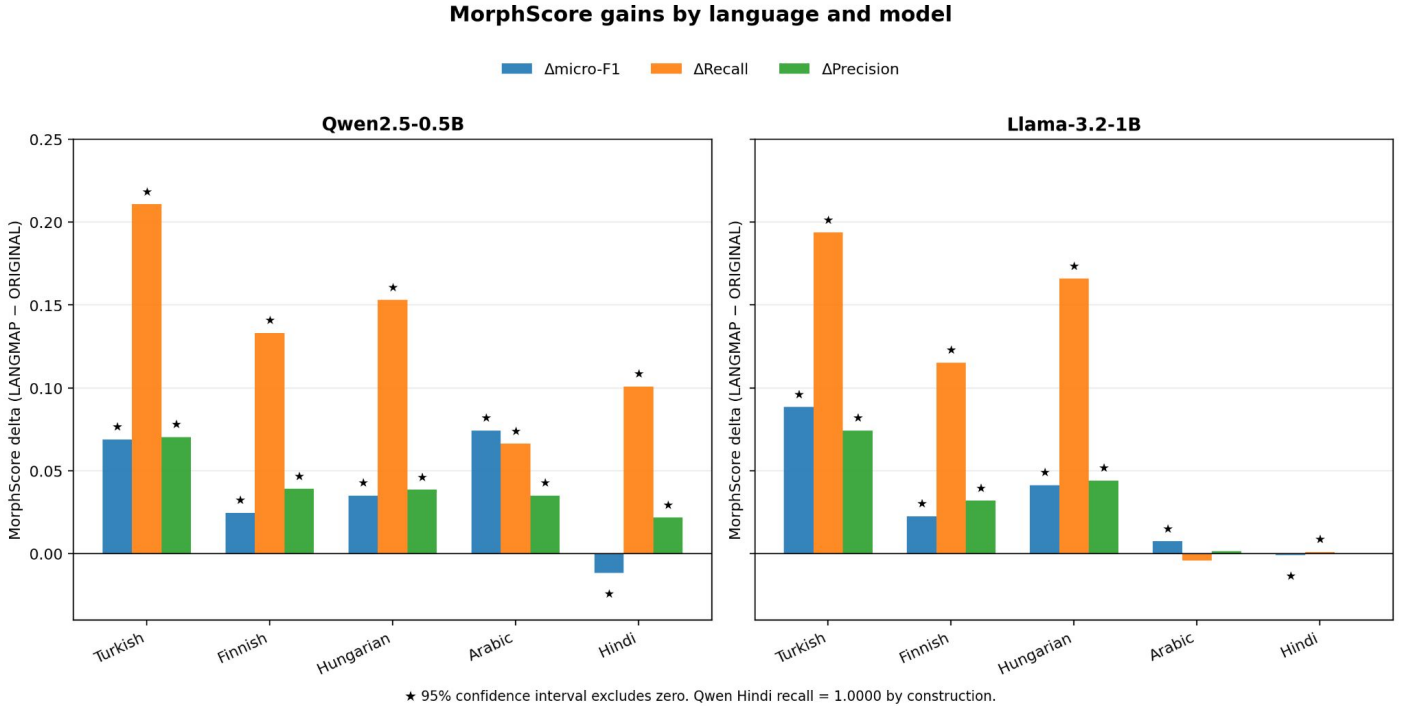
RESULTS

Morphological Segmentation

MorphScore

LangMAP improves morphological alignment.

Δ of LangMAP F1/Recall/Precision of LangMAP compared to Base Tokenizer.

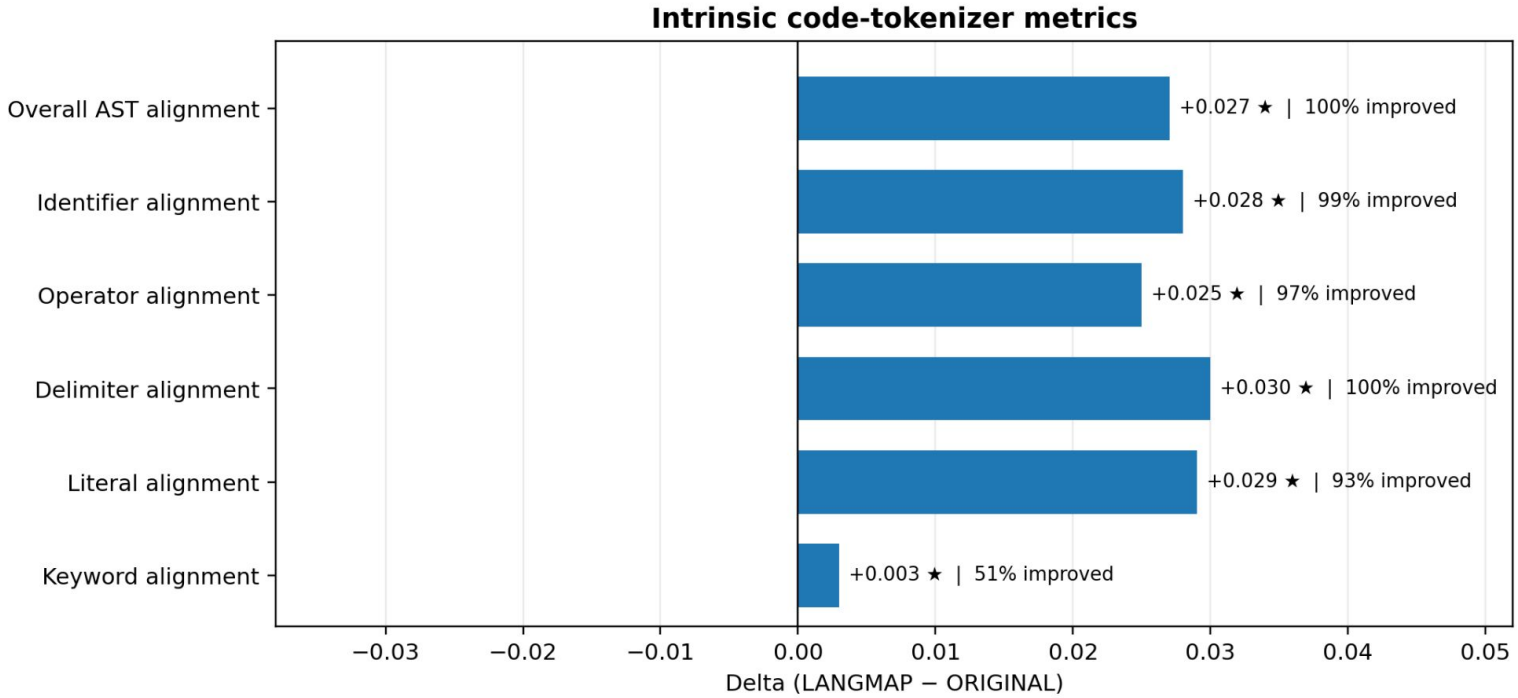


RESULTS

Code Segmentation

LangMAP improves Code Segmentation

Δ of LangMAP F1/Recall/Precision of LangMAP compared to Base Tokenizer.



PRACTICAL ADVANTAGES

Data-Efficient

>70% accuracy with 5 samples/lang

Modular

Add new languages without retraining.Per-language params are independent.

Interpretable

Can easily see and understand why a string was classified as a particular language

Fast Inference

Shared lattice across languages. $O(|\Lambda|)$ extra cost per string.

DISCUSSION

Conclusions

LangMAP adapts Existing Tokenizers without adding to Vocabulary or Changing Model Architecture.

LangMAP improves intrinsic segmentation quality across several natural languages and consistently improves alignment with structural boundaries in code.

Thank You!

LangMAP: A Language-Adaptive Approach to Tokenization

Clara Meister[1], Suchir Salhan[2], Andrzej Szablewksi[2], Pietro Lesci[2], Paula Buttery[2] & Tiago Pimentel[3]

[1]EPFL [2]University of Cambridge [3]ETH Zurich

Additional Slides

OPEN PROBLEMS

Where Current Systems Fail

Open Problems

VOCABULARY COVERAGE
Does the tokenizer's vocabulary contain useful pieces for the language?

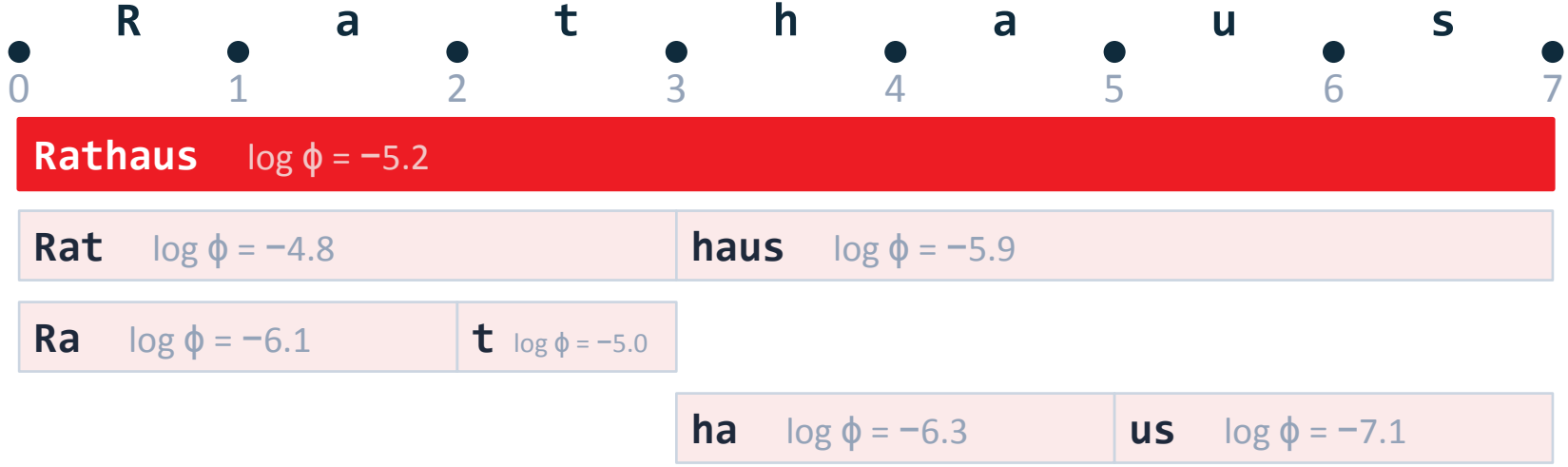
SEGMENTATION
Does the tokenizer choose a linguistically appropriate decomposition?

MODEL ADAPTATION
Does a pretrained LM use a different decomposition effectively?

Pretrained Tokenizers might contain the right pieces for languages, but how does it choose between them?

How Viterbi Segmentation Works

Example: segmenting "Rathaus" given a unigram distribution ϕ with $\mathscr{V}$ = { R, a, t, h, u, s, Ra, ha, us, Rat, haus, Rathaus, … }



Viterbi finds the highest-probability path through the lattice

The lattice encodes all valid segmentations. Each edge is a token v ∈ V with weight log ϕ[v].

Best path: [Rathaus] log p = −5.2 vs. [Rat, haus] log p = −10.7 vs. [Rat, ha, us] log p = −18.2

Introducing LangMAP

A single tokenizer imposes segmentation preferences over all languages, despite languages having systematically different distributions over subwords → Global allocation of probability mass over *V* is not representative of per language distributions

UnigramLM Tokenization

Generative Model

String = detokenized latent token sequence. Each token drawn i.i.d. from categorical ϕ

$$p(v) = \prod \phi[v\square]$$

Marginal: $p(s) = \Sigma\ p(v)$ over all valid segmentations

Parameter Estimation (EM)

Iterate until ~convergence:

- E-step: compute expected token counts via forward–backward over all segmentations
- M-step: normalize counts ⟶ new ϕ
- Vocabulary pruning: remove tokens that hurt likelihood least

Unnecessary details available at: https://cimeister.github.io/blog/unigramlm/