Where do Chinese words begin and end?

Measuring and improving Pindu's approach to Chinese Word Segmentation. 2026-10-22

Evaluation for Anki Add-On v1.16.3 (May 2026).

In short

- A Chinese word segmenter's score depends on what the benchmark calls a word.
- We built reading-oriented answer keys from two corpora's annotations and tested five segmentation approaches. On the UD sample, changing the key changed the comparison between engines; on LCMC, the ranking held steady.
- HanLP scored about 0.90 against our aligned keys and jieba 0.85. We kept jieba for the desktop Anki add-on because it was small and fast.
- The aligned keys express our design judgments about useful reading units. We have not tested whether they improve learning.

In most written languages, finding word boundaries is trivial: you split a text on the literal spacing between words. Chinese, despite some attempts at reform, doesn't have spaces. A sentence is written as an unbroken run of characters, and the reader has to parse the text into meaningful segments as they go along. For native readers, this process is automatic.[1] For learners, it's painstaking. For computers, as it turns out, it's an unsolved problem. That's not so much because the problem (called "Chinese Word Segmentation" or CWS) is difficult but more because there's often no right answer.

Despite that ambiguity, CWS is critical in Pindu for a good reading experience and for proper text modeling. We need to measure and improve it, even if perfecting it is impossible. Since before Pindu launched, we have been bundling the `jieba` CWS package because of its popularity in the community and because its small footprint matched our Anki Add-On deployment shape. But segmentation performance has been a constant headache for us, and we wanted to take a more measured survey of the landscape. This article introduces the problem of CWS, how we think about it at Pindu, evaluations of the major approaches, and ultimately circles back to the choice of one for Pindu.

Why is segmentation hard and why does it matter?

A common example to illustrate the ambiguity of CWS is 南京市长江大桥, as shown in Figure 1. This phrase can naturally be parsed two ways: one is 南京市 / 长江 / 大桥, "the Yangtze River Bridge in Nanjing"; another is 南京 / 市长 / 江大桥, "Nanjing mayor Jiang Daqiao". Both are valid Chinese, and it's only experience, context (and maybe some knowledge of who is or isn't named Jiang) that can tell you which was meant by the writer. Learners lack experience with the language, and older algorithms struggle to model context.[2] To be fair, many sentences in Chinese are straightforward and uncontroversial to decompose into words, but CWS ambiguity is not a rare edge case. Is 人造卫星 ("man-made satellite") one word or three? Is 北京大学 a single institution, or 北京 plus 大学? Reasonable people and reasonable dictionaries can disagree.

[1] Bai et al. (2008) found that native readers parse unspaced text so fluently that adding spaces between words doesn't speed them up at all.

[2] The rise of LLMs has significantly improved context modeling, although application in CWS is still an active area of research (see Zhang et al., 2025).



Figure 1: The same string segmented in two valid ways. A segmenter has to commit to one, but nothing in the sequence itself can determine which the writer intended.

For Pindu this matters practically. The software relies on segmentation in three places, and that's three places where a mistake has consequences. First, the reader interface lets you click on individual segments for dictionary or pronunciation help, so mis-segmenting a string will frustrate the user's intentions. Second, the vocabulary model reasons about a passage word by word, so a bad cut corrupts the leveling and personalization outcomes. Third, Pindu credits what you read back to Anki by matching segments against the cards in your collection, so a misplaced boundary will result in incorrect crediting. So, we care a lot about CWS performance.

How segmentation performance is measured

Because the "correct" segmentation (when it exists) depends on the writer's intent, the passage's context, and the reader's experience, the quality of a particular segmentation must be assessed against a human-annotated ground truth called the "gold" segmentation. A set of raw text, their gold segmentations, and any other annotations is called a gold corpus. Over the past decades, researchers have assembled and meticulously segmented many thousands of Chinese passages into various corpora to support Natural Language Processing (NLP) research. These corpora, the principal of which are summarized in Table 1, span genres of text and are packed with metadata like part-of-speech, topic, etc. To measure the performance of any given segmentation algorithm, you would first select a particular gold corpus as the benchmark, apply the algorithm to the gold text, and then compare the results against the gold segmentation. The score doesn't measure absolute correctness, but conformance to the conventions of the standard. This is an important distinction.

For Pindu we evaluate CWS algorithm performance against two gold corpora. Using multiple allows us to better understand how candidate algorithms generalize: if the algorithms' ranking changes between corpora, that ranking was a property of the corpora, not the algorithms; if it holds steady across corpora, that suggests a property of the algorithms. More than two corpora would be ideal, but we are limited to quality, available, free, and segmented corpora, which rules out most of the field (Table 1). We are left with **LCMC**, a 2003 balanced corpus of written Mandarin from Lancaster University (built as a Chinese counterpart to the FLOB/Frown corpora) and segmented to the Peking University standard, and **UD_Chinese-GSDSimp**, a 2018 Google / Universal Dependencies project based on simplified-character Wikipedia.

corpus	institution (year)	size	standard	availability	used
LCMC	Lancaster Univ. (2003)	~1M words	PKU / ICTCLAS	free, pre-segmented	✓
UD_Chinese-GSDSimp	Google / UD project (2018)	~5k sentences	CTB lineage	free (CC BY-SA)	✓
Chinese Treebank (CTB)	U. Penn / LDC (2000–)	~2M words	CTB	licensed	
SIGHAN Bakeoff (PKU, MSR)	PKU / MSRA (2005)	1–2M words	PKU / MSR	partial / licensed	

corpus	institution (year)	size	standard	availability	used
Academia Sinica (ASBC)	Academia Sinica (1996)	~5M words	Sinica (traditional)	application-gated	
BCC, CCL	BLCU / Peking Univ.	15B / ~700M chars	mixed	online query only	

Table 1: The corpora that Chinese segmenters are typically graded against in NLP research. We use the two that are both free and ship pre-segmented. Sources for each corpus are in Further reading.

The mechanical scoring of a candidate segmentation against the gold segmentation is done with a measure called **token F1**.[3] To explain, picture the characters in a phrase as beads on a string, as in Figure 2. Segmenting decides where to snip between the beads, and the resulting sub-strings are the candidate segments. A candidate segment counts as correct only if it matches a gold segment exactly, with both of its snips in the right place. **Precision** is a measure that asks: of the candidate segments produced, how many exactly matched the gold segments? **Recall** is a measure that asks: of the gold segments, how many candidates were recovered? **F1** is the harmonic mean of the two measures, and is high only when both sub-measures are high, so you can't game it by over-cutting or under-cutting.

$$\text{precision} = \frac{\text{matched words}}{\text{words you produced}}, \qquad \text{recall} = \frac{\text{matched words}}{\text{words in the key}}, \qquad \text{F1} = \frac{2\,(\text{precision})(\text{recall})}{\text{precision} + \text{recall}}.$$

In Figure 2 the measurement is illustrated and computed with the same sentence from Figure 1 to make the idea concrete.

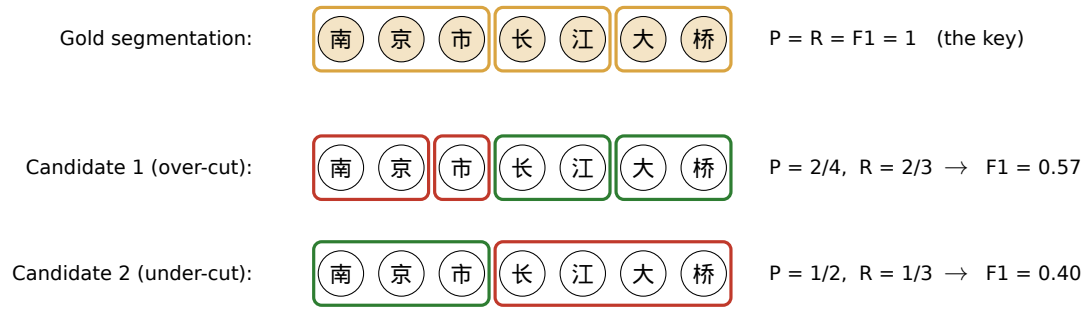

Figure 2: Illustration of the F1 measure. The top row is gold, followed by two candidate segmentations. Green boundaries mark (green) exact matches. A single misplaced boundary can spoil two words at once, which is what makes token F1 a demanding grade.

An overview of the main CWS engines

The algorithms for segmentation are called "engines". There are many available and fall into three broad families (Figure 3). **Dictionary-based** engines (e.g. `jieba`) carry a large word list and split the text into the most likely sequence of listed words, weighting candidates by frequency, with a small character-level guesser for anything unlisted. They are fast and transparent but have no context awareness whatsoever. **Statistical** engines (e.g. `pkuseg`, a CRF, and `thulac`, a structured perceptron) are trained on large amounts of pre-segmented text and label each character as the beginning, middle, or end of a word. They generalize to unseen words well, but their style is fixed to whatever corpus trained them. **Transformer** engines (e.g. `hanlp`) read the whole sentence as context before cutting,

[3] The measures here were standardized through the SIGHAN "bakeoffs" (Emerson 2005).

which lets them resolve real ambiguity by meaning but at a much higher computational cost. None of the three families strictly dominates the others; all are in use, chosen by the cost, accuracy, and deployment needs of the given task.

How about Large Language Models (LLMs) like those from Qwen and OpenAI? The massive efforts applied to the training of these models actually skip word segmentation entirely: they chop text into subword tokens by statistical frequency, so 北京大学 might become 北京 + 大学 or 北 + 京大 + 学 depending on what compressed the training data well. For an LLM doing text generation that's fine, but those pieces aren't necessarily words, so an LLM's own tokenizer is no help with human-facing CWS needs. Instead you can ask the trained LLM to segment a sentence as an explicit task. Their strong context modeling makes this a promising research direction, but without specific reinforcement learning for CWS during training, performance is unlikely to be excellent. They may become an appealing or even default option as inference costs fall, speeds increase, and NLP researchers develop specialized models and harnesses.

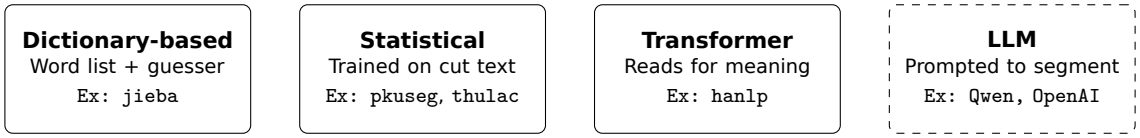

Figure 3: The three established families of Chinese segmenters, all in current use and trading off speed, data needs, and accuracy, plus the emerging LLM approach.

Establishing baseline engine performance

The remainder of this article will focus on an evaluation of CWS approaches specifically for use in Pindu. We tested the most prevalent engines from each of the families against the two gold corpora, as described above. From the dictionary-based family of engines, `jieba` is the de facto standard. From the statistical family, `pkuseg` and `thulac` are the two widely used systems. From the transformer family, `hanlp` is the most popular research-grade suite.[4] In skipping over other engines, we acknowledge a lack of exhaustiveness, but in covering the most popular engines from each of the families, we hope to characterize attainable performance adequately. In that spirit we also test a basic LLM engine: because such engines are not in common use yet, we built a simple one of our own to measure.[5]

Each of these engines also exposes important configuration choices. For example, `hanlp` offers a coarse and a fine model; `pkuseg` ships several models trained on different domains (news, web, a mixed set, and a general default); `jieba` can run with or without its new-word guesser. These are ordinary knobs a user picks among, depending on application. In our evaluation we scored every configuration for each engine against each gold corpus. Figure 4 shows the result: one row per engine, one dot per configuration, one panel per corpus.

[4] Popularity here is a proxy for community adoption rather than a formal ranking. As of May 2026, `jieba` (~35k GitHub stars) and `hanlp` (~36k) are by far the most-starred Chinese segmentation projects, while `pkuseg` (~6.7k) and `thulac` (~2.1k) are the established statistical segmenters from Peking and Tsinghua universities respectively. Repositories and papers for all four are in the Further Reading section.

[5] We use a cheap general-purpose LLM (`gpt-4.1-nano`) with a zero-shot system prompt configuration and no post-processing. For more details, see Appendix A.

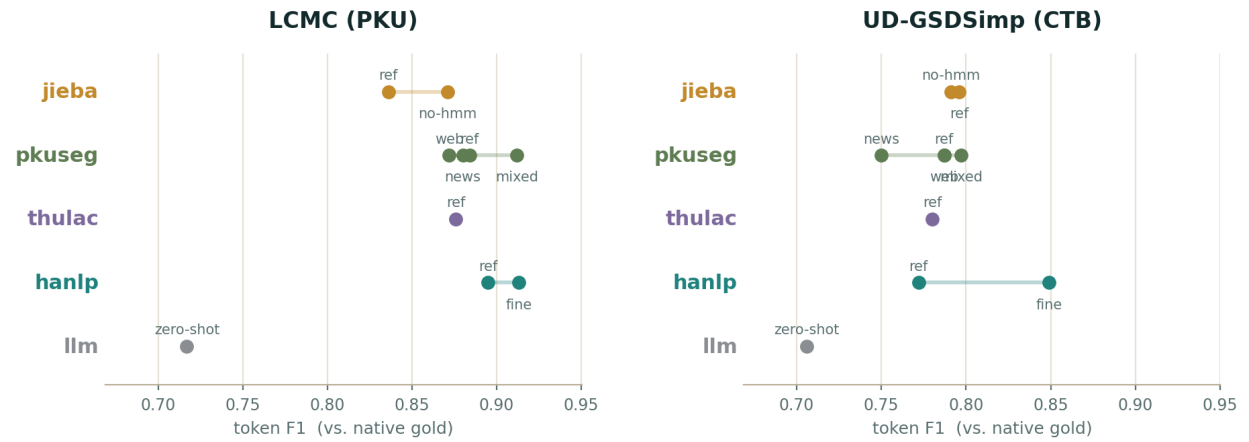

Figure 4: Baseline token F1 for each engine and configuration on both standards. An engine family's own configurations can move the score as much as switching families does. Note that `hanlp` leads and the LLM trails on both standards, while the ranking of the engines in between shifts.

Two things are visible in the chart. First, a single engine family's configuration range can cover as much ground as the differences between families: switching `hanlp` from coarse to fine moves it more than the gap between most engines, and `pkuseg`'s domain models span a wide band. Second, the ordering is not stable across the two standards. `pkuseg:news`, for instance, lands mid-pack on the print-heavy LCMC but last on the Wikipedia-based UD.

The most important conclusion from these results is just that there is not a clear "winner": different engines and different configurations will perform differently depending on the text and the standard (i.e., how the annotator decided to segment the gold corpus). The score reflects how closely an engine's configuration matches the standard it is graded against, at least as much as it reflects the engine itself. This prompts the question of which standard we should be using to select the best CWS engine and configuration for Pindu itself: is the nominal "Pindu-shaped task" more like LCMC or UD?

Developing a Pindu-aligned gold corpus

Our answer is "neither". LCMC and UD, like all the published standards in Table 1, were built for linguists and parsers, not learners. PKU-lineage corpora segment personal names into surname plus given name (吕 / 清舟), but it's probably the case that in a reading-gloss setting the whole name of the person should sit together as one segment. They also split derived words at their affixes (现代 / 化, 重要 / 性), though a learner will want the whole word as a single glossable unit. Thus, many errors against PKU-lineage corpora aren't errors at all for the Pindu use case, and ranking engines against them will lead to misleading conclusions. Pindu makes deliberate design choices that can disagree with a 1990s annotation manual.

With that in mind, we constructed a **Pindu-aligned** version of each gold corpus that applies a few rules to modify the existing annotations. In general, these rules tend to prefer larger segments for Pindu's readers. For example, one rule we add is to merge together personal names that are split; another is to recombine bound affixes like the plural 们 in 运动员们 ("athletes"). We are careful to make these changes principled: they must be driven by a corpus's own grammatical tags, not by a given segmenter's output. There are two different kinds of compounds that are left split up still in the Pindu-aligned corpus, as shown in Table 2. The first kind is genuinely ambiguous, even in Pindu: 国际射击中心 ("international shooting center") could reasonably be one chunk or three (国际 / 射击 / 中心), and we have no confident target to realign it to. The second kind is not ambiguous — we know we would want 人造卫星 ("man-made satellite") whole — but we have no principled rule that produces it. The corpora tag it with ordinary phrase syntax (in UD it parses as 人 / 造 / 卫星: a subject, a modifier, and a conjunct), so no tag-driven rule isolates it without also merging vast stretches of ordinary phrases.

In both cases we would rather leave a known gap in the gold than add a rule that overfits or mangles ordinary text. This encodes our best hypotheses about what a learner wants and leads to a sharper proxy than the published standards for our case.

standard splits	what Pindu wants	why we leave it
国际 / 射击 / 中心	Ambiguous	No clear preference
人 / 造 / 卫星	人造卫星, as one word	Clear, but no rule targets it cleanly

Table 2: Two reasons a compound stays at its native segmentation: we are unsure what a learner wants (top), or we know but have no principled rule that produces it without collateral damage (bottom).

On the UD sample, holding engine output fixed while changing the gold raises jieba's micro-averaged token F1 from 0.796 to 0.833 and pkuseg's default model from 0.787 to 0.852. The comparison changes in pkuseg's favor. On LCMC, the engines in Figure 5 keep their order under both keys, as do the jieba, pkuseg, and THULAC configurations in Figure 4.

How much does this realignment of corpora change the picture? Figure 5 scores each engine against both the original gold corpora and the Pindu-aligned gold corpora.

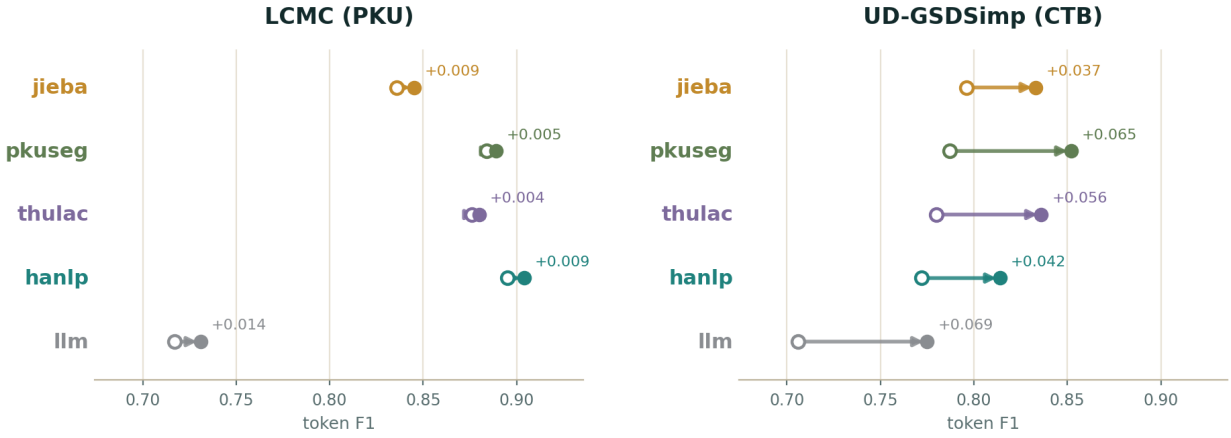

Figure 5: Each engine's token F1 on the native gold corpus (hollow) and on the Pindu-aligned gold corpus (filled), with the delta labeled.

Interestingly, the corpus modifications improve every default configuration shown in the chart (the general-purpose engines all agree with our merge rules), though the lift is uneven. Among the four conventional engines, on LCMC the gains are small (+0.004 to +0.009), while on UD they are much larger (+0.037 to +0.065). The intuition for this is that the UD's CTB-lineage standard splits far more of the affixes and names that Pindu prefers to keep whole. Realigning therefore recovers a lot of apparent "error" that was really standard mismatch. Because these rules are driven by each corpus's own tags and applied to every engine alike, the lift reflects the gold becoming fairer for a learner-facing product, not a gaming of the score for any particular engine.

Developing a Pindu-specific engine wrapper

The obvious complement to the development of a Pindu-aligned gold corpus for testing is the modification of an existing engine to create a Pindu-aligned CWS engine that reliably produces the segmentations that are useful for the Pindu software. In general that is an ambitious task, but there is low-hanging fruit that can be taken by post-processing the results from unmodified engines. The actual architecture of the segmenter in Pindu is a custom wrapper (which we call the orchestration layer)

around a stock engine. The orchestration layer applies rules after initial segmentation to improve final outputs.

The rules are deliberately small and targeted, and three are relevant on these corpora. The first locks idioms and set phrases (chengyu) as single units via a dictionary lookup, so, e.g., an engine won't cut 画蛇添足 into parts. The second merges runs of adjacent digits back into one numeral in the case that an engine over-splits them (so we get 1 9 9 1 instead of 1 / 9 / 9 / 1). The third re-glues the plural suffix 们 back onto a stem an engine has stripped it from — the 们 in 运动员 / 们 — so the derived word stays one glossable unit. We restrict the rule to 们 precisely because it is never a free-standing word; the diminutive 儿 is productive too, but it also stands alone as the noun "son" (给 / 儿), so re-gluing it by rule would fuse a genuine two-word sequence into a nonexistent 给儿. To emphasize, these rules do not re-segment the text from scratch; each corrects one specific, recurring type of error. From a performance perspective, the rules are cheap but not free. The chengyu dictionary lookup, numeral coalescing, and suffix re-merging add a roughly fixed per-token overhead independent of the base engine.

Someone who knows `jieba` might point to a lever we don't use: dictionary-based engines accept custom word lists ("dictionaries") that tune them to a specific application. So why do we post-process instead? While our chengyu preserving rule is expressible via `jieba`'s custom dictionaries, that may not be true for other engines we want to evaluate in Pindu. The orchestration layer is designed to be engine-agnostic. The case of numeral merging shows a second reason: a dictionary adds words, while our layer adds rules. A dictionary cannot encode "merge any run of adjacent digits into one numeral" — we would instead have to enumerate every number, which is impossible. The more ambitious idea, though, of taking apart an open-source segmentation engine, fine-tuning its internals, and putting it back together into a Pindu-flavored engine is something we're actively considering.

Figure 6 shows the effect that the Pindu orchestration layer has on the performance of each engine evaluated against the Pindu-aligned gold. The gap between the two points is exactly what the rules add.

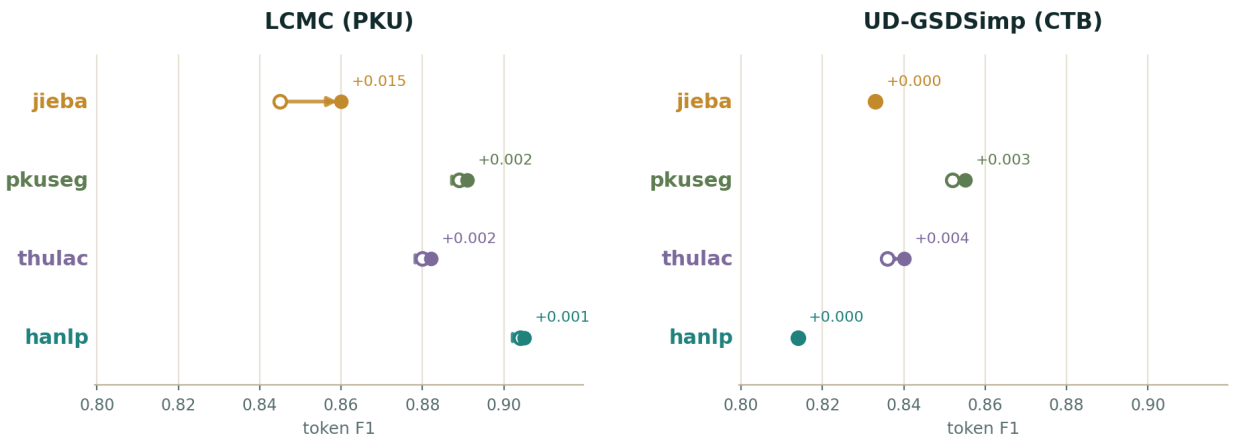

Figure 6: Each engine's token F1 on the Pindu-aligned gold corpora, bare (hollow) and wrapped in Pindu's orchestrator (filled). The rules help `jieba` on LCMC and are nearly flat everywhere else.

The results are lopsided. The rules give `jieba` a real bump on LCMC (+0.015) and nothing on UD; for `pkuseg`, `thulac`, and `hanlp` they move the score by at most +0.004. The reason is mechanical: the numeral-merge rule only fires where an engine over-splits digit runs, and only `jieba` does that, and only on LCMC, whose standard marries numbers to their units (1 9 9 1 年). The suffix re-merge only fires where an engine strands an affix, and that too is mostly `jieba`; the trained engines keep digits, idioms, and affixes intact, so there is almost nothing left to repair. That is a convenient result rather than a disappointing one: the rules pay off most for the cheap dictionary engine. They are a patch over known weaknesses of the base engine, not a general accuracy lever. The suffix re-merge does

close part of a gap we previously had no engine-side fix for — the plural 们 is unambiguous enough to re-glue by rule — but the affixes that double as free words (儿, 者, 性, 率) and the split names remain, since telling those from ordinary text needs context an engine-side rule lacks. That residual is part of why even wrapped `jieba` stays short of the trained engines.

What it all means for Pindu

To recap the evaluation so far: CWS has no objective solution and the best segmentation engine depends upon the application. Because Pindu is a unique application, we developed a unique benchmark and an orchestrator on top of the established engines to improve Pindu-specific performance. So, which is the best for Pindu? Based on the metrics we've discussed so far, that's HanLP: it gets an orchestrated joint F1 of 0.90 across the Pindu-aligned gold corpora, above any other engine. But F1 is not the whole story.

As Table 3 shows and as our initial choice of `jieba` for Pindu revealed, we care about two other features of the CWS engine when deciding which to use in Pindu: token throughput and package footprint. Together, F1, throughput, and footprint trade off against each other instead of pointing at a single best choice. The orchestrated transformer scores highest on F1, but (locally on a desktop CPU) it runs roughly 12× slower than the dictionary-based engines.[6] That mattered for the May 2026 Anki Add-On v1.16.3 deployment evaluated here, which ran segmentation locally on a user's own machine. The dictionary-based engines are the fastest and lightest but have the weakest performance. The statistical engines, tidily, are intermediate.[7]

Engine	Joint Token F1	Orchestrated Joint F1	Throughput (tok/s)	Orchestrated Throughput (tok/s)	Package Footprint
`jieba`	0.84	0.85	~98,000	~28,000	Tiny (pure Python)
`thulac`	0.86	0.86	~46,000	~20,000	Light (perceptron model)
`pkuseg` (mixed)	0.89	0.89	~37,000	~17,000	Moderate (CRF model)
`hanlp` (fine)	0.90	0.90	~2,400	~2,400	Heavy (PyTorch)
`gpt-4.1-nano`	0.75	—	~400†	—	Tiny (API)

Table 3: Joint token F1 is the mean across LCMC and UD on the Pindu-aligned gold; the orchestrated columns wrap each engine in Pindu's rules. Throughput is warm CPU segmentation, excluding model load; footprint is the rough deployment weight. †The LLM is measured differently, since the model itself is so heavy it must be served via 3rd party API. Its throughput is end-to-end API wall-clock at 16-way concurrency, bounded by network latency and rate limits, not local compute.

For Anki Add-On v1.16.3 in May 2026, after this evaluation, Pindu retained the `jieba` engine inside its orchestrator as the default and only CWS engine. Despite the lowest F1 performance, `jieba`'s tiny footprint and fast speed dominated the decision-making because that deployment included the entire package inside the add-on and used the learner's CPU. But the quantification of `jieba`'s lowest performance is comforting in an important way: the results show that its orchestrated F1 of 0.85 is only 6% less than HanLP's orchestrated F1 (0.90)! That is a small gap in accuracy for what is a very large gap in throughput and footprint.

The hosted Reader also uses jieba. As Pindu matures it may in the future offer other engines either locally or via the cloud. Maintaining a benchmark and evaluation suite like the one used for this article

[6] Note that the orchestration toll can be a large fraction of a fast engine's time budget but a rounding error against a slow one, which is why orchestration drops `jieba` from ~98k to ~28k tok/s while leaving the ~40× slower `hanlp` effectively unchanged.

[7] The LLM engine sits off the efficient frontier rather than on it: the least accurate and the slowest in practice. At least part of this is due to the naivete of our implementation (a tiny, cheap model, prompted zero-shot, with no context about the target standard), and we don't take the result as a ceiling on the basic approach itself. In the future a mature LLM segmenter may well extend the frontier as the most accurate but heaviest option.

will pay dividends as we explore those decisions. Because the Reader already segments on Pindu servers, the desktop footprint argument above does not settle its choice, and it could move along the trade-off curve towards higher accuracy but slower models. We can measure the effectiveness of new proposed rules against a Pindu-aligned gold. Good segmentation is what keeps Pindu's glossing, leveling, scheduling, and collection-crediting correct, and that is now something we can check rather than assume. That check measures agreement with our chosen reading units; whether those units improve comprehension or vocabulary learning needs a different test.

References

- Bai, X., Yan, G., Liversedge, S. P., Zang, C., & Rayner, K. (2008). Reading spaced and unspaced Chinese text: Evidence from eye movements. Journal of Experimental Psychology: Human Perception and Performance, 34(5), 1277–1287. https://doi.org/10.1037/0096-1523.34.5.1277
- Emerson, T. (2005). The second international Chinese word segmentation bakeoff. In Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing, 123–133. https://aclanthology.org/I05-3017/
- Zhang, Z., He, L., Li, Z., Zhang, L., Zhao, H., & Du, B. (2025). Segment first or comprehend first? Explore the limit of unsupervised word segmentation with large language models. arXiv:2505.19631. https://arxiv.org/abs/2505.19631

Further reading

- The **corpora** in Table 1, by source: **LCMC** — McEnery & Xiao (2004), The Lancaster Corpus of Mandarin Chinese, LREC 2004, 1175–1178 (L04-1117); **UD_Chinese-GSDSimp** — Nivre et al. (2020), Universal Dependencies v2, LREC 2020, 4034–4043 (2020.lrec-1.497; treebank); **Chinese Treebank** — Xue et al. (2005), The Penn Chinese Treebank, Natural Language Engineering 11(2), 207–238; **SIGHAN Bakeoff** — Emerson (2005), in References; **Academia Sinica (ASBC)** — Chen et al. (1996), Sinica Corpus, PACLIC 11, 167–176 (Y96-1018); **BCC** — Xun et al. (2016), The construction of the BCC Corpus in the age of big data, Corpus Linguistics 3(1), 93–109; **CCL** — Zhan, Guo & Chen (2003), The CCL Corpus of Chinese Texts, Peking University (corpus.pku.edu.cn).
- The **NLP-progress** directory tracks current Chinese word segmentation benchmarks and papers: nlpprogress.com/chinese/chinese_word_segmentation.html.
- Hai Zhao, Deng Cai, Changning Huang & Chunyu Kit (2019), Chinese Word Segmentation: Another Decade Review (2007–2017). A thorough survey of the field's shift into deep learning (arXiv:1901.06079).
- Yuanhe Tian, Yan Song, Fei Xia, Tong Zhang & Yonggang Wang (2020), Improving Chinese Word Segmentation with Wordhood Memory Networks (ACL 2020). A representative modern neural approach near the current state of the art.
- The **SIGHAN Bakeoffs** (2003 onward). The shared tasks that standardized how the field benchmarks segmenters, and a record of how much the choice of standard matters.
- The engines benchmarked here, with code: **jieba** (github.com/fxsjy/jieba); **pkuseg** — Luo et al. (2019), PKUSEG: A Toolkit for Multi-Domain Chinese Word Segmentation (arXiv:1906.11455; github.com/lancopku/pkuseg-python); **THULAC** — Tsinghua NLP (github.com/thunlp/THULAC-Python); **HanLP** — Han He (github.com/hankcs/HanLP).

Appendix A: The LLM engine

LLMs are not yet a common tool for word segmentation, so to include one in the evaluation we built a simple engine of our own. It uses a cheap, general-purpose model (`gpt-4.1-nano`) with a zero-shot prompt that asks it to split the text into words, with no examples and no mention of a target annotation standard. The implementation is deliberately basic: the goal is to measure where the approach lands today, not to show the best a model could do.

Scoring it on the same footing as the other engines takes some care, for two reasons. First, an LLM can change the text it is given — dropping a character, normalizing punctuation, inserting a stray word — which would break a boundary comparison. We avoid this by never trusting the model with the characters, only with where to cut: each word it returns is located back in the original text and the boundary is taken there. The output therefore always reconstructs the input, and a hallucinated or altered word simply fails to match and costs a point. Second, the model loses the thread on long inputs: over a passage of several thousand characters its word stream drifts out of alignment with the source and most of its boundaries are lost. We feed it in chunks of a few sentences instead (capped at about 500 characters), which keeps the alignment reliable.

On both corpora the LLM scores below every conventional engine, with a joint token F1 of 0.75 (it is the gray row in Figure 4 and Figure 5). Its precision is high — about 0.82 on tokens and 0.97 on boundaries — but its recall is low, about 0.67. It cuts less often than the gold, and the cuts it does make are usually right, so it under-segments into coarse but sensible units: it keeps names, compounds, and bound affixes whole, while gluing on grammatical particles such as 的 and 了 that every published standard separates. Its errors are mostly a matter of being too coarse for a given standard, rather than of producing nonsense.

The realignment lift is more telling than the raw score here. Moving from each corpus's native gold to the Pindu-aligned gold raises the LLM more than it raises any trained engine — by 0.069 on UD, the largest gain of any engine. A model asked to find words, untrained on any particular standard, produces something closer to what a learner wants than to a 1990s annotation manual, whereas the trained engines already match their manual by construction and so have less to gain. This is one model on one run, so we would not read much into the exact number, but the direction fits the rest of the article: the LLM is being graded against standards it was never told about.

For all that, the LLM is not a practical option for Pindu. Running the full benchmark cost only about ten cents, but the model segments at a few hundred tokens per second over the network — several times slower than the local transformer and hundreds of times slower than the dictionary engine — and it cannot run offline on a user's machine. Prompting an LLM to segment is a serious research direction when done with stronger models, better prompts, and standard-specific instructions (Zhang et al., 2025); the version here is intentionally naive, included only to place the approach on the same axes as the engines we actually use.