Reverse-Engineering the OpenAI's GPT-5 Tokenizer: What 200,000 Tokens Reveal About AEO/GEO

Metehan Yesilyurt

AI Search & Information Retrieval Researcher · metehan.ai

Published: February 13, 2026 •

Canonical: https://metehan.ai/blog/reverse-engineering-the-gpt-5-tokenizer-aeo-geo/

Hi everyone! Here is my newest post of the February. I needed to think in Turkish first, then wrote it. It's a long post. I also used AI to rewrite some sections with better explanations! And as always, this is experimental. *I was in Dubai in the last week and it was all amazing, thanks to Visi Summit[1]! The next stop is Munich, SMX[2].*

A deep technical analysis of o200k_base, the tokenizer behind GPT-4o, GPT-5, o1, o3, and o4-mini. You may ask do I know every technical detail below? No, I read a lot, research a lot. Think like code & structure understanding.

Methodology & Limiations

Limitations:

- (1) BPE token rank correlates with but does not equal frequency in the training corpus.
- (2) Token-level analysis cannot determine model behavior, which is shaped by pre-training, fine-tuning, and RLHF.
- (3) Sections marked 'experimental' or 'just a guess' contain hypotheses that have not been empirically validated.
- (4) Cross-references with RESONEO's architecture analysis involve interpretation and may not reflect actual system design.
- (5) The relationship between token atomicity and hallucination risk is hypothesized but not empirically tested.
- (6) Understanding the tokenizer ≠ understanding model behavior . The tokenizer is just the input encoding layer, and above it sit the embedding layer, 100+ transformer layers, attention mechanisms, RLHF, and tool-use fine-tuning. Do not start any optimization before understand it deeply.

Let's start!

Can you imagine, what's interesting at the beginning? Some brands and interesting platforms are single tokens. Like Google, like Bentley, like Amazon, like Forbes, like Reddit or reddit or subreddit.

Figure

1. Introduction: Why the Tokenizer Matters

Before GPT-5.x “understands” a single word you type, your text passes through a component that most people never think about: the tokenizer. It’s a compression layer that converts raw text into a sequence of integer IDs and every design decision baked into it has downstream consequences for cost, accuracy, multilingual performance, and hallucination rates.

OpenAI’s tokenizer library, tiktoken[3], *is fully open source! The vocabulary files are hosted on Azure with hardcoded SHA-256 hashes for integrity verification(oooookay).*

This means we can download the vocabulary that GPT-5 uses, inspect every one of its 200,000 tokens, and draw **conclusions** about training data composition, architectural priorities, and retrieval mechanics. You can also access it with an interface, for free. https://platform.openai.com/tokenizer[4]

This is ***what. I. found.***

2. What I Downloaded

I cloned the tiktoken repository[3] and extracted the complete o200k_base encoding:

Artifact	Size	Description
o200k_base_raw.tiktoken	3.6 MB	Raw vocabulary file from openaipublic.blob.core.windows.net
o200k_base_all_tokens.jsonl	36.4 MB	All 200,000 tokens with IDs, base64, hex, UTF-8, byte lengths
o200k_base_all_tokens.csv	12.0 MB	Same data in spreadsheet-friendly format
o200k_base_config.json	1.6 KB	Full configuration: regex, special tokens, model mappings
o200k_base_stats.json	2.1 KB	Vocabulary statistics and Unicode script distribution
o200k_base_deep_analysis.json	—	URL patterns, brand analysis, citation tokens, domain density

Source URL: https://openaipublic.blob.core.windows.net/encodings/o200k_base.tiktoken[5]

SHA-256: 446a9538cb6c348e3516120d7c08b09f57c36495e2acfffe59a5bf8b0cfb1a2d

The file format is simple: each line contains a base64-encoded token and its integer rank, separated by a space. The rank simultaneously serves as the token ID and the BPE merge priority, lower rank can mean the token is more common and gets merged first during encoding (likely!).

HOWEVER there are some important points, you need to consider,

- **Rank isn't equal exact frequency**. It's an ordering of merge operations, not a guaranteed global frequency table. Different pairs can be close in frequency, ties, etc.
- **Token ID/rank may not reflect frequency in every tokenizer.** Some tokenizers reorder IDs, add special tokens or use different algorithms where rank score means something else.
- **"Common in your prompts" isn't equal common in the training corpus.** Domain/language shifts can make "low-rank" tokens rare in your text and "higher-rank" tokens common for you. *A BPE can be architectured for entropy optimization.*

- **Some OpenAI developers aren't happy with the tokenizer. There is a note in official GitHub repo (Check Section 12)**

Let me show you an example. I'll just ask a question to ChatGPT, for BPE, a basic question (needs to be verified, of course)

Figure

Then let's review the IDs and check tokenizer.

Figure

In the last step, I'll check the 2733 token ID. And yes, emdash is there.

Figure

3. The Evolution: From 50k to 200k Tokens

Learn more here: https://developers.openai.com/cookbook/examples/how_to_count_tokens_with_tiktoken/[6] Published at Dec, 16, 2022.

OpenAI has shipped seven tokenizer encodings across five generations. Each jump reveals a strategic shift:

Encoding	Vocab Size	Year	Models	Key Change
gpt2	50,257	2019	GPT-2	Baseline BPE tokenizer
r50k_base	50,257	2020	GPT-3 (davinci, curie, ada)	Retrained on larger corpus, same size
p50k_base	50,281	2021	Codex models (davinci, ada)[6]	New tokens // Not for programming[7]
p50k_edit	50,284	2022	Edit models	+3 Fill-In-the-Middle tokens
cl100k_base	100,277	2023	GPT-3.5, GPT-4, embeddings	2x vocabulary, multilingual expansion
o200k_base	200,019	2024	GPT-4o, GPT-4.1, GPT-5, o1, o3, o4-mini	2x again, camelCase-aware regex
o200k_harmony	201,088	2024	GPT-5 with tools	Same vocab? + 1,070 tool-use special tokens

Pattern: The vocabulary doubles at each major generation boundary (50k → 100k → 200k), roughly tracking a doubling in the number of languages and modalities the model supports.

The naming convention is revealing(**just a guess!**):

- "r" = retrained (same architecture, new data)
- "p" = Codex era // an important note: the early GPT models were for text generation. This "codex" is different than today.
- "cl" = ChatML / multilingual (the chat format era)

- "o" = omni (multimodal era)
- "harmony" = unified tool-use format

I conducted some test to see efficiency gains by encoding identical text through all five major encodings:

Content Type	gpt2	r50k	p50k	cl100k	o200k	Reduction (vs gpt2)
English prose	20	20	20	20	20	-
Python code	48	48	38	28	28	42% fewer
URL	18	18	18	14	13	28% fewer
Turkish text	39	39	39	28	22	44% fewer
Chinese text	56	56	56	29	18	68% fewer
Arabic text	70	70	70	49	21	70% fewer

English prose saw zero improvement. It was already optimal in 2019. The gains are entirely in code, URLs, and non-Latin scripts. Arabic text went from 70 tokens to 21, a 3.3x compression improvement. This reflects a deliberate investment in making models genuinely multilingual rather than English-first. *All Arabic LLMs need improvement, still!*

4. Regex Evolution: How “Word Splitting” Changed Across Four Generations

Before BPE merging even begins, the tokenizer splits text into chunks using a regex pattern. This “pre-tokenization” step determines the atomic boundaries the BPE algorithm operates within. The regex has been rewritten three times, and each rewrite reveals a shift in what OpenAI considers a “word.”

Generation 1: GPT-2 / r50k / p50k (2019-2022)

'(?:[sdmt]|ll|ve|re)| ?\p{L}++| ?\p{N}++| ?[^\s\p{L}\p{N}]++|\s++$|\s+(?!\S)|\s

Simple and fast. Splits on English contractions ('s, 't, 're, 've, 'm, 'll, 'd), then groups of letters, numbers, or punctuation. No distinction between uppercase and lowercase. No special handling for line breaks.

Generation 2: cl100k_base (2023)

'(?i:[sdmt]|ll|ve|re)|[^\r\n\p{L}\p{N}]?+\p{L}++|\p{N}{1,3}+| ?[^\s\p{L}\p{N}]++[\r\n]+|\s++$|\s[\r\n]|\s+(?!\S)|\s

Three critical changes:

1. Case-insensitive contractions ((?i:...)) - handles “I’M” and “i’m” identically
2. Numbers limited to 1-3 digits (\p{N}{1,3}) — forces 2024 to split into 202 + 4, preventing the model from memorizing arbitrary numbers as atomic tokens
3. Explicit newline handling (\s*[\r\n]) - preserves document structure

Generation 3: o200k_base (2024)

[^\r\n\p{L}\p{N}]?[\p{Lu}\p{Lt}\p{Lm}\p{Lo}\p{M}]*[\p{Ll}\p{Lm}\p{Lo}\p{M}]+(?i:'s|'t|'re|'ve|'m|'ll|'d)?

[^\r\n\p{L}\p{N}]?[\p{Lu}\p{Lt}\p{Lm}\p{Lo}\p{M}]+[\p{Ll}\p{Lm}\p{Lo}\p{M}]*(?i:'s	't	're	've	'm	'll	'd)?

\p{N}{1,3}

?[^\s\p{L}\p{N}]+[\r\n/]*

\s*[\r\n]+

\s+(?!\S)

\s+

A complete rewrite with seven alternation branches. The most important innovation: native camelCase and PascalCase awareness.

- Branch 1 matches words ending in lowercase: Hello, camelCase → splits into camel + Case
- Branch 2 matches words ending in uppercase: XMLParser, GPT → keeps acronyms intact
- Both branches distinguish between \p{Lu} (uppercase), \p{Ll} (lowercase), \p{Lo} (other letters like CJK), and \p{M} (combining marks)

This means ChatGPT is split into Chat + GPT at the regex level, before BPE even runs. The tokenizer natively understands programming naming conventions.

A practical demonstration:

Input: "camelCaseVariable"

GPT-2 regex: ["camelCaseVariable"] → 1 chunk → BPE decides

o200k regex: ["camel", "Case", "Variable"] → 3 chunks → each processed independently

5. Special Token Archaeology: From Documents to Tool Calls

Special tokens are control signals that the model sees but humans don’t type. Their evolution tells the story of OpenAI’s product architecture:

GPT-2 Era (2019): One token, one job

(ID 50256) - the only special token

The model could mark “end of document.” That’s it. No chat, no tools, no system prompts.

Codex Era (2021-2022): Fill-In-the-Middle

(50256)

(50281) - code before the cursor

(50282) - what to generate

(50283) - code after the cursor

These three tokens are how early GPT model works. The model sees: [prefix code] [suffix code] and generates the code that should appear at the cursor position. The reordering (suffix before middle) is intentional. It lets the model attend to both surrounding code before generating.

GPT-3.5/4 Era (2023): Chat format

(100257)

(100258)

(100259)

(100260)

(100276)

separates system/user instructions from the model's response. This is the structural foundation of the chat format. Note: and (used in ChatML for message boundaries) are NOT in the tiktoken encoding. They're added at the API layer, not the tokenizer layer.

GPT-4o/5 Base (2024): Stripped down

(199999)

(200018)

Only two special tokens. FIM tokens are gone. Between endoftext (ID 199999) and endofprompt (ID 200018), there are 19 deliberately empty ID slots (199998, 200000-200017). These gaps are designed to be filled by deployment-specific configurations. Just remember the web search, early DALL-E and file search.

GPT-5 Harmony (2024): The tool-use explosion

Note: The o200k_harmony encoding details below are derived from [specific source: community reverse-engineering / API inspection / tiktoken source code / other]. The exact special token names and their IDs should be verified against the tiktoken repository.

(199998) - document start

(199999) - document end

(200002) - tool output boundary

(200003) - constrained generation trigger

(200005) - multimodal stream selector

(200006) - generic start delimiter

(200007) - generic end delimiter

(200008) - chat message boundary

(200012) - tool/function call marker

+ 1,075 reserved slots (200013-201087) - future expansion

1,091 total special tokens. The harmony encoding fills every gap in the base encoding and adds 1,075 reserved slots. The name "harmony" suggests this is the unified format for all tool types; browser, code interpreter, DALL-E, file search, and any future tool. Now we are using ChatGPT native apps.

The reserved slots (200013-201087) are pre-allocated for tools that don't exist yet. Each new tool type can get its own dedicated special token without requiring a vocabulary retrain.

Key architectural insight: o200k_base and o200k_harmony encode regular text identically. They share the same regex, the same 199,998 mergeable tokens, and the same BPE merge order. The difference is purely in control tokens. This means the base model weights can be shared. The tool-use behavior is entirely in the special token embeddings and the post-training. (Just a guess!)

6. The Vocabulary Under a Microscope

6.1 Basic Statistics

Metric	Value
Total tokens	200,000 (199,998 mergeable + 2 special)
Valid UTF-8 tokens	198,436 (99.2%)
Raw byte sequence tokens	1,562 (0.8%)
Single-byte tokens	256 (the full byte range)
Average token length	6.99 bytes
Maximum token length	128 bytes

6.2 Byte Length Distribution

The vocabulary forms a roughly normal distribution centered around 5-7 bytes, with a long tail reaching 128 bytes:

1 byte: 256 tokens (individual bytes)

2 bytes: 4,653 tokens

3 bytes: 18,185 tokens

4 bytes: 26,166 tokens

5 bytes: 26,796 tokens

6 bytes: 31,386 tokens ← PEAK

7 bytes: 24,918 tokens

8 bytes: 15,950 tokens ████████

9 bytes: 16,567 tokens ████████

10 bytes: 10,126 tokens ████

...

128 bytes: 1 token (one extremely long token)

The peak at 6 bytes matches common English words with a leading space (e.g., hello = 6 bytes). Tokens longer than 20 bytes are typically whitespace patterns (indentation), repeated characters, or very common multi-word phrases baked into single tokens.

The 15 longest tokens by byte length:

#	ID	Bytes	Content	Category
1	72056	128	128 spaces	Whitespace (code indentation)
2	179041	113	----...---- (112 dashes + space)	Separator line
3	182513	112	----...---- (112 dashes)	Separator line
4	141084	97	----...---- (96 dashes + space)	Separator line
5	85810	96	----...---- (96 dashes)	Separator line
6	168258	96	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\...\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ (96 asterisks)	Comment block
7	193856	96	====...==== (96 equals signs)	Separator line
8	195732	95	95 spaces	Whitespace (code indentation)
9	174893	91	91 spaces	Whitespace (code indentation)
10	146780	88	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\...\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ (88 asterisks)	Comment block
11	134788	87	87 spaces	Whitespace (code indentation)
12	116288	83	83 spaces	Whitespace (code indentation)
13	166044	82	====...==== (81 equals + space)	Separator line
14	79593	81	----...---- (80 dashes + space)	Separator line
15	111383	81	/\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\...\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ (C-style block comment)	Comment block

These are all code formatting artifacts. The longest token in the entire vocabulary - 128 consecutive spaces - exists because deeply indented source code appeared frequently enough in the training corpus for BPE to merge individual spaces all the way up to a 128-byte sequence. Similarly, separator lines (---, *, ===) are ubiquitous in code comment headers, markdown, and text files, producing tokens up to 113 bytes long. These tokens are extraordinarily efficient at compressing boilerplate: a single token (1 generation step) replaces what would otherwise be 128 individual byte tokens.

The longest meaningful (non-whitespace, non-repeated) tokens tell a different story:

#	ID	Bytes	Characters	Content	Language
1	150141	61	20	สำนักเลขานุการองค์กร	Thaï : « Secretariat of the Organization »
2	135127	55	18	วิเคราะห์บอลวันนี้	Thaï : « Today’s football analysis »
3	173900	51	17	เปิดอภิปรายทั่วไป	Thaï : « Open general debate »
4	193807	49	16	தெரிவித்துள்ளார்	Tamoul : « has stated »
5	186053	43	14	მნიშვნელოვანია	Géorgien : « is important »
6	132700	40	13	प्रधानमन्त्री	Hindi : « Prime Minister »
7	134086	40	13	প্রধানমন্ত্রী	Bengali : « Prime Minister »
8	162457	40	13	विश्वविद्यालय	Hindi : « University »
9	161518	26	26	abcdefghijklmnopqrstuvwxyz	Alphabet ASCII
10	184150	26	26	ABCDEFGHIJKLMNOPQRSTUVWXYZ	ALPHABET ASCII
11	130756	21	20	verantwoordelijkheid	Néerlandais : « responsibility »
12	193348	19	18	Wahrscheinlichkeit	Allemand : « probability »
13	106123	19	18	telecommunications	Anglais
14	197767	19	18	disproportionately	Anglais
15	154976	19	19	.onreadystatechange	API JavaScript

Thai dominates the longest meaningful tokens because Thai script uses combining marks and vowels that consume 3 bytes per character in UTF-8, so a 20-character Thai phrase takes 61 bytes. The fact that these Thai phrases exist as single tokens means they appeared extremely frequently in the training data, phrases like “Secretariat” and “football analysis” suggest a heavy representation of Thai government/news/sports content. Georgian, Tamil, Hindi, and Bengali also appear, reflecting the vocabulary’s multilingual reach at the byte level.

The longest pure-English word as a single token is telecommunications (18 chars, ID 106123). The longest ASCII token is the full alphabet abcdefghijklmnopqrstuvwxyz (26 chars, ID 161518), clearly a training data artifact from examples, character sets, and documentation.

The peak at 6 bytes matches common English words with a leading space (e.g., hello = 6 bytes). Tokens longer than 20 bytes are typically whitespace patterns (indentation), repeated characters, or very common multi-word phrases baked into single tokens.

6.3 Unicode Script Distribution

Script	Token Count	% of Vocabulary
Latin	118,902	59.5%
Cyrillic	14,111	7.1%
Arabic	7,985	4.0%
CJK	7,165	3.6%
Devanagari	3,947	2.0%
Hangul (Korean)	2,346	1.2%
Hebrew	2,344	1.2%
Georgian	2,112	1.1%
Bengali	2,095	1.0%
Armenian	1,726	0.9%
Other scripts	~36,000	18.0%

Latin dominates with 59.5% of the vocabulary, despite representing roughly 40% of internet content. This imbalance is precisely what an OpenAI developer commented on (more in Section 12).

6.4 Domain Density: Who Gets the Most Tokens?

I tried to measure what percentage of common domain-specific words are encoded as single tokens (the most efficient representation):

Programming: 100% single-token 

News/media: 70% single-token 

Finance: 50% single-token 

Medical: 40% single-token 

Web/HTML: 40% single-token 

Math/science: 30% single-token 

Legal: 30% single-token 

Programming is the only domain with 100% single-token coverage. Every common keyword; function, class, return, import, export, async, await, null, undefined, true, false is a single token. This is the strongest signal that OpenAI's training data is heavily weighted toward code.

6.5 Code Tokens in the First 1,000 IDs

I found 62 tokens in the first 1,000 IDs that are commonly used in code, including parentheses, brackets, and indentation patterns. However, many of these (parentheses, brackets, colons) appear frequently in non-code contexts: mathematics, academic writing, URLs, and structured text. The strongest code-specific signals are the 4-

space indent (ID 257), 8-space indent (ID 269), and statement terminators like);\n (ID 362). These whitespace/syntax patterns are genuinely code-specific and suggest significant code representation in training data, though the overall count of 62 overstates the case by including general-purpose punctuation.

ID 7: (

ID 393: //

ID 560: ==

ID 8:)

ID 405: {\n

ID 609: ->

ID 58: [

ID 412: ,\n

ID 742: ::

ID 60:]

ID 416: ()

ID 354: {

ID 90: {

ID 446:)\n

ID 257: (4-space indent)

ID 92: }

ID 362:);\n

ID 269: (8-space indent)

The 4-space indent is token ID 257, just barely past the 256 single-byte tokens. The 8-space indent is ID 269. // (line comment) is ID 393.);\n (end of statement) is ID 362. These are among the most common patterns in the entire training corpus.

7. Hallucination Risk: Single-Token vs Multi-Token Entities

A plausible **hypothesis** from the tokenizer analysis: single-token entities may carry lower hallucination risk than multi-token entities, all else being equal. However, this effect is likely secondary to training data frequency. A multi-token entity seen millions of times (e.g., 'OpenAI' = 2 tokens) will hallucinate far less than a rare single-token entity in a low-resource language. Token atomicity is one factor among many, including training exposure, RLHF reinforcement, and retrieval grounding.

Brand/Entity Atomicity

Single-token (atomic, lower hallucination risk):

Word	Definition	Example Sentence
Entity	A thing with distinct and independent existence.	"Google" is listed as an entity in the table.
Token	A symbolic representation of something else.	The "Token ID" column contains unique numerical tokens.
Notes	A brief record of facts, topics, or ideas, used as an aid to memory.	The "Notes" column provides additional context for some entities.
Tech giant	A very large and influential technology company.	Google is described as a "Tech giant".
Decision	A conclusion or resolution reached after consideration.	The note for Google mentions a "one decision".

Multi-token (sequential generation, higher risk): // Note: Token count increases theoretical generation risk, but training data frequency is the primary factor. "Mustafa Kemal Atatürk" = 7 tokens but zero hallucination risk because of massive training exposure. IT DEPENDS.

Entity	Tokens	Pieces	Risk
OpenAI	2	['Open', 'AI']	Medium
ChatGPT	2	['Chat', 'GPT']	Medium
JavaScript	2	['Java', 'Script']	Medium
Bloomberg	2	['Bloom', 'berg']	Medium
Anthropic	2	['Anth', 'ropic']	Medium
PostgreSQL	3	['Post', 'gre', 'SQL']	Higher
Silicon Valley	3	['Sil', 'icon', ' Valley']	Higher
Elon Musk	3	['El', 'on', ' Musk']	Higher
Mustafa Kemal Ataturk	7	['Must', 'afa', ' Kem', 'al', ' At', 'at', 'urk']	LOW

URL Hallucination

No domain name is a single token. Even the most common websites require 2-4 tokens:

google.com → ['google', '.com'] (2 tokens)

wikipedia.org → ['w', 'ikipedia', '.org'] (3 tokens)

bloomberg.com → ['b', 'loom', 'berg', '.com'] (4 tokens) // or 'bloom', 'berg' *(both possible??? I really don't know)*

A full URL like https://en.wikipedia.org/wiki/Artificial_intelligence requires 11 tokens. Each token is an independent generation step where the model could turn off course. This is why ChatGPT frequently generates plausible-looking but non-existent URLs, the path after the domain is generated token-by-token with no ground truth to anchor it.

Date Hallucination

Years are always 2+ tokens: 2024 → ['202', '4'], 2025 → ['202', '5']. The 202 prefix is shared across an entire decade, but the final digit is a separate prediction. This is why models sometimes generate temporally confused dates, the prefix is correct but the suffix is drawn from the wrong distribution.

8. AI Search Optimization (AEO): Writing for Tokenizers // EXPERIMENTAL

As AI-powered search engines (ChatGPT, AIO, AI Mode, Perplexity, Gemini, etc.) become primary information retrieval interfaces, content optimization shifts from keyword matching to token-level efficiency. Here's what the tokenizer tells us about how to write content that AI systems can retrieve, process, and cite accurately.

8.1 Token Density by Content Format

I measured characters per token across different content formats. Higher density = more information per token = more content fits in the model's context window:

Format	Chars/Token	Relative Efficiency
Plain English prose	5.9	100% (baseline)
Markdown article	4.8	81%
Code with comments	4.4	75%
Academic citation text	4.3	73%
URL-heavy content	4.2	71%
JSON structured data	4.0	68%
Turkish prose	3.6	61%
Markdown tables	2.7	46%

Plain English prose is the most token-efficient format. Markdown formatting, JSON structure, and table syntax all consume tokens that carry formatting metadata rather than semantic content. A markdown table uses 2.2x more tokens than the same information expressed as plain text.

Practical implication: In a RAG pipeline with 8,000 tokens allocated for retrieved content, plain English delivers ~47,200 characters of information while markdown tables deliver only ~21,600 characters. The formatting cuts usable content by more than half.

8.2 Content Structure Recommendations

Lead with the answer. AI retrieval systems often truncate content to fit context windows. **IF** the first 500 tokens of a page contains the core information, it may have advantage for the retrieval pipeline.

For factual content with discrete data points (specs, metrics, comparisons), structured lists can reduce token count by 20-40% compared to verbose prose, because they eliminate transitional phrases. However, note that plain prose remains the most token-dense format per character (5.9 chars/token vs 4.8 for markdown). The efficiency gain from lists comes from eliminating filler words, not from the list format itself. For narrative content or explanations, prose is more token-efficient.

PROSE (67 tokens):

The system is 40% faster than previous versions, costs 25% less, and maintains 99.9% uptime.

LIST (28 tokens):

Benefits:

- Performance: 40% faster
- Cost: 25% lower
- Reliability: 99.9% uptime

Use numerals, not words. 1,234,000 is 5 tokens. one million two hundred thirty-four thousand is 8 tokens. The numeral form is both more token-efficient and less ambiguous for the model to cite.

Prefer common abbreviations for multi-word entities. MIT is 1 token. Massachusetts Institute of Technology is 5 tokens. The abbreviation is 5x more efficient and has zero hallucination risk (single token, single decision).

Include explicit attribution signals. The words according (ID 149330), source (ID 4935), research (ID 113140), study (ID 5012), and report (ID 22869) are all single tokens. Using these words helps the model recognize content as citable. Place attribution every 2-3 factual claims.

Use single-token transition words.

8.3 Things to Avoid

Long URLs in body text. A URL like https://en.wikipedia.org/wiki/Artificial_intelligence#Machine_learning costs 10+ tokens and is a hallucination magnet. Use anchor text instead and place the URL in metadata or a footnote.

Excessive Markdown formatting. Every #, *, , |, - consumes tokens that could carry content. Use formatting sparingly and only when it genuinely improves structure.

Unicode decoration. Lines of ═══════ or ▓▓▓▓▓▓▓ or clusters of emoji like 🎉🎊🎈 consume tokens aggressively for zero informational value.

Nested parenthetical expressions. ((like this (especially))) creates deep token nesting that the model must track through its context window.

9. The Developer Comment: Internal Disagreement at OpenAI

In the source file tiktoken_ext/openai_public.py, lines 101-103 contain a revealing comment from the tiktoken author: [8]

**# 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.

This tells us several things:

1. The o200k tokenizer was NOT designed by the tiktoken library author. The tiktoken maintainer (likely Shantanu Jain / @hauntsaninja based on commit history) is commenting on someone else's work. The

tokenizer vocabulary was trained by a different team, probably the pre-training team.

2. There is internal disagreement about language token allocation. The comment specifically calls out cross-language token efficiency as a point of contention.
3. The regex is acknowledged as suboptimal. The seven-branch alternation pattern works correctly but is slower than it needs to be. This suggests o200k was optimized for downstream model performance (training loss, perplexity) rather than encoding speed.
4. The critique about language allocation is supported by the data:

Latin scripts: 118,902 tokens (59.5% of vocabulary)

Non-Latin scripts: 81,096 tokens (40.5% of vocabulary)

Latin scripts represent roughly 40% of internet content but receive 59.5% of the token allocation. Meanwhile, CJK scripts (Chinese, Japanese, Korean) representing a significant portion of global internet content, receive only 3.6% of the vocabulary. The developer's comment suggests they believe a more balanced allocation would improve multilingual performance without significantly degrading English quality.

10. Multilingual Tax: The Cost of Non-English Content**

The tokenizer creates a measurable efficiency gap between languages. I can call this the "multilingual tax." *I often see many English query fan-outs in Turkiye, even I ask long questions in Turkish. It also explains very well.*

Using a parallel test sentence (same meaning across all languages: a statement about the AI market reaching $500B by 2030):

Language	Tokens	Chars/Token	vs English	Character
English	28	5.57	1.00x (baseline)	Latin
Spanish	35	5.49	1.25x	Latin
French	36	5.06	1.29x	Latin
Arabic	40	3.25	1.43x	Arabic
Turkish	42	3.50	1.50x	Latin
Chinese	29	1.69	1.04x	CJK

French and Spanish: Closer to English, But Not Equal

French and Spanish, the two largest Latin-script languages after English, pay a 25-29% token tax. This is far less than Turkish (50%) or Arabic (43%), but still meaningful at scale:

- French apostrophe contractions are the main overhead driver. Every l', d', c', n', qu', j' splits at the apostrophe, adding 1 token per contraction. A typical French paragraph has 5-8 such contractions, AND SOMETIMES adding 5-8 tokens of pure grammatical overhead. *BUT, Mistral has different architectures! And it needs to be tested on my side.*
- Spanish accent splits fragment otherwise-common words: está → ['est', 'á'], económico → ['econ', 'ómico'], tecnología → ['te', 'cn', 'ología']. The accent character forces a token boundary. PLUS: "económico" is a

single token(ID 65316), too. *tecnologíaaaaaaaaaaaa*

- Single-token rate for domain vocabulary: Both French and Spanish achieve only 29% single-token rate for key technical/political words (6/21 tested). Compare: France and España are single tokens, but gouvernement splits into 3, inteligencia into 3, cybersécurité into 4, and ciberseguridad into 4.
- Spanish is ~4% more efficient than French for equivalent content (5.49 vs 5.06 chars/token), largely because Spanish lacks the apostrophe contraction overhead.

You see? MANY PROBABILITIES for efficiency.

French SEO content tokenizes well at the phrase level, référencement naturel (natural SEO) costs 3 tokens, and common articles/prepositions (le, la, les, de, du) are all single tokens. But compound technical terms and accented words consistently fragment.

For queries specifically:[4]

"What is machine learning?" → 5 tokens

"¿Qué es el aprendizaje automático?" → 6 tokens (1.2x English)

"Qu'est-ce que l'apprentissage auto?" → 9 tokens (1.8x English - apostrophe splits)

"yapay zeka nedir?" → 7 tokens (1.4x English)

"Türkiye ekonomisi 2025 büyüme oranı" → 11 tokens (2.2x English)

Figure

The French query is revealing: Qu'est-ce que l'apprentissage auto? produces 9 tokens because every apostrophe contraction (Qu', l') forces a split: ['Qu', "'est", '-ce', ' que', ' l', "'ap", 'prentissage', ' auto', '?']. The equivalent Spanish query with no contractions costs only 6 tokens. This means:

1. Higher API cost for the same information in non-English languages
2. Less content fits in the context window for non-English RAG retrieval
3. More generation steps for non-English responses = more opportunities for error
4. Proper nouns vary wildly: Macron → 2 tokens (Mac + ron), España → 1 token, Mustafa Kemal Atatürk → 7 tokens, Elon Musk → 3

The situation improved dramatically from cl100k to o200k for all languages, but the gap with English remains.

Turkish-Specific Tokens in the Vocabulary

The vocabulary contains 2,103 tokens with Turkish-specific characters (ş, ç, ğ, ü, ö, ı, İ, Ş, Ç, Ğ, Ü, Ö). Each Turkish special character exists as its own single-byte token:

ID 572: ü

ID 704: ç

ID 1678: ğ

ID 573: ö

ID 981: ş

ID 4599: İ

ID 612: ı

Common Turkish suffixes are also tokenized:

ID 3681: ın

ID 8311: lı

ID 10569: ını

ID 4515: ün

ID 6637: ır

ID 10884: ında

ID 4041: ça

ID 9671: ış

ID 23219: ması

Türkiye (ID 177744) is a single token, a good sign. But İstanbul splits into ['İ', 'stanbul'] (2 tokens), and Merhaba splits into ['Mer', 'haba'] (2 tokens).

11. RAG Pipeline Economics: Token Budgets in Practice (Numbers are just an estimation)

LET'S SAY IF: In a typical AI search interaction, the token budget breaks down as follows:

System prompt: 500 tokens (instructions, persona, format)

User query: 30 tokens (the question)

Retrieved documents: 8,000 tokens (RAG context - the main cost)

Model response: 800 tokens (the answer)

Total input: 8,530 tokens

Total output: 800 tokens

The retrieved documents (8,000 tokens) dominate the cost. Optimizing content to be more token-dense directly reduces the cost of AI search. If your content achieves 5.9 chars/token (plain English) instead of 2.7 chars/token (table-heavy), you deliver 2.2x more information within the same token budget.

Context Window Capacity

How much content can different models hold in their context window (assuming 60% allocated to retrieved content):

Model	Available Tokens	English (Approximate Words)	Turkish (Approximate Words)
GPT-4 (8k)	4,900	~5,800 words	~3,500 words
GPT-4o (128k)	76,800	~90,800 words	~55,300 words
GPT-5 (est. 256k)	153,600	~181,500 words	~110,600 words

Turkish content gets roughly 61% of the capacity that English content gets in the same context window, a direct consequence of the tokenizer efficiency gap.

12. Conclusions and Implications

What the tokenizer reveals about OpenAI's priorities

1. Code is king. Programming tokens dominate the early vocabulary (low IDs = high frequency in training data). 100% of common programming keywords are single tokens. The training corpus is heavily code-weighted.
2. Multilinguality is improving but imperfect. The jump from cl100k to o200k dramatically improved non-Latin script efficiency (Arabic: 70→21 tokens, Chinese: 56→18 tokens). But the vocabulary allocation still favors Latin scripts at 59.5%, and an OpenAI developer has publicly noted this imbalance. THIS IS VERY INTERESTING! [8]
3. Tool use is the next frontier. The harmony encoding's 1,075 reserved special token slots signal that OpenAI is building toward a rich ecosystem of model capabilities, each with dedicated control tokens. Today I see , , , . Tomorrow there may be tokens for browser actions, image generation, audio processing, and capabilities I haven't imagined.
4. Citation accuracy CAN BE structurally limited by tokenization.
5. The gap between base and harmony is the gap between a language model and an AI assistant. Same vocabulary, same BPE merges, same regex. The difference is entirely in 1,089 control tokens that structure tool calls, message boundaries, and constrained generation. The assistant is a language model wearing a protocol layer.

For content creators and SEO practitioners

The shift from traditional SEO to Answer Engine Optimization (AEO) or GEO requires understanding that AI systems see your content through the lens of a tokenizer. Token density, entity atomicity, attribution signals, and structural clarity all affect whether your content is retrieved, how much of it fits in context, and whether it's cited accurately.

The single most impactful change: write content that is dense, structured, and attribution-rich. Lists over prose. Numerals over words. Common abbreviations over full names. Explicit source attribution every 2-3 facts. Lead paragraphs with answers, not context.

The tokenizer is the first gate. Everything the model does retrieval, reasoning, citation, generation, operates on what the tokenizer produces. Understanding its behavior is understanding the foundation of modern AI.

13. Cross-Reference: RESONEO's ChatGPT Search Architecture vs My Tokenizer Findings

In February 2026, the French SEO research firm RESONEO published a comprehensive reverse-engineering of ChatGPT Search's internal architecture[9], based on months of network traffic analysis, code decompilation, and systematic testing. Their work operates at the infrastructure layer, HTTP requests, JSON payloads, provider APIs, feature flags. My tokenizer analysis operates at the foundational layer, the 200,000 atomic units that every piece of text must pass through before the model can process it.

Together, these two analyses form a complete picture: RESONEO shows what ChatGPT does, and my tokenizer analysis shows why certain things are structurally easier or harder for the model to do.

13.1 The Fan-Out Engine Meets the Tokenizer

RESONEO's finding: ChatGPT doesn't issue one search query. It decomposes user queries into 1-3 parallel "fan-out" queries (up to 20+ in thinking mode), each targeting different angles of the question. Shopping and image fan-outs run in parallel on separate pipelines.

My tokenizer insight: Fan-out query generation is itself a token-by-token process. The efficiency of the fan-out depends on how the original query tokenizes:

User query: "best gaming PC for Black Friday under $1500"

Tokens: ['best', ' gaming', ' PC', ' for', ' Black', ' Friday', ' under', ' $', '150', '0'] = 10 tokens

Fan-out 1: "gaming pc specs 2025" → 6 tokens

Fan-out 2: "best gaming pc black friday" → 6 tokens

Fan-out 3: "gaming pc deals under 1500" → 7 tokens

Each fan-out is generated from the model's understanding of the original query tokens. The model has ~10 tokens of "input signal" to work with. In thinking mode, the chain-of-thought reasoning consumes additional tokens before fan-out generation even begins, which explains why thinking mode produces more and better fan-outs: the model has more internal token budget to reason about query decomposition.

Queries in token-expensive languages use more tokens to represent the same meaning. This has a direct cost implication (more input tokens = higher API cost) and means more of the context window is consumed by the query itself. However, the fan-out engine operates on the model's internal representations, where the semantic content is preserved regardless of token count. The model understands the meaning equally well whether it took 5 or 11 tokens to encode it.The real multilingual disadvantage is not 'less semantic signal per token' but rather:

- (1) higher cost per query,
- (2) less remaining context budget for retrieved content
- (3) possible quality degradation if the model's training data underrepresents the language.

Community observations suggest English fan-outs for non-English queries, which is more likely due to the English-dominant training distribution than token efficiency.

13.2 The Sonic Classifier Through the Tokenizer Lens

RESONEO's finding: Before any search, the Sonic Classifier (a probabilistic classifier, latency ~196ms) determines whether external data is needed, with a threshold of ~65%. It runs on a config called sonic_force_pg_switcher.

My tokenizer insight: The Sonic Classifier operates on tokenized input. Its decision is influenced by the token composition of the query:

The Sonic Classifier operates on the model's internal representations of the query. While input is tokenized, the classifier likely works at the embedding/representation level, not on raw token IDs. The search probability is more plausibly driven by the model's confidence in its parametric knowledge (how well it 'knows' the answer from training) rather than token atomicity.

A multi-token entity like 'OpenAI' that appeared millions of times in training would have strong embeddings despite being 2 tokens, while a single-token entity from a low-resource domain might still trigger search.

Note: The claim that multi-token entities 'might trigger search more often' is speculative and has no supporting evidence. I will mark here as a hypothesis.

13.3 Entity Linking: Knowledge Graph Meets Token Atomicity

RESONEO's finding: ChatGPT uses a proprietary NER system with entity disambiguation. When you click an entity, a dynamic prompt is generated: "Tell me about [ENTITY]. The entity category is [CATEGORY]. The disambiguation is [DESCRIPTION]." This is sent to gpt-5-instant to generate a sidebar. The entity format uses Unicode private-use characters (\ue200, \ue201, \ue202) for structured markup.

Our tokenizer insight: Entity recognition accuracy is directly correlated with token atomicity:

Entity	Tokens	NER Difficulty	RESONEO Category
Google	1	Trivial	company
Microsoft	1	Trivial	company
Reuters	1	Trivial	—
Wikipedia	1	Trivial	—
OpenAI	2 (Open + AI)	Medium - ambiguous boundary	company
ChatGPT	2 (Chat + GPT)	Medium	software
Bloomberg	2 (Bloom + berg)	Medium - could be person or company	company
Silicon Valley	3 (Sil + icon + Valley)	Hard - fragments don't signal the entity	place
Elon Musk	3 (El + on + Musk)	Hard - El and on are generic	people

This explains RESONEO's observation that entity taxonomy quality is "variable", categories like sports_event containing only historical battles (35% accuracy), or fictional_character mixing gods and film characters (57%

accuracy). The NER system struggles most with entities that the tokenizer fragments into ambiguous subword pieces.

The Unicode private-use area insight: The entity markup characters \ue200 (U+E200), \ue201 (U+E201), \ue202 (U+E202) are NOT in the o200k_base vocabulary as special tokens. Each character encodes as 3 UTF-8 bytes (e.g., U+E200 = 0xEE 0x88 0x80), and crucially, none of these byte sequences exist as merged tokens in the vocabulary, the pair 0xEE+0x88 has no BPE merge rule, so each PUA character decomposes into 3 individual byte-level tokens. In the entire 200,000-token vocabulary, only 5 merged tokens start with 0xEE, and none involve the 0x88 second byte. This is significant: BPE merges are frequency-driven, byte pairs that appear often in the training corpus get merged into single tokens. The absence of any 0xEE+0x88 merge means these PUA characters were vanishingly rare in the pre-training data, which indirectly supports the conclusion that the model learned to use them as entity markers during post-training (instruction tuning / RLHF), not from the pre-training corpus.

A subtle context-dependent behavior: in running text, the leading 0xEE byte can merge with an adjacent character. For example, a preceding space forms a 0x20+0xEE merged token (rank 95,677), so " \ue200entity" tokenizes as [..., 0x20EE, 0x88, 0x80, "entity"] rather than [..., " ", 0xEE, 0x88, 0x80, "entity"]. The PUA delimiter still costs 3 token positions, but the byte boundary shifts, the space gets “absorbed” into the delimiter’s first byte.

This is architecturally different from the and tokens in o200k_harmony, which ARE dedicated special tokens with single atomic IDs (200012 and 200008 respectively). The entity system is a post-training convention imposed on the base vocabulary’s byte-level encoding, while the tool-calling system is an architectural feature with dedicated token slots that are structurally distinct from regular text.

13.4 Citation Types: Three Tiers Explained by Tokenization

RESONEO’s finding: ChatGPT generates three types of links:

1. Citations - visible, numbered [1][2][3] links in text (maximum visibility)
2. Other Sources - in the “More” section below (moderate visibility)
3. Hidden Links - consumed by the LLM but never shown to users (internal grounding only, e.g. arXiv, Wikipedia for factual grounding)

My tokenizer insight: The citation format itself reveals why these tiers exist:

Visible citation [1]: ['[', '1', ']'] = 3 tokens (cheap)

Full source link: ['https', '://', 'en', '.wikipedia', ...] = 7-15 tokens (expensive)

Hidden grounding URL: Consumed at retrieval, never generated = 0 output tokens

The three-tier system is a token economy optimization:

- Citations (3 tokens each) are cheap to generate and provide user-visible attribution
- Other Sources are URLs that the model retrieved and used but doesn’t spend output tokens referencing inline
- Hidden Links are the most token-efficient: the model consumed the content during retrieval (input tokens, paid by OpenAI) but never generates the URL (saving output tokens)

At current GPT-4o pricing ($10/M output tokens), a single inline URL like https://en.wikipedia.org/wiki/Artificial_intelligence costs (10 tokens) ~$0.00015 in output tokens. Across millions

of queries per day, hidden links vs visible citations represents a meaningful cost difference. // **Observation**

13.5 Recency Bias: The Training Cutoff Meets Web Search

RESONEO's finding: The model generates fan-outs with explicit recency parameters: {"q": "query", "recency": 30} for news, "recency": 365 for established information. The explanation: "The model already has older content baked into its training data up to the cutoff date. Web search only serves to fill the gap with fresh content."

My recency bias post is here. [10]

My tokenizer insight: This is confirmed by the token ID distribution. Year tokens tell the story:

"2020" → ['202', '0'] - both subwords deeply embedded (ranks

13.9 The*_Finding: Crawlers See Raw HTML, Not Rendered DOM_*(MORE TESTS NEEDED)

RESONEO's finding: Most LLM crawlers (ChatGPT, Claude, Gemini) do NOT execute JavaScript. Only Bing Copilot and Grok do. Content must be in static HTML or fallbacks.

My tokenizer insight: This has a second-order implication for token efficiency. JavaScript-rendered content often includes:

- Framework boilerplate (React.createElement, __NEXT_DATA__, webpack chunks)
- Escaped Unicode, HTML entities (&, ')
- JSON state hydration blobs
- CSS-in-JS strings

If a crawler somehow did execute JS and tokenized the full rendered output, the non-content tokens would pollute the context window. A clean HTML page with semantic markup tokenizes far more efficiently than a JS-hydrated page:

AI Market Report 2025

→ ~8 tokens (mostly content)

→ ~12 tokens (zero content)

Static HTML isn't just about crawler accessibility. It's about token density. Every class="tailwind-gibberish" attribute that gets into the crawler's view wastes tokens that could carry actual content.

13.10 The Google vs SerpAPI Lawsuit: Infrastructure Fragility (PERSONAL THOUGHTS)

RESONEO's finding: Google sued SerpAPI in December 2025 under DMCA Section 1201 for circumventing SearchGuard. This threatens the entire data supply chain of ChatGPT Search.

My tokenizer debate: If SerpAPI access is cut off, OpenAI faces a fundamental problem that goes beyond just replacing a search provider.

Switching to a fundamentally different search backend (e.g., a proprietary crawler with different coverage) would create a distribution mismatch: the model's tokenizer and weights are optimized for content that looks like

Google's index, but the retrieval would serve content with a different distribution. This could manifest as:

- Lower-quality fan-out queries (the model generates queries optimized for Google's ranking signals) - however ChatGPT is using query decomposition.
- Degraded entity recognition (the NER was trained on Google-indexed entities????)
- Citation quality drops (the model's citation patterns were learned from Google-style SERP formats) // or this is just false. MS-MARCO can be the answer. I don't know!

13.11 Summary: Two Layers of the Same System

Layer	RESONEO Analysis	My Tokenizer Analysis
Query Processing	Sonic Classifier, fan-out engine	Token efficiency determines classifier behavior
Search Execution	SerpAPI → Google,SearchApi.io[11]	Fan-out queries are token-by-token generation
Content Retrieval	SERP scraping, snippets, full pages	Token budget
Entity Recognition	NER + disambiguation + Unicode markup	Token atomicity predicts NER accuracy
Citation Generation	3-tier system (visible/more/hidden)	Token cost explains the tier structure
Response Generation	Synthesized answer with citations	Every character costs tokens; density matters
Recency	Explicit recency filters per fan-out	Year tokens after training cutoff need web grounding
Shopping	Mercury quiz + Google Shopping	Structured data is token-expensive; quiz saves budget
Multilingual	Fan-outs often generated in English even for non-English users	Non-English queries cost 1.3-2.3x more tokens

The combined insight: ChatGPT Search is an orchestration layer built on top of a token economy. Every architectural decision, from the fan-out count to the citation tier system to the shopping quiz, can be understood as an optimization of the token budget. RESONEO mapped the plumbing; I tried to map the water pressure.

I will share my findings and more testings with my Substack followers.[12]

Analysis conducted on tiktoken v0.11.0, o200k_base encoding (SHA-256: 446a9538…0cfb1a2d). Data extracted from the openai/tiktoken[3] *repository. RESONEO research referenced from* think.resoneo.com/chatgpt[9] *(February 2026). All token counts and statistics are reproducible using the extraction scripts included in TikToken repository.*

The GPT-5 tokenizer is an AEO input that almost nobody maps. Token boundaries shape what ChatGPT, Perplexity and Gemini can retrieve, which decides which pages earn AI citations. Answer engine optimization without tokenizer awareness is half-blind.

Generative engine optimization benefits from tokenizer-level thinking. Token boundaries decide which phrases generative engine optimization can actually move in ChatGPT, Perplexity and Gemini.

REFERENCES & SOURCES

[1] **Visi Summit** — https://visisummit.com

[2] **SMX** — https://smxmuenchen.de/

[3] **tiktoken** — https://github.com/openai/tiktoken

[4] **https://platform.openai.com/tokenizer** — https://platform.openai.com/tokenizer

[5] **https://openaipublic.blob.core.windows.net/encodings/o200k_base.tiktoken** — https://openaipublic.blob.core.windows.net/encodings/o200k_base.tiktoken

[6] **https://developers.openai.com/cookbook/examples/how_to_count_tokens_with_tiktoken/** — https://developers.openai.com/cookbook/examples/how_to_count_tokens_with_tiktoken/

[7] **New tokens // Not for programming** — https://developers.openai.com/api/docs/models/davinci-002

[8] **In the source file tiktoken_ext/openai_public.py, lines 101-103 contain a revealing com...** — https://github.com/openai/tiktoken/blob/main/tiktoken_ext/openai_public.py

[9] **RESONEO published a comprehensive reverse-engineering of ChatGPT Search's internal arch...** — https://think.resoneo.com/chatgpt/

[10] **My recency bias post is here.** — https://metehan.ai/blog/i-found-it-in-the-code-science-proved-it-in-the-lab-the-recency-bias-thats-reshaping-ai-search/

[11] **SearchApi.io** — http://searchapi.io/

[12] **I will share my findings and more testings with my Substack followers.** — https://metehanai.substack.com