UNDERSTANDING AI · WRITTEN EDITION · 2026-09-05
Day 2 · typeset 2026-09-10 23:00Z

The best-known failure of large language models is that they cannot count the letters in a word, and the best-known explanation for it is that the word arrives broken into pieces. Run the published tokenisers on the question people actually ask and that explanation does not survive: in the sentence "How many r's are in strawberry?", the word is a single symbol in all five vocabularies tested. The truth is less comfortable than the story. The word is not badly divided; it is not divided at all. What decides that is not the size of the vocabulary but the space in front of the word, which belongs to the symbol — and a benchmark published this year finds that tokenisation explains where a letter sits in a word far better than it explains how many are there.

---

1. The failure, and the explanation everyone gives for it

From 2024, asking a chat model how many times the letter R appears in "strawberry" commonly produced the answer two — often enough that the exchange became a shorthand for a wider unease: a system that could work through a mathematical argument could not perform an operation a six-year-old can.

The explanation that attached itself to the failure is now more widely repeated than the failure. Language models, it runs, do not see letters. They see the word cut into chunks — `str`, `aw`, `berry` — so the three R's are scattered across separate pieces and cannot be counted.

The account is close to right about the mechanism and wrong about the specific case, in a way that matters for what may be concluded from it. Five published tokenisers, run on 5 September 2026 over the sentence a person actually types, give a different answer.

Figure 1. "How many r's are in strawberry?", tokenised

8
Tokens in the whole question
gpt2, cl100k_base, o200k_base and Qwen3.5-4B, identically; nine in Gemma 4, which splits the apostrophe

1
Tokens the word itself becomes
in all five — the space in front of it is part of the symbol

3
Tokens it becomes at the start of a line
in four of the five; in Gemma 4 it is still one

262,144
Entries in the largest vocabulary measured
Google Gemma 4 12B IT, in which "strawberry" is a single entry in every position

Measured 5 September 2026 by executing tiktoken (gpt2, cl100k_base, o200k_base, o200k_harmony) and HuggingFace tokenizers (Qwen3.5-4B, Gemma 4 12B IT) on the string shown. The reproduction scripts and raw output are kept with the episode's research record and are not published here.

▼ **Table view**

Figure 1. "How many r's are in strawberry?", tokenised

Measure	Value
Tokens in the whole question	8
Tokens the word itself becomes	1
Tokens it becomes at the start of a line	3
Entries in the largest vocabulary measured	262,144

The word does not arrive as three awkward fragments. It arrives as one symbol, and what is inside it is not visible from outside. That is a stronger claim than the popular one, not a weaker one — but it is a different claim, and the difference is the whole subject.

Tokeniser	Vocabulary	`strawberry` at a line start	`strawberry` after a space	`Strawberry`
`gpt2` (2019)	50,257	`st` `raw` `berry`	**`strawberry`**	`St` `raw` `berry`
`cl100k_base` (2022)	100,277	`str` `aw` `berry`	**`strawberry`**	`Str` `aw` `berry`
`o200k_base` (2024)	200,019	`st` `raw` `berry`	**`strawberry`**	`Str` `aw` `berry`
Qwen3.5-4B (2026)	248,070	`str` `aw` `berry`	**`Ġstrawberry`**	`Str` `aw` `berry`
Gemma 4 12B IT (2026)	262,144	**`strawberry`**	**`_strawberry`**	**`Strawberry`**

The two 2026 tokenisers write the leading space differently — `Ġ` is GPT-2's byte-level convention, `_` SentencePiece's — and the difference is notation, not behaviour: in both, the space is inside the symbol.

Two readings of the failure are worth refusing before going further.

The first is that this is a defect awaiting a patch. It is not. It follows from a choice made before the model was trained, and the choice buys a great deal.

The second is the opposite: that a system unable to count letters cannot be doing anything worth the name. That reading has the form of rigour without the substance. A fluent reader who cannot report how many serifs appeared on a page is not thereby revealed as illiterate; the units of the question were never the units of the task.

2. Three ways to cut a sentence, and why the field chose the middle one

Before any text reaches a model, a separate program divides it into pieces drawn from a fixed list and hands over the numbers of those pieces. The list is settled once, before training, and then frozen. Everything the model has ever read, in training and in use, arrived this way.

The design space has three regions.

Letters, or bytes. Nothing is ever unrepresentable. The sequences are long, and the computation a transformer performs grows sharply with sequence length, so length is the binding cost. Cheap to store, expensive to think with.

Whole words. The sequences are short. The scheme fails on the first word absent from the list — a name, a typo, a compound, a chemical, a word from another language — and there is always such a word.

Pieces of words. Common words get a single piece; rare ones are assembled from several; and nothing is unrepresentable, because the pieces bottom out in the alphabet the vocabulary was built over.

The third is where the systems in wide use have landed. What decides *which* pieces is where the history becomes surprising.

3. A compression algorithm from February 1994

Byte pair encoding was not invented for language. It was published as a data compression method by Philip Gage in the *C Users Journal* in February 1994, and its opening sentence names the problem it was for:

> Data compression is becoming increasingly important as a way to stretch disk space and speed up data transfers.

The algorithm occupies one paragraph of that article:

> The algorithm compresses data by finding the most frequently occurring pairs of adjacent bytes in the data and replacing all instances of the pair with a byte that was not in the original data. The algorithm repeats this process until no further compression is possible, either because there are no more frequently occurring pairs or there are no more unused bytes to represent pairs.

Two constraints in that sentence are doing more work than they appear to. The replacement symbol must be **a byte that was not in the original data**, and the process stops when the unused bytes run out. Gage was compressing a file, so every symbol he invented had to fit in the same 256-value alphabet the file was written in. His vocabulary could never exceed 256 entries, and it had to shrink the data to be worth anything.

The rest of the article is an engineer's account of a practical tool. The benchmark file is `WIN386.EXE` from Windows 3.1, timed on a 33MHz 486DX. The advantage claimed is not the compression ratio, which the article puts at "almost as much compression as the popular Lempel, Ziv, and Welch (LZW) method", but the

size of the decompression routine, which Gage estimates "should require only about 2K of memory for all code and data" once coded in assembler — small enough, he writes, for "self-extracting programs, image display, communication links and embedded systems". The author's own summary of the result is characteristically flat:

> It's surprising that the BPE algorithm works as well as it does, considering that it discards all information on previous data and does not use variable-sized bit codes, contrary to many modern compression techniques.

Nothing in the article anticipates language modelling, and nothing in it should be read as having done so.

4. What 2015 changed, and the change that usually goes unmentioned

On 31 August 2015 Rico Sennrich, Barry Haddow and Alexandra Birch posted *Neural Machine Translation of Rare Words with Subword Units* (arXiv 1508.07909). The problem was that a translation system carries a fixed vocabulary while translation is, in their phrase, "an open-vocabulary problem", and the previous remedy was to fall back to a dictionary for unknown words.

The paper's own claim is narrower than its later reputation, and the number usually attached to it is narrower still than it looks. The figure in circulation — 1.1 and 1.3 BLEU on the WMT 15 English–German and English–Russian tasks over a back-off dictionary baseline — comes from the fifth version, posted 10 June 2016. The version dated 31 August 2015 says **0.8 and 1.5 BLEU**. The paper as published carries a hedge that arXiv's metadata abstract drops: improvements of "**up to** 1.1 and 1.3 Bleu". And the body reports the range plainly — "the subword ensembles outperform the WDict baseline by 0.3–1.3 Bleu".

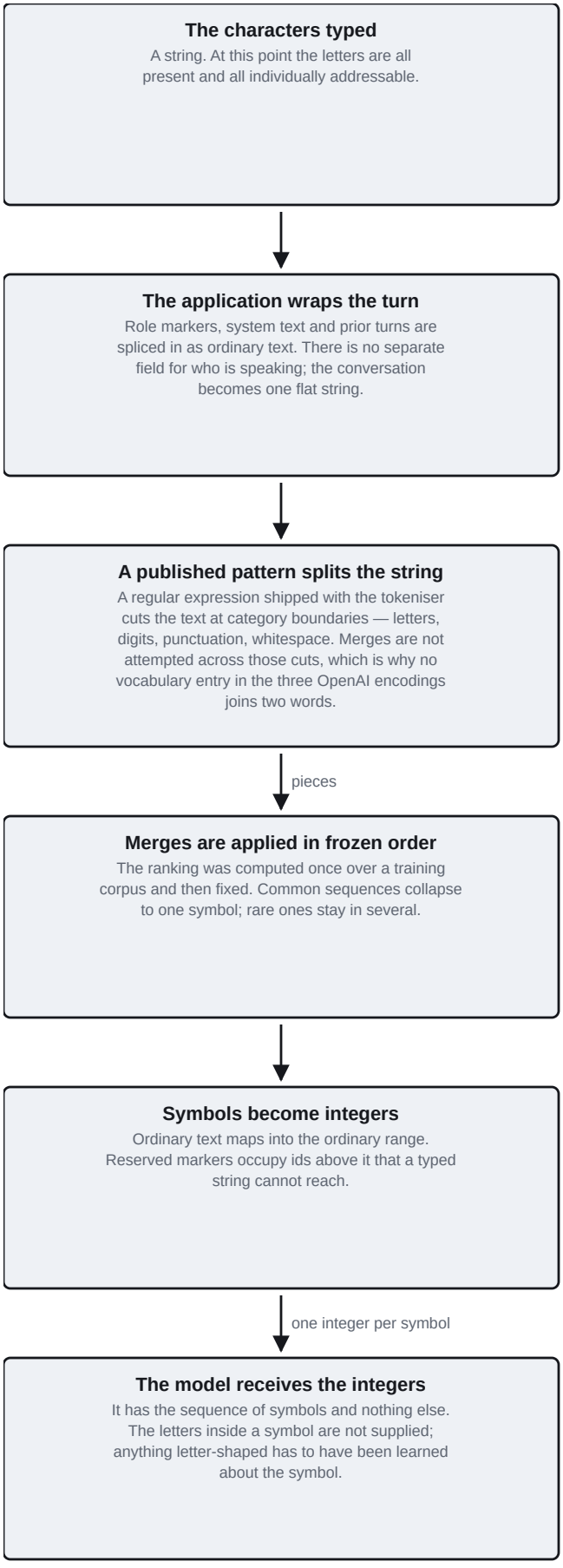
Half of the headline figure is not byte pair encoding's. Table 2 of the paper gives, for English–German with an eight-model ensemble, 24.2 for the WDict baseline, 24.5 for BPE-60k, 24.7 for the joint BPE-J90k, and **25.3 for C2-50k, a character-bigram system**. The +1.1 is the character-bigram result; the best BPE ensemble on that pair gained 0.5. Only the English–Russian 1.3 belongs to a byte-pair system. Byte pair encoding was one of several segmentation techniques under comparison, and on one of the two language pairs it was not the winner.

The adaptation that made the method travel is the one the popular telling omits. Gage stops when he runs out of unused bytes, because a compressor must express its output in the alphabet it started with. A system building a *vocabulary* has no such obligation: a merged pair simply becomes a new entry on the list, and the list is as long as its author decides. Removing that ceiling is what turns a 256-symbol compressor into a 200,000-symbol tokeniser. It is a small change to the algorithm and the entire difference in what it can be used for.

What survives from Gage is the mechanism and its consequence. The merges are performed in frequency order over a corpus, then stopped at a chosen count and frozen. The resulting list is therefore not designed. It is a frequency ranking of somebody's pile of text, preserved. Which words are cheap and which are expensive was settled by what was in the pile.

5. What happens between the keyboard and the model

Figure 2. Every request, before the model sees anything


The characters typed
A string. At this point the letters are all present and all individually addressable.
The application wraps the turn
Role markers, system text and prior turns are spliced in as ordinary text. There is no separate field for who is speaking; the conversation becomes one flat string.
A published pattern splits the string
A regular expression shipped with the tokeniser cuts the text at category boundaries — letters, digits, punctuation, whitespace. Merges are not attempted across those cuts, which is why no vocabulary entry in the three OpenAI encodings joins two words.
pieces
Merges are applied in frozen order
The ranking was computed once over a training corpus and then fixed. Common sequences collapse to one symbol; rare ones stay in several.
Symbols become integers
Ordinary text maps into the ordinary range. Reserved markers occupy ids above it that a typed string cannot reach.
one integer per symbol
The model receives the integers
It has the sequence of symbols and nothing else. The letters inside a symbol are not supplied; anything letter-shaped has to have been learned about the symbol.

Schematic. Drawn from the published tokeniser files and the tiktoken and HuggingFace tokenizers libraries, read 5 September 2026. Every stage is software that runs before the model is called, and every stage is a choice made by somebody other than the person typing.

▼ **Table view**

Figure 2. Every request, before the model sees anything — stages

#	Stage	Note
1	The characters typed	A string. At this point the letters are all present and all individually addressable.
2	The application wraps the turn	Role markers, system text and prior turns are spliced in as ordinary text. There is no separate field for who is speaking; the conversation becomes one flat string.
3	A published pattern splits the string	A regular expression shipped with the tokeniser cuts the text at category boundaries — letters, digits, punctuation, whitespace. Merges are not attempted across those cuts, which is why no vocabulary entry in the three OpenAI encodings joins two words.
4	Merges are applied in frozen order	The ranking was computed once over a training corpus and then fixed. Common sequences collapse to one symbol; rare ones stay in several.
5	Symbols become integers	Ordinary text maps into the ordinary range. Reserved markers occupy ids above it that a typed string cannot reach.
6	The model receives the integers	It has the sequence of symbols and nothing else. The letters inside a symbol are not supplied; anything letter-shaped has to have been learned about the symbol.

Figure 2. Every request, before the model sees anything — connections

From	To	Label
The characters typed	The application wraps the turn	
The application wraps the turn	A published pattern splits the string	
A published pattern splits the string	Merges are applied in frozen order	pieces
Merges are applied in frozen order	Symbols become integers	
Symbols become integers	The model receives the integers	one integer per symbol

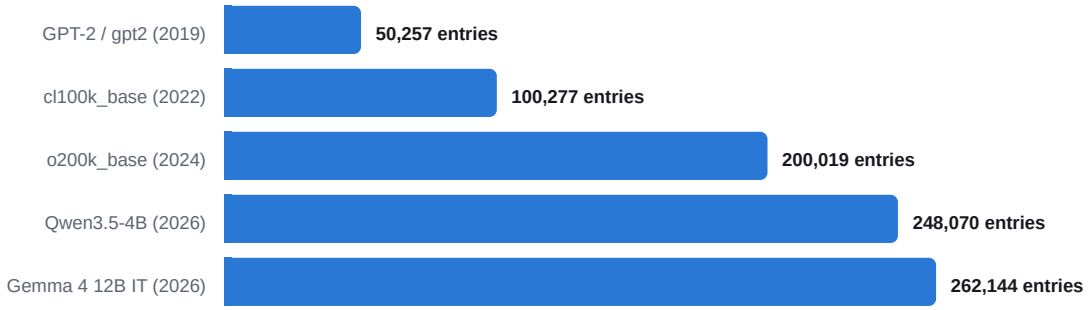
Two properties of that pipeline are checkable rather than folkloric, and both were checked.

No vocabulary entry joins two words. Every entry in each of the three OpenAI encodings that decodes to text was examined for an interior space with content on both sides. There are none: zero among the 49,913 decodable entries of `gpt2`'s 50,257, zero among 99,488 of `cl100k_base`'s 100,277, zero among 198,438 of `o200k_base`'s 200,019. The remainder are partial byte sequences that are not text at all, and cannot join two words for that reason. Consistently, `of the`, `New York`, `United States` and `is not` are two tokens in all three. This is usually stated as a rule about merging; it is more precisely a consequence of the splitting pattern that runs first, which cuts the string at category boundaries so that no merge is ever offered a pair that straddles one.

A leading space belongs to the symbol. `strawberry` and `strawberry` are different entries, which is why position in a sentence changes how a word is cut — and why the popular three-piece account, which is correct for the word standing alone, is wrong for the word inside a question.

6. The vocabularies grew, and what that does and does not explain

Figure 3. Vocabulary size of published tokenisers, by year of release



Read off each encoding itself on 5 September 2026 — tiktoken's n_vocab for the three OpenAI encodings, get_vocab_size() for the two HuggingFace tokenisers — rather than from documentation. o200k_base entered OpenAI's public tiktoken repository on 13 May 2024, in a commit whose changelog entry reads "Support for gpt-4o".

▼ **Table view**

Figure 3. Vocabulary size of published tokenisers, by year of release

Tokeniser	Entries
GPT-2 / gpt2 (2019)	50,257 entries
cl100k_base (2022)	100,277 entries
o200k_base (2024)	200,019 entries
Qwen3.5-4B (2026)	248,070 entries
Gemma 4 12B IT (2026)	262,144 entries

The obvious inference from that column — bigger vocabulary, fewer words left divided — does not survive the measurement. Standing alone at the start of a line, "strawberry" is still three pieces in four of the five tokenisers, including Qwen3.5-4B at 248,070 entries, the second largest measured. Only Gemma 4 keeps it whole in every position, and Gemma 4 differs from the other four in scheme as well as size: a SentencePiece-style normaliser, byte fallback, and a declared unknown token. Size and scheme move together across that row, so neither is isolated, and the trend the column appears to show is not established by it.

What does hold, in all five, is the smaller fact. The word is a single symbol in the sentence a person actually types, and the reason is the leading space: `strawberry` and `strawberry` are different entries, so a word's position in a sentence decides how it is cut. That is a property of the pre-tokenisation rule rather than of the vocabulary's size, and it is what the popular three-piece account gets wrong about the famous case.

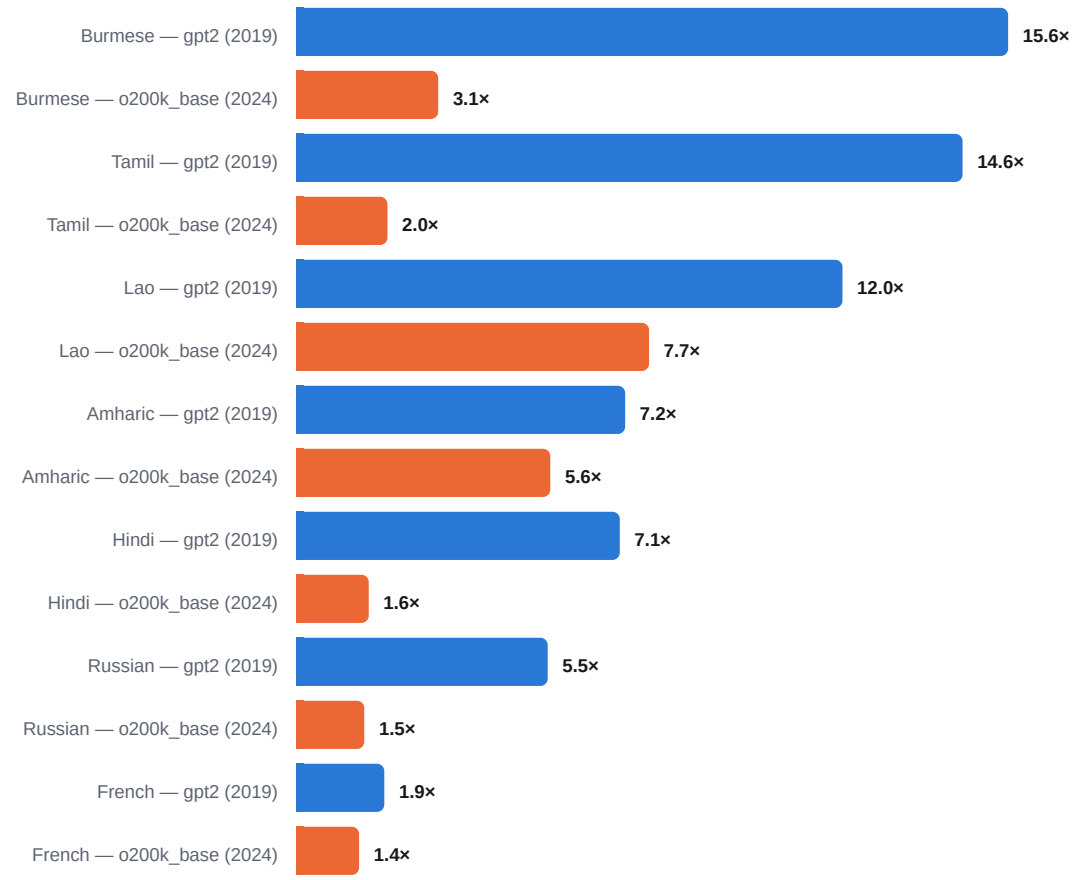
The same growth is visible across the open-weight models whose configurations are published. Six were fetched again on 5 September 2026, and each one's `config.json` and `tokenizer_config.json` were read directly: DeepSeek V4 Pro at 129,280, GLM-5.3 at 154,880, Kimi K3 at 163,840, `gpt-oss-120b` at 201,088, Qwen3.8-27B at 248,320 and Gemma 4 at 262,144. All six ship a tokeniser configuration. Mistral's Small 4 repository answered 401 to the same request, and its figure is therefore absent from the list.

7. What a frozen frequency ranking costs, and to whom

If the vocabulary is a preserved ranking of one corpus, then the cost of writing in a given language depends on how much of that language was in the corpus — and cost here is literal, because providers bill by the token and context windows are measured in them.

The effect is measurable rather than inferable. The table below runs the first 100 sentences of the FLORES-101 development set — the same source articles, professionally translated into each language — through three generations of published OpenAI encoding and one 2026 open-weight tokeniser. Figures are tokens relative to the identical 100 sentences in English.

Figure 4. Tokens for the same 100 sentences, relative to English



Measured 5 September 2026 over the first 100 sentences of each language in the FLORES-101 dev split, encoded with tiktoken. Blue is the 2019 encoding, orange the 2024 one. Seven of the 41 languages measured are shown; the full 41-language table is not reproduced here. Relative cost is that language's token count divided by English's for the identical content.

▼ **Table view**

Figure 4. Tokens for the same 100 sentences, relative to English

Language and encoding	Relative cost
Burmese — gpt2 (2019)	15.6×
Burmese — o200k_base (2024)	3.1×
Tamil — gpt2 (2019)	14.6×
Tamil — o200k_base (2024)	2.0×
Lao — gpt2 (2019)	12.0×
Lao — o200k_base (2024)	7.7×
Amharic — gpt2 (2019)	7.2×
Amharic — o200k_base (2024)	5.6×
Hindi — gpt2 (2019)	7.1×
Hindi — o200k_base (2024)	1.6×
Russian — gpt2 (2019)	5.5×

Language and encoding	Relative cost
Russian — o200k_base (2024)	1.5×
French — gpt2 (2019)	1.9×
French — o200k_base (2024)	1.4×

Language	gpt2 (2019)	cl100k_base (2022)	o200k_base (2024)	Qwen3.5-4B (2026)
Lao	11.96×	8.98×	**7.72×**	4.44×
Amharic	7.20×	7.31×	**5.56×**	4.12×
Khmer	14.76×	8.86×	3.38×	5.18×
Burmese	15.59×	11.16×	3.10×	3.30×
Punjabi	7.53×	7.72×	2.60×	3.66×
Yoruba	3.60×	2.85×	2.10×	2.43×
Tamil	14.59×	7.36×	1.99×	2.53×
Japanese	2.91×	2.29×	1.70×	1.22×
Hindi	7.08×	4.68×	1.58×	1.98×
Russian	5.50×	2.51×	1.48×	1.44×
Chinese (simplified)	3.26×	2.00×	1.35×	1.05×
French	1.92×	1.60×	1.37×	1.38×
Portuguese	1.82×	1.46×	1.21×	1.22×
English	1.00×	1.00×	1.00×	1.00×

Two findings sit in that table and they point in opposite directions.

The improvement is large and deserves to be stated first. Tamil fell from 14.59× to 1.99×, Burmese from 15.59× to 3.10×, Hindi from 7.08× to 1.58×. A five-year-old complaint about these systems is substantially less true than it was, and the newest tokeniser measured is the best of the four for most languages in the set.

The improvement is also uneven, and it is least where it was worst. Lao began furthest from English and remains furthest: 11.96× in 2019, 7.72× in 2024. Amharic moved from 7.20× to 5.56×. A speaker of Lao pays roughly eight times what an English speaker pays to put the same paragraph in front of the same model, and that ratio has improved by about a third in five years while Tamil's improved sevenfold. Why any particular language moved as it did is not visible from outside: the vocabularies are published as frozen files, and the corpora they were built over are not.

8. The markers a keyboard cannot reach

A vocabulary also holds entries that no typed character sequence produces. These are how an application separates its own instructions from the user's text, and the separation is thinner than it looks.

OpenAI's published `o200k_harmony` encoding was read directly, every identifier encoded and its integer read back.

Marker	Id		Marker	Id
`<|endoftext|>`	199,999		`<|start|>`	200,006
`<|return|>`	200,002		`<|end|>`	200,007
`<|constrain|>`	200,003		`<|message|>`	200,008
`<|channel|>`	200,005		`<|call|>`	200,012

The role is not a marker at all. `assistant` encodes to id 173,781 and `user` to id 1,428 — ordinary words, in the ordinary range, indistinguishable as symbols from any other word. There is no field carrying who is speaking. A conversation is one flat run of integers, and within that run the only thing dividing the application's frame from the user's contribution is that the markers live above the range a typed string can reach.

That framing is also billed. `What is 2 + 2?` is eight tokens. The same question wrapped as one conversational turn — `<|start|>user<|message|>What is 2 + 2?<|end|><|start|>assistant` — is fourteen. Six tokens of punctuation on an eight-token question, on every turn.

9. An acknowledgement inside the library

The unevenness in Figure 4 is not a discovery made from outside. OpenAI's `tiktoken` library is open source, and the function that defines `o200k_base` carries a comment that entered the file on 2 October 2024 and was still there when it was read on 5 September 2026:

> This regex could be made more efficient. If I was the one working on this encoding, I would have done a few other things differently too, e.g. I think you can allocate tokens more efficiently across languages.

The comment is unsigned and arrived in a bulk synchronisation from OpenAI's internal codebase, and its own wording — "If I was the one working on this encoding" — places its author outside the team that built the encoding. Who wrote it is not recoverable from the repository. Directly beneath it sits the pattern the comment is about, and one line of that pattern — `\p{N}{1,3}` — is why numbers of four or more digits are cut into groups of at most three, which is the published cause of the arithmetic behaviour that is usually described as mysterious.

The same encoding, unchanged, is what OpenAI's library maps its current model family to. As read on 5 September 2026 the file `tiktoken/model.py` routes every `gpt-5`-prefixed name to `o200k_base`, and the only later encoding in the library, `o200k_harmony`, is built by calling `o200k_base()` and reusing its merge table and its splitting pattern verbatim. Loading both on 5 September 2026 and comparing them confirms it: 199,998 merges each, identical; the same splitting pattern; 1,089 additional reserved markers and no

additional merges. The newest set of merges OpenAI publishes is therefore still the one added on 13 May 2024. Two years and two model generations have passed over one frozen list — as published; what a hosted model does behind the API is not visible from a client library.

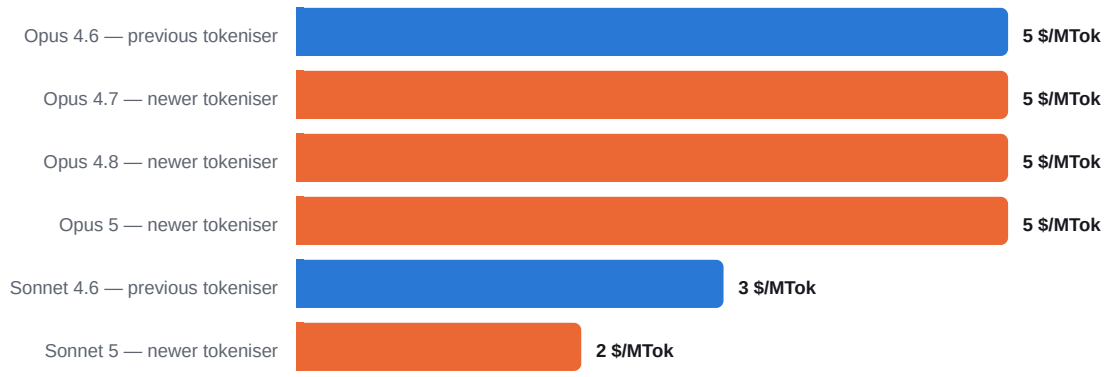
10. When the tokeniser is the price

The clearest demonstration that a vocabulary is an economic object rather than a technical detail sits in Anthropic's developer pricing documentation, where it was read on 5 September 2026:

> Claude 4.7 and later models and Claude Mythos Preview use a newer tokenizer that contributes to their improved performance on a wide range of tasks. This tokenizer produces approximately 30% more tokens for the same text. The exact increase depends on the content and workload shape. Claude Sonnet 4.6 and earlier models use the previous tokenizer.

A vendor is stating that the same text now counts as roughly 30% more of the thing it bills for. What follows from that depends entirely on the per-token price on each side of the boundary, and the same page carries those figures.

Figure 5. Anthropic's published input price per million tokens, across the tokeniser change



Anthropic developer pricing documentation, read 5 September 2026; the same page states that models from 4.7 onward produce approximately 30% more tokens for the same text. Blue is the previous tokeniser, orange the newer one. Across the Opus line the per-token price is identical on both sides. Across the Sonnet line it falls by a third. The two lines therefore move in opposite directions once the token-count change is applied, and the page attributes no pricing decision to the tokeniser.

▼ **Table view**

Figure 5. Anthropic's published input price per million tokens, across the tokeniser change

Model	Input price
Opus 4.6 — previous tokeniser	5 $/MTok
Opus 4.7 — newer tokeniser	5 $/MTok
Opus 4.8 — newer tokeniser	5 $/MTok
Opus 5 — newer tokeniser	5 $/MTok
Sonnet 4.6 — previous tokeniser	3 $/MTok
Sonnet 5 — newer tokeniser	2 $/MTok

Across the Opus line the published input price is $5 per million tokens on both sides of the change, so approximately 30% more tokens for the same text means approximately 30% more tokens billed for it. Across the Sonnet line the price falls from $3 to $2 between Sonnet 4.6 and Sonnet 5, a reduction of a third that roughly offsets the same increase. Those models differ in more than their tokeniser, and the page draws no connection between the two facts; what it establishes is that a tokeniser change and a price change are the same kind of event, and that at least one vendor now discloses the first as a pricing note.

11. How much of the failure tokenisation actually explains

Tokenisation is where an explanation of the strawberry failure has to start, and the popular account tends to absorb several things that do not belong to it.

The letters are not destroyed. A model asked to spell a word out will generally do it, so something about the composition of each symbol has been learned during training rather than supplied at input. The accurate statement is narrower than "it cannot see the letters": the letters are not the units it was handed, so any letter-shaped operation is a reconstruction rather than a lookup.

Handing the letters over directly removes the obstacle. `s t r a w b e r r y` is exactly ten tokens — one per letter — in `gpt2`, `cl100k_base`, `o200k_base` and Gemma 4, and Zhang, Cao and You measured what that does to performance in *Counting Ability of Large Language Models and Impact of Tokenization* (arXiv 2410.19730, October 2024): separating the items produced "consistent improvements (13%-40%) over pure BPE". Their error analysis is the more telling half. Under ordinary subword tokenisation every error they recorded was an *undercount* — never once too many — which is the signature the mechanism predicts, since "the model might fail to count any 'a's in a single token like 'abaa'". With the items clearly separated the extreme errors disappear and what remains clusters between one and three, which they judge "likely due to minor arithmetic mistakes by the model".

A benchmark published this year partitions the claim more sharply still. CharBench, by Omri Uzan and Yuval Pinter (arXiv 2508.02591, v3 6 April 2026; AAAI-26), evaluates character-level reasoning across proprietary and open-weight models and reports an average accuracy of 50.3%. Its finding on the two task families is the important one, in its own words:

> For counting tasks, we find that tokenization properties are weakly correlated with correctness, while the length of the queried word and the actual character count play a more significant part. In contrast, for tasks requiring intra-word positional understanding, performance is negatively correlated with the length of the token containing the queried character, suggesting that longer tokens obscure information on character position.

On that measurement the vocabulary explains where a letter sits better than it explains how many there are — which cuts against the popular account of the strawberry failure specifically, since counting is what that failure is.

Zhang, Cao and You's paper is the other strong argument against treating tokenisation as the whole explanation. Without chain-of-thought prompting, its models' counting accuracy fell from around 50% to 8% as strings grew from ten-to-twenty characters to thirty-to-forty — "regardless of tokenization", and barely above the 3-4% of random guessing. With chain-of-thought the same range declined from 96% to 56%. The paper's framing puts the ceiling in the architecture: a transformer computes at constant depth, counting needs depth that grows with length, and tokenisation is what pushes an already-limited system further from the answer. Two causes, then, and on CharBench's measurement the architectural one carries more of the counting result than the vocabulary does.

These are measurements of what reaches a model, not of what a model does with it: whether any particular system answers the strawberry question correctly in September 2026 was not tested, and no closed frontier model's tokeniser could be measured, because none is published as a file. `o200k_base` is OpenAI's published encoding — the nearest available object, and not the tokeniser running behind the current API.

12. Two ways to keep the promise, and one attempt to remove the step

The guarantee that nothing is unrepresentable is usually explained one way: since 2019 the merges have run over raw bytes, so every possible input is expressible and there is no such thing as an unknown symbol. That explanation covers the OpenAI lineage and does not cover the field.

Google's Gemma 4 12B IT, refetched and read out of its own `tokenizer.json` on 5 September 2026 — 32,169,626 bytes, sha256 `cc8d3a0c…`, byte-identical to the previous day's copy — is a BPE tokeniser with `byte_fallback` set true, a normaliser that replaces spaces with `▁` in the SentencePiece manner, 514,906 merges — and an `<unk>` token present at vocabulary id 3. A byte-level vocabulary cannot have an unknown token, because every byte is already in it. Gemma 4 reaches the same guarantee by falling back to bytes when a character is not covered, which is a different mechanism with the same promise: 🫆 `ÞR ∀` encodes to sixteen tokens, round-trips exactly, and emits no `<unk>`. Kimi K3's published configuration, fetched the same day, declares `[UNK]` as well. Two models shipped in 2026 carry a symbol that a byte-level scheme has no use for.

There is also a research programme trying to remove the step altogether, and it has moved a long way since Meta's *Byte Latent Transformer* (arXiv 2412.09871, December 2024) made the case that patches scale better than tokens. Since then: *Dynamic Chunking for End-to-End Hierarchical Sequence Modeling* from Sukjun Hwang, Brandon Wang and Albert Gu (arXiv 2507.07955, July 2025), which learns where the boundaries go rather than being told; *Bolmo* from the Allen Institute (arXiv 2512.15586, December 2025, revised February 2026), which converts existing subword models into byte-level ones and is downloadable; and *Fast Byte Latent Transformer* (arXiv 2605.08044, May 2026). Bolmo's own abstract is careful about what it demonstrates — the models "approach the capabilities of subword-based systems" and remain "competitive across standard benchmarks", which is a claim about closing a gap rather than reversing it.

Two things about that programme are worth stating precisely, because both cut against the obvious reading.

The first is adoption. On 5 September 2026 the HuggingFace API reports 428 downloads over the preceding thirty days for `allenai/Bolmo-7B` and 3,195,490 for `google/gemma-4-12B-it`. The byte-level line publishes competitive results; almost nobody runs it.

The second is definitional, and it was put most sharply by Catherine Arnett in an article for Hugging Face on 25 September 2025, subtitled "An argument in defense of tokenizers":

> No matter how you chunk up your input data, you're doing tokenization.

Bolmo's own published files make the case for her, and its `config.json`, read on 5 September 2026, makes it twice. It ships a `tokenization_bolmo.py` and a `tokenizer_config.json` declaring a `BolmoTokenizer` class; its `vocab_size` is **520**; and in the same object sits `"subword_vocab_size": 100278`, a nested `original_identifier` of `allenai/dolma2-tokenizer`, and a special token named `<bpe_token_end>`. The vocabulary did not disappear. It went from roughly a hundred thousand entries to 520, and the file that says so still records the hundred thousand it was converted from. What the research line is really contesting is how coarse the units should be and who chooses them, not whether there are units.

That reframing also settles what byte-level models would and would not fix. Petrov, La Malfa, Torr and Bibi, in *Language Model Tokenizers Introduce Unfairness Between Languages* (arXiv 2305.15425, May 2023), measured "differences up to 15 times in some cases" between the same text in different languages — a figure the FLORES measurement above independently reproduces in order of magnitude for the tokeniser of that period. The same abstract carries a sentence that is quoted far less often:

> Character-level and byte-level models also exhibit over 4 times the difference in the encoding length for some language pairs.

Removing the vocabulary narrows the disparity. On the authors' own measurement it does not remove it, because some writing systems simply need more bytes per unit of meaning than others.

13. Three things that can be checked, and on what date

The newest set of merges in OpenAI's own library. `tiktoken_ext/openai_public.py` defines seven encodings. The last one to introduce a new merge table is `o200k_base`, added 13 May 2024; `o200k_harmony`, which comes after it, calls `o200k_base()` and reuses that table unchanged — the two encodings' merge tables were loaded and compared on 5 September 2026 and are identical. Changelog entries since have added model *lookups* and no new merges. A new merge table in that file, or a `gpt-` prefix in `tiktoken/model.py` pointing somewhere other than `o200k_base`, is the observable event that the published vocabulary has been rebuilt, and both files are public.

What that map is a claim about. On 17 August 2026, commit `212b893` — titled "Fail open for GPT-5 model tokenisers" — changed one line of `tiktoken/model.py` from the prefix `"gpt-5-"` to `"gpt-5"`, so that any model name beginning with those five characters resolves to `o200k_base` instead of raising an error. The file states the trade in its own comment: prefix matching "avoids needing library updates for every model version release", and "this can match on non-existent models (e.g., gpt-3.5-turbo-FAKE)". As read on 5 September 2026 the map contains no entry naming a `gpt-6` model at all. What the map records is which encoding OpenAI's client library will use for a name, which is not the same object as the tokeniser running behind the API, and after 17 August it will answer for names nobody has shipped.

A frontier model shipping with no tokeniser. Byte-level and dynamic-chunking models already exist and can be downloaded, so their existence is not the signal. The signal is one of the flagship systems a laboratory charges money for arriving without a tokeniser file or a published encoding — an absence that is checkable on a release date.

What "better support for language X" means when a provider announces it. Two different things travel under that phrase: more of the language in the training corpus, and a rebuilt vocabulary in which the same text costs fewer tokens. Only the second changes the bill, and only the second is measurable from outside — the same passage, encoded before and after, in about a minute.

14. What it comes down to

A model does not receive what was typed. It receives a run of integers pointing into a list that was frozen before it was trained, produced by a compression algorithm run over a corpus that was never published, with the merges taken in frequency order and stopped at a round number.

A great many of the small surprising failures trace back to that, on Karpathy's own account, and so does a distributional question that looks like an engineering detail. The letters are not hidden; they were never the units. The word is not badly divided in the sentence people actually ask; it is one symbol, because the space in front of it is part of the symbol.

And the received explanation for the most famous of those failures — `str`, `aw`, `berry` — turns out to be one measurement away from wrong. It described a real mechanism, applied to the wrong case, and it survived years of repetition because running the check takes a minute and nobody had a reason to.

15. Sources

Source	Date	Location
Philip Gage, *A New Algorithm for Data Compression*, *C Users Journal* 12(2), pp. 23–38	February 1994	Text from the Internet Archive capture of 29 Mar 2021, `web.archive.org/web/20210329111703/http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM`, fetched 4 Sep 2026. The month is not on that page; it comes from the bibliographic record, checked against OpenAlex on 5 Sep 2026, which dates it 1994-02-01 at volume 12, issue 2, and files the magazine under its later name, *C/C++ Users Journal*
Rico Sennrich, Barry Haddow, Alexandra Birch, *Neural Machine Translation of Rare Words with Subword Units*	arXiv v1 31 Aug 2015	arXiv 1508.07909
Andrej Karpathy, *Let's build the GPT Tokenizer*, and the written lecture in `karpathy/minbpe`	video 20 Feb 2024; lecture re-fetched 5 Sep 2026, sha256 `1dcf4236f8c8…`, byte-identical to the previous day's copy	karpathy/minbpe on GitHub — `lecture.md`
OpenAI, `tiktoken`: `tiktoken_ext/openai_public.py`, `tiktoken/model.py`, `CHANGELOG.md`	`o200k_base` added 13 May 2024 (commit `9d01e56`, v0.7.0); language comment added 2 Oct 2024 (commit `05e66e8`); prefix widened 17 Aug 2026 (commit `212b893`); all read 5 Sep 2026	openai/tiktoken on GitHub

Source	Date	Location
Google, Gemma 4 12B IT `tokenizer.json` (32,169,626 bytes, sha256 `cc8d3a0c…`) and `tokenizer_config.json`	re-fetched 5 Sep 2026, byte-identical to the 4 Sep copy	huggingface.co/google/gemma-4-12B-it
Published configurations: DeepSeek V4 Pro, GLM-5.3, Kimi K3, Qwen3.8-27B, `gpt-oss-120b`, Gemma 4	`config.json` and `tokenizer_config.json` re-fetched 5 Sep 2026	huggingface.co
FLORES-101 development split, 41 languages × 100 parallel sentences	re-encoded 5 Sep 2026; every relative figure identical to the 4 Sep run	HuggingFace datasets-server
Petrov, La Malfa, Torr & Bibi, *Language Model Tokenizers Introduce Unfairness Between Languages*	arXiv v1 17 May 2023	arXiv 2305.15425
Zhang, Cao & You, *Counting Ability of Large Language Models and Impact of Tokenization*	arXiv v1 25 Oct 2024	arXiv 2410.19730
Omri Uzan & Yuval Pinter, *CharBench: Evaluating the Role of Tokenization in Character-Level Tasks*	arXiv 2508.02591, v3 6 Apr 2026; AAAI-26	arXiv 2508.02591
Pagnoni et al., *Byte Latent Transformer*; Hwang, Wang & Gu, *Dynamic Chunking*; Minixhofer et al., *Bolmo*; Kallini, Pagnoni et al., *Fast Byte Latent Transformer*	13 Dec 2024; 10 Jul 2025; 17 Dec 2025; 8 May 2026	arXiv 2412.09871, 2507.07955, 2512.15586, 2605.08044
Catherine Arnett, *There is no such thing as a tokenizer-free lunch*	25 Sep 2025	huggingface.co/blog/catherinearnett/in-defense-of-tokenizers
Anthropic developer pricing documentation: the tokeniser note and the published per-token prices	read 5 Sep 2026	platform.claude.com/docs/en/about-claude/pricing
`allenai/Bolmo-7B` `config.json`, `tokenizer_config.json` and the HuggingFace model API download counts	read 5 Sep 2026	huggingface.co/allenai/Bolmo-7B