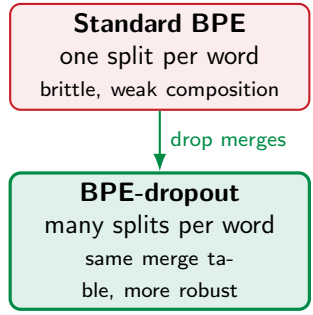
One word, always the same split

Before a model sees any text, a **tokenizer** chops each word into subword units. Byte Pair Encoding (BPE) is the standard choice.

But BPE is **deterministic**: a word gets *exactly one* segmentation, every time. `unrelated` is always `un+related` – the model never sees `un+rel+ated` or any other split.

So it never learns that these are *the same word*, barely learns word **composition**, and shatters on a single typo.

BPE-dropout: at training time, randomly *skip* some merges – the same tokenizer now emits *many* segmentations. No new model, up to **+2.3 BLEU**.



Outline

One segmentation per word

How BPE splits text – and why "always the same split" hurts.

How BPE segments a word

BPE learns two things from a corpus: a **vocabulary** of subwords and a **merge table** – an ordered list of pair-merges by priority.

To segment a word: start from characters, then **greedily apply the highest-priority merge available**, and repeat until none is left.

`l o w e s t` $\rightarrow$ `lo w e s t` $\rightarrow$ `low e s t` $\rightarrow$ `low es t` $\rightarrow$ `low est`

The merge table is fixed and the rule is greedy, so the output is a **deterministic** function of the word: one word $\rightarrow$ one segmentation. Every time.

Why "always the same split" is a problem

A model trained on one fixed segmentation per word:

- **Underuses morphology / composition.** It sees `un+related` but never the pieces `rel`, `ated`, so it does not connect `unrelated`, `relate`, `related` through shared units.
- **Rare subwords are poorly understood.** Frequent character sequences almost always hide *inside* bigger tokens; as standalone tokens they are rare and get weak embeddings (we return to this in the analysis).
- **Brittle to segmentation errors.** One inserted / deleted character can re-route the whole greedy merge path, giving a segmentation the model has never seen.

A single deterministic split throws away free supervision: the many *other* valid ways to cut the same word.

The known fix, and its price

Multiple segmentations help – but the 2018 recipe needs a whole new segmenter.

Subword regularization: sample the split

Idea (Kudo, 2018): instead of one fixed split, treat the segmentation as *random* and marginalize the loss over segmentation candidates:

$$\mathcal{L} = \sum_{(X,Y)\in\mathcal{D}} \mathbb{E}_{\substack{x\sim P(x|X) \\ y\sim P(y|Y)}} \big[\log P(y \mid x, \theta) \big]$$

- At each training step, **sample one** segmentation x of the sentence.
- The model sees the same word cut many different ways over training – a regularizer / data augmentation *at the subword level*. It is not NMT-specific.
- Consistently beats using a single fixed segmentation.

The catch: it abandons BPE

Standard BPE is deterministic, so to *sample* segmentations Kudo (2018) had to build a different segmenter from scratch:

- Train a separate **unigram language model** over subwords.
- Optimize the vocabulary with the **EM** algorithm.
- Sample segmentations at train time with **Viterbi** / *n*-best.

It works, but it is a heavier, separate pipeline that **forbids conventional BPE** – friction that keeps many practitioners from adopting it.

Open question: do we *really* need a new segmentation algorithm to get multiple segmentations? Or can plain BPE already do it?

The whole method in one line

At each merge step, drop each candidate merge with probability p.

You already know this idea: it is dropout

Standard **dropout** randomly zeroes neurons during training, so the network cannot lean on any single unit and is forced to build *redundant* representations. BPE-dropout is the *same* trick, moved into the tokenizer.

NN dropout	**BPE-dropout**
drops *neurons*	drops *merges*
can't lean on one unit	can't lean on one split
redundant features	learns a word's pieces
off at test time	off at inference (p=0)

Same word, cut a different way each pass. The model is forced to recognize `unrelated` from `un+related`, `un+rel+ated`, ... – so it learns the word from its *parts*, not one frozen spelling.

That is the whole intuition. The next slide is just *where* in BPE the coin flip goes.

The method: drop merges at random

Keep BPE's *exact* vocabulary and merge table. Change only **one line** of the segmentation loop:

split ← characters of the word
repeat
  merges ← all applicable merges
  drop each merge w.p. p *(the only change)*
  if merges left: apply the highest-priority one
until no merges left

A dropped high-priority merge forces a *lower* one (or none), so the word breaks at different places each pass.

$p = 0$**:** no drops ⇒ exactly standard BPE.

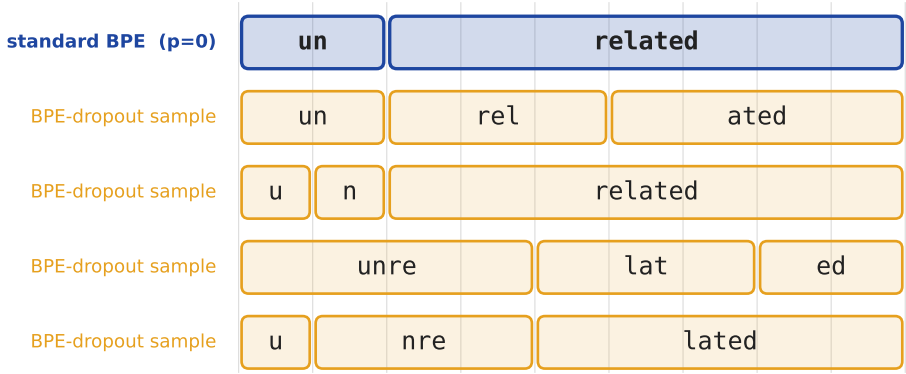
$p = 1$**:** drop everything ⇒ pure characters.

$0 < p < 1$**:** a knob on segmentation *granularity*.

No unigram LM, no EM, no Viterbi – just a coin flip on top of the BPE you already have.

One merge table, many segmentations

One word, one BPE split vs many BPE-dropout splits (same merge table) — illustrative



All of these come from the *same* BPE merge table – randomness in *which* merges are skipped produces the variety. The model now sees `un`, `rel`, `ated`, `lat`, `ed...` as real tokens, not only buried inside `related`.

Train stochastic, infer deterministic

Training: use $p > 0$ (typically $p = 0.1$). Each epoch re-segments words on the fly, exposing the model to many splits – augmentation for free.

Inference: set $p = 0$, i.e. ordinary deterministic BPE. No sampling, no rescoring, standard decoding.

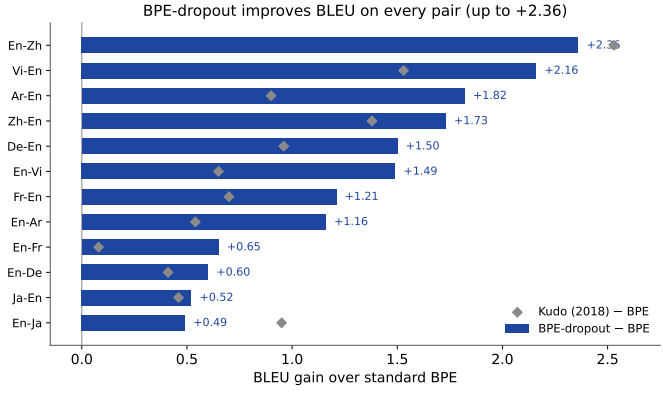
It is regularization / data augmentation *inside the tokenizer*: same model, same vocabulary, same test-time pipeline – only the *training* segmentation is randomized.

So adoption cost is near zero: keep your BPE, flip on dropout during training, and turn it off to deploy.

Does a coin flip actually help?

Translation quality across languages, the effect of p, and of data size.

Consistent BLEU gains across language pairs



On **every** one of 12 IWSLT / WMT / ASPEC pairs, BPE-dropout beats standard BPE – up to **+2.36** BLEU (En-Zh).

It also matches or beats Kudo's costlier subword regularization on **8 of 12** pairs (grey diamonds), while staying inside plain BPE.

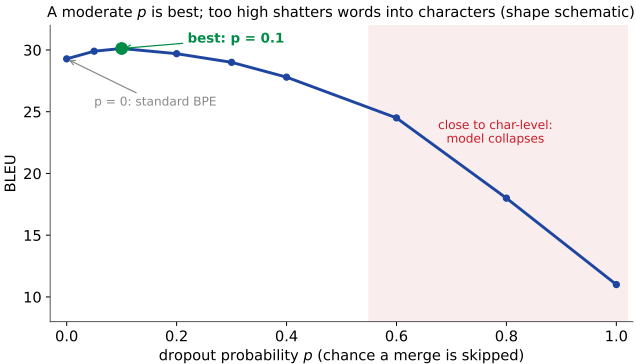
Gains are biggest for morphologically rich / lower-resource pairs; Chinese and Japanese, which need language-specific segmentation, benefit least.

Predict first: what is the best dropout rate p?

If a little randomness helps, does more help more? What happens as $p \to 1$?

Predict first: what is the best dropout rate p?

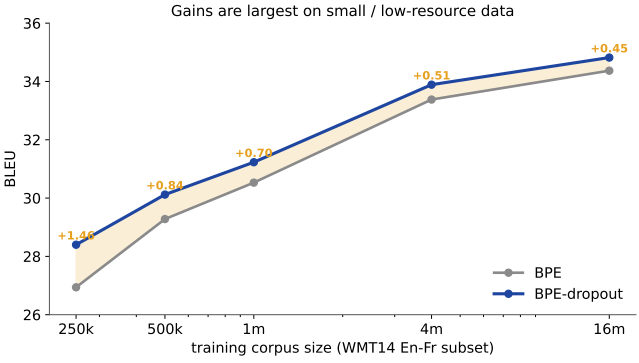
If a little randomness helps, does more help more? What happens as $p \to 1$?



A **moderate** $p \approx 0.1$ is best. Push p too high and training segmentation drifts toward **characters** – far from the deterministic BPE used at inference – and quality **collapses**.

More noise is not better: BPE-dropout is a regularizer, and like any regularizer it has a sweet spot. ($p = 0.6$ for Chinese/Japanese, to match length.)

The gain is largest when data is scarce



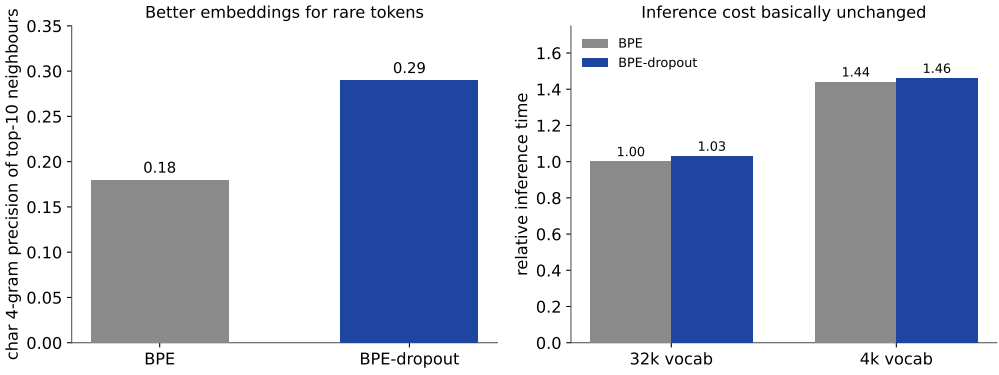
BPE-dropout wins at every corpus size, but the margin **shrinks as data grows** – expected of any regularizer: less overfitting to fight when data is plentiful.

Most valuable for **low-resource** settings. For very large corpora, apply dropout to the *source* side only (both sides can start to hurt).

Why does it work?

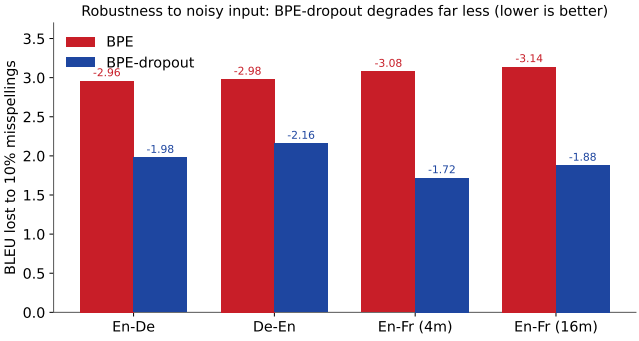
Better subword embeddings, and robustness to noise – at no inference cost.

Better embeddings, essentially free at inference



Left: because frequent substrings now surface as standalone tokens, their embeddings improve – nearest neighbours share more character 4-grams (**0.29** vs **0.18**), so rare tokens are no longer stranded in embedding space. **Right:** generated sequences are not longer in practice, so inference time is **basically unchanged** (Table 4).

Robustness to misspelled input



Corrupt 10% of source words with random edits at test time. Standard BPE loses ∼**3** BLEU; BPE-dropout loses about **half** that.

Striking because the models were **never trained on misspellings** – seeing many segmentations alone teaches robustness. Gains reach **+1.6 to +2.3** BLEU on noisy input *even for large corpora*, where clean gains had nearly vanished.

Recap and practical notes

- Standard BPE is **deterministic**: one split per word, so the model never sees alternatives and is brittle.
- Kudo's subword regularization fixes this but needs a **separate unigram-LM segmenter**; BPE-dropout gets the same benefit from *plain* BPE.
- **BPE-dropout** = at each merge step, skip each merge with probability p. Same merge table, *many* segmentations, **no new model**.
- **Train with** $p \approx 0.1$**, infer with** $p = 0$ – subword-level regularization; too-high p collapses to characters.
- Up to **+2.3** BLEU over BPE (**+0.9** over Kudo); biggest gains on **small / low-resource** data and **noisy** input; near-zero inference cost.
- **Not** a new tokenizer and **not** used at test time – it changes only *how training data is segmented*.

Takeaway: the multiplicity of segmentations was hiding inside plain BPE all along – BPE-dropout just stops forcing a single one.