StrawberryBench: Three R's and Counting — Has the Strawberry Problem Been Solved?

Florian Leuerer
`flo@agenticage.to`

February 18, 2026

Abstract

The ability to count individual characters within words is a seemingly trivial skill, yet it has proven surprisingly difficult for large language models (LLMs). This limitation was brought into public consciousness in 2023 when ChatGPT famously failed to count the occurrences of the letter *r* in the word strawberry—the correct answer being three. We introduce **StrawberryBench**, a focused benchmark of 847 carefully constructed letter-counting questions spanning seven difficulty tiers: easy (short words, 3–6 characters), medium (7–12 characters), hard (13+ characters), sentence-level phrases, paragraph-length passages (100+ characters), proper names, and foreign-language words. We evaluate 14 frontier LLMs via OpenRouter in zero-shot fashion. Our results reveal a striking 53-percentage-point spread: **Gemini 3 Pro and Grok 4 each achieve 99.7% accuracy**—but no model reaches a perfect score—while MiMo-V2-Flash scores only 46.6%. We further document systematic failure modes including off-by-one counting errors, a dramatic paragraph-level accuracy cliff for many models, and a strong zero-count bias in which models identify absent letters far more reliably than they count present ones.

1. Introduction

In late 2023, a viral exchange exposed an unexpected weakness in ChatGPT: when asked "How many *r*'s are in strawberry?", the model confidently—and incorrectly—replied "two". The correct answer is *three* (`strawberry`). This single anecdote sparked widespread discussion about a fundamental limitation of autoregressive language models: their inability to reliably reason at the character level.

The root cause is tokenization. Byte-Pair Encoding (BPE) [7], the dominant tokenization scheme in modern LLMs, decomposes text into sub-word units. The word `strawberry`, for instance, may be tokenized as [`str`|`aw`|`berry`]—three tokens, none of which isolates the letter *r* as a distinct symbol. Counting characters therefore requires the model to *look inside tokens*, a form of sub-token reasoning that is not directly supervised during standard autoregressive pretraining.

It is important to note that the original failure—and the vast majority of letter-counting complaints reported by users—occurred in AI *products* such as ChatGPT, Gemini, or Claude.ai, not through raw API calls. In these products the user has no control over the system prompt, temperature, model version, or routing: the provider may silently serve a smaller or cheaper model, apply safety filters that alter responses, or enable code-execution tools that bypass the limitation entirely. Complaints about letter-counting are therefore often product failures, not necessarily model failures. StrawberryBench is designed to cut through this ambiguity: by calling every model directly via its API under identical, controlled conditions, we measure the model's *intrinsic* character-level counting ability, independent of any product-layer variables.

Despite the viral attention this failure received, systematic and reproducible evaluation of this capability remains limited. Existing work on LLM evaluation focuses on reasoning [9, 11], code [1], and language understanding [12]. Several studies examine LLM character-level understanding [2, 4, 8] and benchmark character-level task performance [10], but no prior work focuses specifically on letter *counting* across a controlled difficulty spectrum on a broad set of frontier models.

We fill this gap with **StrawberryBench**, which makes the following contributions:

1. A dataset of 847 letter-counting questions across seven difficulty tiers (easy, medium, hard, sentence, paragraph, names, foreign), with ground truth computed deterministically via Python `str.count()`.

2. A reproducible evaluation harness that calls models via the OpenRouter API with two question templates per tier and exact-match scoring with fuzzy number-word parsing.

3. A zero-shot evaluation of 14 frontier models, revealing a 53-percentage-point spread in accuracy between the best (Gemini 3 Pro and Grok 4, 99.7%) and worst (MiMo-V2-Flash, 46.6%) performers.

4. Analysis of systematic failure modes including off-by-one errors, paragraph-level context collapse, and a strong zero-count bias.

2. Background and Related Work

Tokenization and character-level processing. BPE tokenization [7] was introduced to balance vocabulary size and coverage of rare words. A consequence of sub-word tokenization is that individual characters are not first-class citizens in the model's input representation. Transformer self-attention operates over tokens, not characters, so counting a character that spans multiple token positions (e.g., the letter *r* appearing in `str` and `berry`) requires the model to implicitly reconstruct character boundaries from its internal representations—a capability that emerges inconsistently, if at all. Hiraoka and Inui [6] provide direct evidence: token embeddings carry almost no information about individual characters beyond the initial letter, and the model must reconstruct exact spelling in the deeper transformer layers from contextual signals alone.

Compositional and character-level tasks. Dziri et al. [3] demonstrate that transformers systematically struggle with compositional tasks that require tracking discrete symbolic operations. Character-level counting is a simple instance of this broader class of challenges. Prior benchmarks such as BIG-bench [9] include orthographic tasks (e.g., word length estimation), but do not isolate counting with the controlled difficulty ramp we introduce here.

Letter counting in LLMs. Fu et al. [4] investigate why LLMs fail at letter counting and trace the root cause analytically to BPE tokenization: when a letter is split across token boundaries it is invisible to the model's next-token prediction head. Shin and Kaneko [8] extend this analysis, showing that LLMs broadly lack understanding of the character composition of words—not just for repeated letters but for any character-level linguistic property. Cosma et al. [2] further show that character-level understanding emerges only slowly during training and is fundamentally constrained by vocabulary size and the average number of characters per token. Uzan and Pinter [10] present CharBench, a broader benchmark evaluating how tokenization affects performance across a range of character-level tasks—a complementary direction to the single-task depth we pursue here. Our work is complementary: rather than providing an analytical account of *why* models fail, we provide a large-scale, multi-model benchmark that quantifies *how much* models fail across a controlled difficulty spectrum, enabling ranking and comparison of 14 frontier systems on a common axis.

LLM evaluation. General-purpose leaderboards aggregate pairwise human preferences across diverse tasks and are widely used as capability rankings. As we show, such rankings are a weak predictor of StrawberryBench accuracy—motivating targeted benchmarks for specific capability classes. Instruction-following evaluation [13] is the closest related work, as it also tests whether models comply with precise formatting constraints; our work focuses on *factual accuracy* rather than format compliance.

3. The StrawberryBench Dataset

3.1. Task Definition

Each example consists of:

- **word**: a word, name, or phrase (e.g., `strawberry`)
- **letter**: a lowercase letter (e.g., `r`)
- **question**: a natural language question in one of two templates
- **answer**: the integer count, computed via `word.lower().count(letter)`

Each difficulty tier uses two question templates to reduce surface-form memorization. Templates vary by tier:

Tier	Templates (T0 / T1)
Easy/Med/Hard	*How many times does '{l}' appear in the word '{w}'?* *In the word '{w}', how many '{l}'s are there?*
Sentence	*How many times does '{l}' appear in the phrase '{w}'?* *In the phrase '{w}', how many '{l}'s are there?*
Paragraph	*How many times does '{l}' appear in the following text: '{w}'?* *In the following text: '{w}', how many '{l}'s are there?*
Names	*How many {l}'s are in the name '{w}'?* *In the name '{w}', how many times does the letter '{l}' appear?*
Foreign	*How many times does '{l}' appear in the {lang} word '{w}'?* *In the {lang} word '{w}', how many '{l}'s are there?*

({l} = letter, {w} = word/phrase, {lang} = language name)

The names tier intentionally uses the original viral question format ("How many r's are in the name strawberry?") as template T0, making it the closest analogue to the anecdote that motivated this work.

3.2. Difficulty Tiers

Table 1: StrawberryBench dataset statistics by difficulty tier.

Tier	**Word length**	**Examples**	**Zero-count**	**Example word**
Easy	3–6 chars	183	13%	`apple, bee, fuzzy`
Medium	7–12 chars	228	11%	`strawberry, committee`
Hard	13+ chars	146	10%	`accommodation, embarrassment`
Sentence	phrase	46	13%	*the library closes at nine…*
Paragraph	100+ chars	55	9%	*the ambitious astronaut…*
Names	proper name	112	11%	*Jennifer, Christopher…*
Foreign	non-English	77	12%	*Weltanschauung, bouillabaisse…*
Total		**847**	**11.1%**	

The dataset is seeded for reproducibility (seed = 42). For each word, we sample up to four present letters (letters that actually appear, with count > 0) and stochastically include one absent letter (count $= 0$) with probability 0.45. This yields an overall zero-count rate of 11.1%, a deliberate design choice to probe whether models recognize when a letter is entirely absent.

3.3. Word Selection

Words and phrases were drawn from four sources:

- **Common English vocabulary**: everyday words with known repetition patterns (e.g., `committee`, `embarrassment`, `strawberry`).
- **Orthographically challenging words**: words with many repeated letters or deceptive structure (e.g., `mississippi`, `accommodation`, `onomatopoeia`).
- **Proper names**: first names chosen for tricky repeated-letter patterns (e.g., *Annabelle*, *Christopher*, *Maximilian*).
- **Foreign-language words**: non-English words from German, French, Spanish, Italian, Dutch, Welsh, Finnish, and Portuguese, selected for dense or unusual letter patterns (e.g., *Weltanschauung*, *bouillabaisse*, *verantwoordelijkheid*).

Sentence and paragraph items use original everyday phrases and prose passages composed to contain high letter frequencies. All ground-truth answers are deterministic Python computations; no human annotation is required.

4. Experimental Setup

4.1. Models

We evaluate 14 frontier models as of February 2026 (Table 2). The selection covers six major providers—Anthropic, OpenAI, Google, xAI, Alibaba, and NVIDIA—as well as smaller players including Zhipu AI, Moonshot AI, MiniMax, DeepSeek, and Xiaomi. This breadth allows us to study whether letter-counting ability clusters by provider, scale, or training approach.

4.2. Prompt Protocol and Controlled Conditions

A key design goal of StrawberryBench is to measure *model capability*, not product behaviour. We therefore hold all non-model variables fixed: every model receives an identical system prompt instructing it to "respond with ONLY the integer count and nothing else"; temperature is set to 0 for reproducibility; no tools, retrieval, or code execution are enabled; and no in-context examples are provided. All calls go through OpenRouter to a single, pinned model version—no auto-routing. This isolation means our results cannot be attributed to differences in system instructions, sampling parameters, or provider-side post-processing, and directly reflect each model's intrinsic character-level reasoning ability.

4.3. Scoring

We use **exact-match** accuracy on the integer count. To handle models that return written-out numbers ("three" instead of "3"), we apply a fuzzy parser that maps English number words 0–20 to integers. When a response contains multiple numbers or cannot be parsed directly, we use a fast LLM extraction step (Gemini 3 Flash) to identify the model's final intended answer. Models that ultimately return an unparseable string are scored as incorrect.

All API calls are made through OpenRouter, ensuring a consistent inference endpoint across all providers. Each model processes 847 questions with 8 concurrent threads.

5. Results

5.1. Overall Accuracy

Results are summarized in Table 2. The spread across models is dramatic: Gemini 3 Pro and Grok 4 both achieve **99.7%** on all 847 questions, yet neither reaches a perfect score. MiMo-V2-Flash scores only 46.6%—a 53-percentage-point gap. Notably, models that rank highly on general-purpose leaderboards can place anywhere in this distribution, confirming that character-level counting is an orthogonal capability axis.

5.2. Performance by Difficulty Tier

A consistent pattern emerges across models: accuracy degrades as input length increases, with the paragraph tier being universally the most challenging (Figure 2). The *rate* of degradation, however, varies substantially:

- **Gemini 3 Pro** is the only model to achieve 100% on both the sentence and paragraph tiers, making it uniquely robust to long-context counting.
- **Grok 4** achieves 100% on easy, medium, hard, names, and foreign tiers but slips slightly on sentence (97.8%) and paragraph (96.4%), suggesting a mild degradation under multi-word contexts.
- **Claude Opus 4.6** also achieves 100% on the sentence tier and 92.7% on paragraphs, placing 8th overall despite perfect sentence-level performance.
- **MiniMax M2.5** exhibits a striking cliff: 100% on the medium tier but only 52.7% on paragraphs and 80.4% on sentences, suggesting a special code path for short, isolated words that breaks down under longer-context demands.
- **GPT-5.2** shows a surprisingly weak paragraph score (58.2%) relative to its otherwise strong performance (93.6% overall), falling behind Nemotron 3 Nano (87.3%) and Claude Opus 4.6 (92.7%) on that tier.
- **Names and foreign** tiers are generally easier than paragraph for most models—even Gemini 3 Flash scores 97.3% on names despite only 89.5% overall—suggesting that proper names and isolated foreign words do not add difficulty beyond their character length.

6. Analysis

6.1. Three Failure Archetypes

Analysing the 1,373 incorrect answers across all models reveals three qualitatively distinct failure archetypes.

Off-by-one near-misses. The dominant failure mode across high-performing models is a ś1 miscounting error. Models in this cluster include Claude Opus 4.6 (OBO rate: 93.5% of errors), GPT-5.2 Codex (85.7%), GPT-5.2 (83.3%), and Gemini 3 Flash (84.3%). These models engage in genuine character-level reasoning but occasionally slip by a single character under load. The off-by-one signature is consistent with a tokenization-boundary artefact: the model tracks letter occurrences at the sub-word level but misattributes one occurrence at a token boundary.

Systematic overcounters. Claude Sonnet 4.6 (222 errors; 89% overcounts) and MiMo-V2-Flash (452 errors; 91% overcounts) show a consistent bias toward reporting counts *higher* than the true value. Their non-zero accuracy (71.2% and 43.7% respectively) is far below their zero-count accuracy (94.7% and 70.2%), indicating that they can identify when a letter is absent but consistently add phantom occurrences when the letter is present.

Context-length collapse. DeepSeek V3.2 (54.5% overall) and MiMo-V2-Flash (46.6%) both collapse dramatically on longer inputs: DeepSeek scores only 18.2% on the paragraph tier and 26.1% on sentences, versus 76.5% on easy single words. The same pattern holds for MiMo: 84.7% on easy words but 17.4% on sentences and 18.2% on paragraphs. This length-dependent degradation—absent in top models— suggests that character-level attention fails to scale with input length in these architectures.

6.2. Off-by-One Error Analysis

For models that produce valid integer answers, the off-by-one rate (OBO%) measures near-correctness. Table 3 shows the distribution, which reveals a clear bimodal structure:

Models at the top of the leaderboard make predominantly near-miss errors—Gemini 3 Pro, Grok 4, and Kimi K2.5 achieve 100% OBO, meaning *every* one of their rare failures is exactly ± 1. The collapse models (DeepSeek, MiMo) have hundreds of errors spread more broadly. A high OBO rate thus signals genuine-but-imperfect counting, while a low OBO rate combined with

Table 2: Zero-shot accuracy on StrawberryBench by model and difficulty tier. Models are sorted by overall accuracy (descending). **Bold** = best per column (ties included). "Para." = paragraph; "For." = foreign.

Rank	Model	Provider	Overall	Easy	Medium	Hard	Sent.	Para.	Names	For.
1	Gemini 3 Pro	Google	**99.7**	**100.0**	99.6	99.3	**100.0**	**100.0**	99.1	**100.0**
2	Grok 4	xAI	**99.7**	**100.0**	**100.0**	**100.0**	97.8	96.4	**100.0**	**100.0**
3	Kimi K2.5	Moonshot AI	98.9	99.5	99.1	98.6	**100.0**	96.4	98.2	**100.0**
4	Qwen3.5 397B	Alibaba	98.9	**100.0**	99.6	99.3	95.7	94.5	**100.0**	97.4
5	GLM-5	Zhipu AI	98.0	99.5	99.1	98.6	95.7	85.5	99.1	98.7
6	Nemotron 3 Nano	NVIDIA	96.8	98.4	99.6	98.6	93.5	87.3	96.4	90.9
7	GPT-5.2 Codex	OpenAI	96.7	98.9	98.2	99.3	93.5	74.5	97.3	98.7
8	Claude Opus 4.6	Anthropic	96.3	**100.0**	98.2	90.4	**100.0**	92.7	93.8	97.4
9	MiniMax M2.5	MiniMax	94.8	98.4	**100.0**	99.3	80.4	52.7	96.4	98.7
10	GPT-5.2	OpenAI	93.6	97.3	95.2	97.3	84.8	58.2	97.3	98.7
11	Gemini 3 Flash	Google	89.5	96.7	94.7	89.0	71.7	47.3	97.3	87.0
12	Claude Sonnet 4.6	Anthropic	73.8	95.6	86.4	62.3	69.6	58.2	57.1	44.2
13	DeepSeek V3.2	DeepSeek	54.5	76.5	57.0	48.6	26.1	18.2	60.7	40.3
14	MiMo-V2-Flash	Xiaomi	46.6	84.7	52.2	29.4	17.4	18.2	36.6	24.7

Table 3: Off-by-one (OBO) error rates for all 14 models. **Bold** = worst per column.

Model	Errors	OBO%
Kimi K2.5	9	100.0%
Gemini 3 Pro	3	100.0%
Grok 4	3	100.0%
Claude Opus 4.6	31	93.5%
Claude Sonnet 4.6	222	90.5%
GPT-5.2 Codex	28	85.7%
Gemini 3 Flash	89	84.3%
GPT-5.2	54	83.3%
Nemotron 3 Nano	27	77.8%
Qwen3.5 397B	9	77.8%
GLM-5	17	76.5%
DeepSeek V3.2	385	68.6%
MiMo-V2-Flash	**452**	61.5%
MiniMax M2.5	44	**59.1%**

many errors signals systematic miscount rather than near-miss.

6.3. The Code Hypothesis

GPT-5.2 Codex (96.7%) outperforms the base GPT-5.2 (93.6%) by 3.1 pp overall and is substantially stronger on the hard tier (99.3% vs. 97.3%). This is consistent with code training improving structured, rule-following tasks [1]: character-by-character counting is analogous to iterating over a Python string. The Codex advantage narrows on the paragraph tier (74.5% vs. 58.2%), suggesting that longer-context character tracking does benefit from code training but cannot fully compensate for context-length effects.

6.4. Zero-Count Bias

Questions where the target letter is absent (answer = 0, 11.1% of the dataset) reveal a systematic asymmetry (Table 4): nearly every model is substantially more accurate on these than on positive-count questions.

Table 4: Accuracy on zero-count (absent letter) vs. non-zero questions. Gap = zero% − non-zero%.

Model	Zero	Non-zero	Gap
DeepSeek V3.2	100.0%	48.9%	+51.1
MiMo-V2-Flash	70.2%	43.7%	+26.5
Claude Sonnet 4.6	94.7%	71.2%	+23.5
Gemini 3 Flash	100.0%	88.2%	+11.8
GPT-5.2	100.0%	92.8%	+7.2
Claude Opus 4.6	100.0%	95.9%	+4.1
Qwen3.5 397B	98.9%	98.9%	≈0

DeepSeek V3.2 shows the most extreme case (+51.1 pp): it identifies letter absences perfectly yet answers only 48.9% of non-zero questions correctly. The sole model near parity is Qwen3.5 397B. The consistent positive bias supports a *response-conservatism* hypothesis: confirming absence requires only recognising a mismatch, whereas affirming a count requires constructing and verifying a specific integer.

6.5. Letter Difficulty

Aggregating across all 14 models by target letter reveals a consistent hierarchy. Letters that appear frequently in the dataset words and create ambiguous token boundaries are hardest: `q` (78.6%), `n` (84.7%), `r` (84.8%), and `f` (84.8%). The difficulty of `r`—the letter in the eponymous strawberry example—is not coincidental: it appears multiple times in dense consonant clusters that BPE tokenizers merge (`str`, `rry`). Letters with low frequency or visually distinctive forms are easiest: `z` (96.4%), `b` (93.9%), and `a` (92.4%).

6.6. Template Sensitivity

The two question templates per tier differ in word order: Template 0 leads with the letter ("How many times does

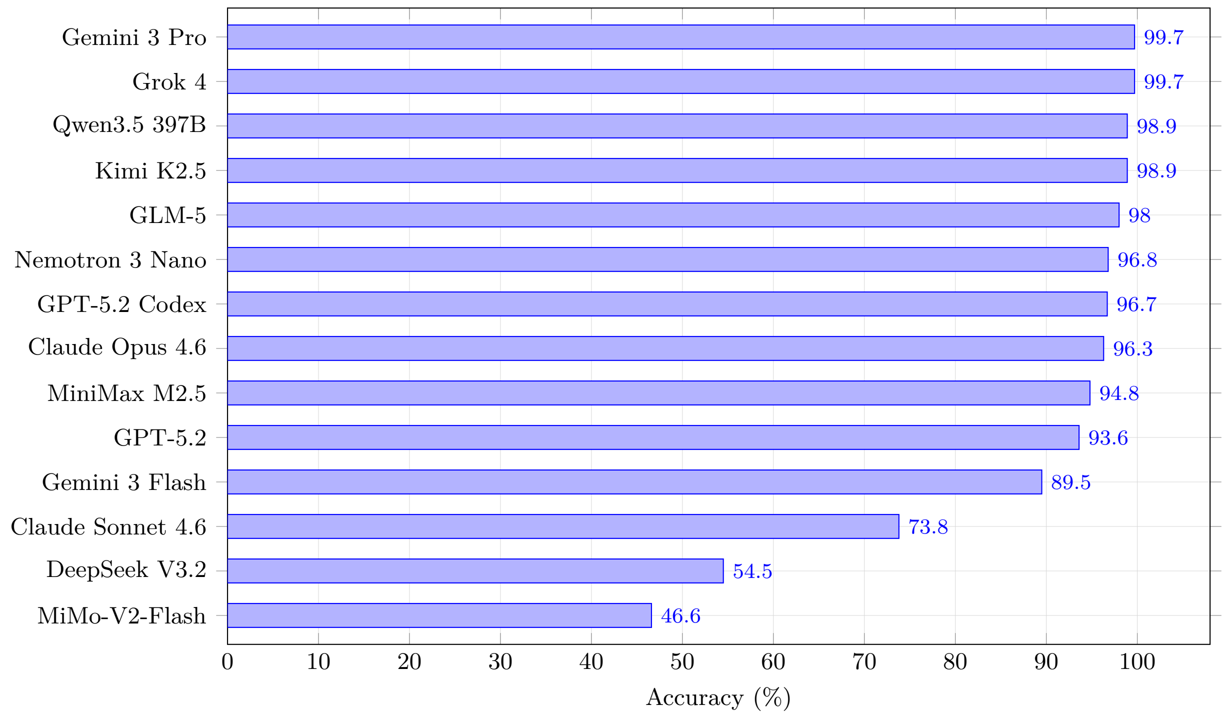

Figure 1: Zero-shot accuracy on StrawberryBench (all 847 questions). Gemini 3 Pro and Grok 4 are tied at the top with 99.7%; no model achieves a perfect score.

x appear in *word*?"), while Template 1 leads with the word ("In *word*, how many x's are there?"). Averaged across all models, Template 1 achieves 90.2% vs. 86.5% for Template 0—a 3.7 pp advantage. The effect is highly model-dependent: DeepSeek V3.2 shows a dramatic 39.5-point gap (73.6% vs. 34.1%), whereas top-performing models (Gemini 3 Pro, Grok 4) show essentially no sensitivity (<1 pp). Template effects imply that reported accuracy reflects not only character-level reasoning ability but also prompt formulation, and that weaker models may be substantially improved by phrasing choice alone.

6.7. The Hardest Questions

The hardest questions—answered correctly by only 6 of 14 models—are paragraph-level items with high target-letter frequencies: "*mississippi and massachusetts are notoriously difficult to spell...*", letter a, expected answer 9; and "*the ambitious astronaut carefully adjusted...*", letter n, expected answer 6. Seven of the ten hardest questions are from the paragraph tier, confirming that long-context counting is the frontier challenge. The three hardest single-word questions all feature a single occurrence of a common letter embedded in a dense, heavily-tokenized cluster.

6.8. Long-Context Analysis

The paragraph tier shows the greatest spread between models (18.2% to 100%), while the easy tier shows the least (76.5% to 100%). This mirrors findings in Dziri et al. [3]: compositional task difficulty scales with the number of discrete operations required. Counting a letter across a 130-character passage requires roughly 20–30× more tracking steps than across a 6-character word, and errors compound for weaker models.

Within the paragraph tier, accuracy correlates strongly and inversely with the expected count: zero-count paragraphs score 94.3% while paragraphs with 9 or more target letters score only 43–50%—the highest-count examples are universally the hardest. The same trend holds for sentences: zero-count sentences score 94.0% while sentences with five or more target letters score 64–70%.

6.9. Price-Quality Analysis

Table 5 lists OpenRouter input and output prices for all 14 models alongside their StrawberryBench accuracy, sorted by combined price per million tokens. Three models form the *Pareto frontier* for this task—no other model is simultaneously cheaper and more accurate:

- **Nemotron 3 Nano 30B** ($0.25/M combined)

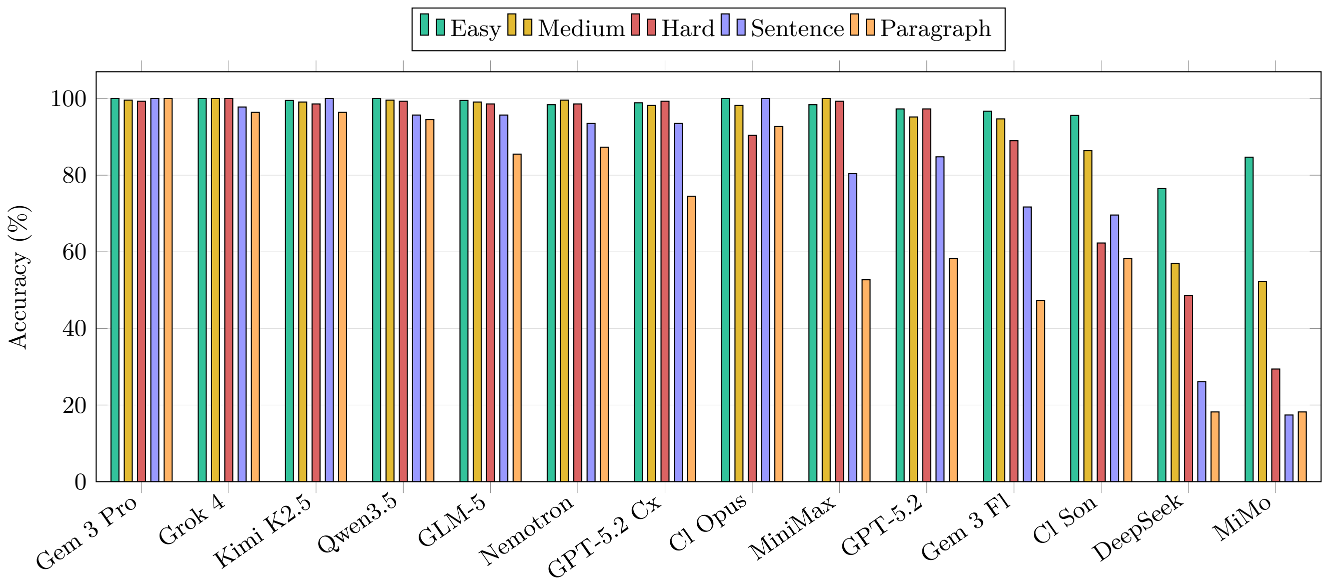

Figure 2: Accuracy by difficulty tier for all 14 models (easy, medium, hard, sentence, and paragraph). Names and foreign tiers are discussed in text. Gemini 3 Pro achieves 100% on both sentence and paragraph. MiniMax M2.5 shows an extreme medium/paragraph gap.

achieves 96.8% for a fraction of any premium model's cost.

- **Qwen3.5 397B** ($1.15/M) reaches 98.9%—matching Kimi K2.5 at one-third the price ($3.23/M).
- **Gemini 3 Pro** ($14.00/M) reaches the top accuracy of 99.7% at a lower price than Grok 4 ($18.00/M), which achieves the same score.

The Pareto frontier reveals strongly diminishing returns: the jump from $0.25/M to $1.15/M buys 2.1 pp of accuracy, while spending a further $12.85/M to reach Gemini 3 Pro gains only 0.8 pp more—a 38× drop in marginal return per dollar.

The most notable inefficiency is **Claude Opus 4.6** ($30.00/M), which scores 96.3%—lower than both Qwen3.5 397B (98.9%, 26× cheaper) and Nemotron 3 Nano (96.8%, 120× cheaper). **Claude Sonnet 4.6** is the worst-value model overall: at $18.00/M it costs as much as Grok 4 yet delivers only 73.8% accuracy, a 25.9 pp gap for the same price.

Table 5: OpenRouter pricing and StrawberryBench accuracy for all 14 models, sorted by combined price per million tokens. **P** = Pareto-optimal (not dominated on price and accuracy).

Model	**In $/M**	**Out $/M**	**Acc. %**	**P**
Nemotron 3 Nano 30B	0.05	0.20	96.8	✓
MiMo-V2-Flash	0.09	0.29	46.6	
DeepSeek V3.2	0.26	0.38	54.5	
Qwen3.5 397B	0.15	1.00	98.9	✓
MiniMax M2.5	0.30	1.10	94.8	
GLM-5	0.30	2.55	98.0	
Kimi K2.5	0.23	3.00	98.9	
Gemini 3 Flash	0.50	3.00	89.5	
Gemini 3 Pro	2.00	12.00	99.7	✓
GPT-5.2	1.75	14.00	93.6	
GPT-5.2 Codex	1.75	14.00	96.7	
Grok 4	3.00	15.00	99.7	
Claude Sonnet 4.6	3.00	15.00	73.8	
Claude Opus 4.6	5.00	25.00	96.3	

7. Further Research

Code execution as a solution. The most direct path to reliable letter-counting accuracy is to treat the task as a program synthesis problem rather than a language modeling problem. Program-aided Language Models [5] delegate arithmetic and symbolic reasoning to an interpreter, with the LLM generating code and an external runtime executing it. Applied to StrawberryBench, a model could emit a one-liner such as `word.lower().count(letter)` and let Python evaluate it—guaranteeing exact answers regardless of tokenization.

Few-shot and chain-of-thought prompting. All models in this study were evaluated zero-shot. Few-shot examples that spell out character-by-character counting (e.g., “s-t-r-a-w-b-e-r-r-y: the letter *r* appears at positions 3, 9, and 10 — count: 3”) may substantially improve accuracy for weaker models by teaching the intermediate decomposition strategy [11]. Future work should measure the gap between zero-shot and chain-of-thought performance across all difficulty tiers, especially paragraph and sentence.

Character-aware tokenization. Recent work on modifying token internal structure [14] demonstrates that enriching token embeddings with character-level information can improve performance on orthographic tasks. Evaluating architectures with explicit character encoders (e.g., ByT5, Canine) on StrawberryBench would directly test whether tokenization-free or hybrid approaches eliminate the failure modes we document.

Non-Latin scripts. The benchmark includes Latin-script foreign words. Extending it to agglutinative languages with non-Latin orthographies (e.g., Arabic, Hebrew, Thai) or logographic scripts (Chinese, Japanese) would reveal whether character-counting difficulty is a universal consequence of sub-word tokenization or specific to Latin orthography.

Longitudinal tracking. StrawberryBench is designed as a lightweight diagnostic. Re-running it on successive model versions (e.g., successive Claude or GPT releases) would allow model developers to track whether character-level reasoning improves monotonically or regresses between releases.

8. Limitations

Zero-shot only. We evaluate all models zero-shot; few-shot and chain-of-thought strategies are reserved for future work and may substantially improve results for weaker models.

Single strategy. We instruct models to return only an integer. Models that are inherently verbose may be penalized for returning well-formatted reasoning alongside a correct answer, though our fuzzy parser and LLM-extraction fallback mitigate this for most cases.

English-centric. Although the benchmark includes foreign-language words, the words are all in Latin script and the questions are posed in English. Letter-counting in scripts without subword tokenization (e.g., ideographic languages) would require a separate study.

Model versions. Model versions are fixed as of February 2026 and accessed via OpenRouter; results may differ with provider-direct endpoints or future model updates.

Model capability vs. product experience. StrawberryBench intentionally measures raw model capability via direct API calls under controlled conditions. Real-world AI products (ChatGPT, Gemini, Claude.ai, etc.) add layers the user cannot control—system prompts, safety filters, code-execution tools, and automatic model routing that may silently serve a different, cheaper variant. The viral “strawberry” failure originated in the ChatGPT *product*, not a raw API call. A user interacting with such a product today may therefore still encounter letter-counting errors even if the underlying flagship model scores 99.7% on StrawberryBench, because the product may route their query to a smaller model, apply a system prompt that alters counting behaviour, or—conversely—use a code interpreter that trivially solves the task. Our results establish what is achievable at the model level; the gap to product-level experience is a separate, important open question.

Prompts and sampling parameters. Reported accuracy reflects not only character-level reasoning ability but also the specific prompt engineering and inference configuration. Our Template Sensitivity analysis already demonstrates a 3.7 pp average accuracy gap between question phrasings—and up to 39.5 pp for individual models. Analogous sensitivity exists for system-prompt stringency and temperature (fixed to 0 here for reproducibility). Absolute accuracy numbers should therefore be interpreted as valid for our specific configuration, not as fixed properties of the underlying model.

9. Conclusion

We introduced StrawberryBench, a focused benchmark measuring whether large language models can count letter occurrences—a task that exposes the character-level reasoning gap induced by sub-word tokenization. Our zero-shot evaluation of 14 frontier models on 847 questions across seven difficulty tiers reveals a 53-percentage-point spread, with Gemini 3 Pro and Grok 4 each achieving 99.7% and MiMo-V2-Flash scoring 46.6%. Crucially, *no model achieves a perfect score*, with the remaining errors concentrated in the paragraph tier—long-context character tracking remains an unsolved challenge even for the strongest models.

The benchmark surfaces failure modes invisible to aggregate capability rankings: models that score highly on general-purpose leaderboards can land anywhere in the StrawberryBench distribution. The dominant failure mode for capable models is an off-by-one error (77–100% of mistakes for 10 of the 14 models evaluated), while weaker models exhibit systematic overcounting or context-length collapse.

We hope StrawberryBench serves as a lightweight diagnostic tool for model developers to track character-level reasoning across model generations—an important capability for downstream tasks including spell-checking, anagram solving, ciphers, and any application requiring fine-grained orthographic awareness.

StrawberryBench is available at https://www.strawberrybench.fyi (leaderboard), https://huggingface.co/datasets/floleuerer/strawberry-bench (dataset), and https://github.com/floleuerer/strawberry-bench (code).

Acknowledgments

Declaration of AI Assistance. The author used Claude Code (Anthropic) as a writing and coding assistant during the preparation of this manuscript. All research decisions, results, and conclusions are the author's own.

References

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv:2107.03374*, 2021.

[2] A. Cosma, S. Ruseti, E. Radoi, and M. Dascalu. The strawberry problem: Emergence of character-level understanding in tokenized language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. arXiv:2505.14172.

[3] N. Dziri, X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, P. West, C. Bhagavatula, R. Le Bras, J. D. Hwang, S. Sanyal, S. Welleck, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, 2023. NeurIPS 2023 Spotlight. arXiv:2305.18654.

[4] T. Fu, R. Ferrando, J. Conde, C. Arriaga, and P. Reviriego. Why do large language models (LLMs) struggle to count letters? *arXiv:2412.18626*, 2024.

[5] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. arXiv:2211.10435.

[6] T. Hiraoka and K. Inui. Spelling-out is not straightforward: LLMs' capability of tokenization from token to characters. *arXiv:2506.10641*, 2025.

[7] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1162. arXiv:1508.07909.

[8] A. Shin and K. Kaneko. Large language models lack understanding of character composition of words. *arXiv:2405.11357*, 2024. ICML 2024 Workshop on Large Language Models and Cognition.

[9] A. Srivastava, A. Rastogi, A. Rao, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. arXiv:2206.04615.

[10] O. Uzan and Y. Pinter. CharBench: Evaluating the role of tokenization in character-level tasks. *arXiv:2508.02591*, 2025.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. arXiv:2201.11903.

[12] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. arXiv:1905.07830.

[13] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv:2311.07911*, 2023.

[14] X. Zhu, Z. Zhao, Z. Zhang, Y. Liu, Q. Shen, F. Liu, Y. Kuang, J. He, and C. Liu. Enhancing character-level understanding in LLMs through token internal structure learning. *arXiv:2411.17679*, 2024.