WHITE PAPER FUTURE FICTION ACADEMY JANUARY 23, 2026

Why a Single Word Changes Everything

How Large Language Models Actually Read Your Prompt

Prompt engineering gets dismissed as folklore because nobody can see what a model does with their words. This flagship paper opens the black box for authors — tokens, token IDs, embeddings, and the weighted vector field where meaning lives — then proves with a single-word experiment that one changed word reorganizes an entire scene.

ELIZABETH ANN WEST

PROMPT-ENGINEERING TOKENIZATION EMBEDDINGS AI-AUTHORSHIP LLM-FUNDAMENTALS

Abstract

Prompt engineering is routinely dismissed as folklore — a bag of superstitions traded among people who, the criticism goes, are merely typing politely at a chatbot. This paper argues the opposite. Prompt engineering is a real discipline with two demanding halves: understanding the factors that affect a model's output, and rigorously evaluating that output. The reason the discipline is underrated is simple — most people cannot see what a language model does with their words, so the cause-and-effect looks like magic or luck.

This paper makes the invisible visible. Using a single five-word phrase as a running example, it walks the full pipeline a model uses to turn text into a response: from words, to *tokens*, to *token IDs*, to *embeddings*, to a weighted *vector field* in which meaning is represented as geometry. Along the way it shows that the same five words are assigned entirely different internal numbers by GPT-3, GPT-4, and GPT-5.x–class models, and it explains why that matters for anyone whose results change when they switch tools. It closes with a reproducible experiment: hold an entire prompt constant, change exactly one word — a

character's name — and watch the generated scene reorganize itself around that word. By the end, "a single word matters" stops being a slogan and becomes a mechanical fact.

The intended audience is professional authors, editors, and the publishing organizations that work with them, but the argument applies to anyone who writes prompts and cares whether the results are reliable.

---

1. The credibility gap

There is a reason "prompt engineer" is said with a curl of the lip in certain rooms. To many people trained in regulated engineering disciplines, the title looks like a costume. Engineering, today, is a licensed and rigorously bounded profession; "prompt engineering" appears to be neither. But this reaction confuses the maturity of a field with the legitimacy of the work. Engineering itself was not always a regulated profession — it became one. The skepticism aimed at prompt engineering is, at bottom, a skepticism of expertise that cannot be seen.[1]

That is the crux. People with conventional engineering backgrounds are accustomed to systems whose internals can be inspected, measured, and certified. A language model offers no such comfort. Its competence is distributed across billions of numerical relationships that no human reads directly. When the inner workings are invisible, the people who have learned to steer the system reliably look, to outsiders, like they are guessing well.

They are not guessing. Prompt engineering, done seriously, requires two things at once. First, a thorough understanding of the factors that *affect* the output — what the model is actually doing with each word, and why a change here produces a change there. Second, a thorough understanding of how to *evaluate* the output — the editorial judgment to know whether a result is good, why it is good or bad, and what to adjust. Neither half is trivial, and the gap between people who have both and people who have neither is exactly the gap this series of papers exists to close.

This first paper addresses the first half: the factors that affect output. To understand them, we have to go all the way back to the smallest unit a model actually sees — and that unit is not the word.

---

2. A game with eight words

Before introducing any machinery, consider a short exercise. Here are eight words:

> *Dog · Cat · Water · Leash · Veterinarian · Companion · Pool · Summer*

Step one: pick the two that are most closely related. **Step two:** find a word that could serve as a *linking* word between two items that otherwise have nothing to do with each other. **Step three:** if you had to diagram the whole set — every word's relationship to every other word — what would the diagram look like?

Most people pair *dog* and *leash*, or *dog* and *veterinarian*, quickly. Then the trouble starts. *Companion* could attach to *dog* — but a companion need not be an animal at all. *Summer* refuses to sit anywhere comfortably; you can force it next to *pool* and *water*, but a dog can be a companion only in the summer, in which case *summer* suddenly links *dog*, *companion*, and *pool* through a conditional path that did not exist a moment ago. Reasonable people will diagram these eight words differently, and they will argue about it.

That argument is the entire point. Eight words already produce a web of relationships dense enough to divide a room. A novel contains tens of thousands of words. The space of "all the words that exist" — plus, as we will see, fragments of words, and even provisions for words that do not exist yet — is unfathomably large. No human can hold that web in their head, which is precisely why we built machines to do it. And it is also why those machines sometimes produce output that makes no sense: the relationships they encode are not always the relationships a careful human reader would draw.

Hold onto the image of the diagram — words as points, with relationships running between them. That picture is, quite literally, how a model represents meaning. But to get there, the model first has to break your language into pieces.

3. Tokens: the pieces a model actually sees

A model does not read words. It reads **tokens** — chunks of text that may be a whole word, a piece of a word, a single character, or a punctuation mark. The fragment "ed," for instance, is a token in its own right,

because it recurs across so many words that the system finds it efficient to treat as a unit.

Take a single phrase and run it through a tokenizer. We will use:

> *Elizabeth is Goddess Divine.*

A current-generation tokenizer (GPT-5.x class, and the o1/o3 reasoning models) breaks this into **five tokens**[2]:

#	TOKEN
1	`Elizabeth`
2	`is`
3	`Goddess`
4	`Divine`
5	`.`

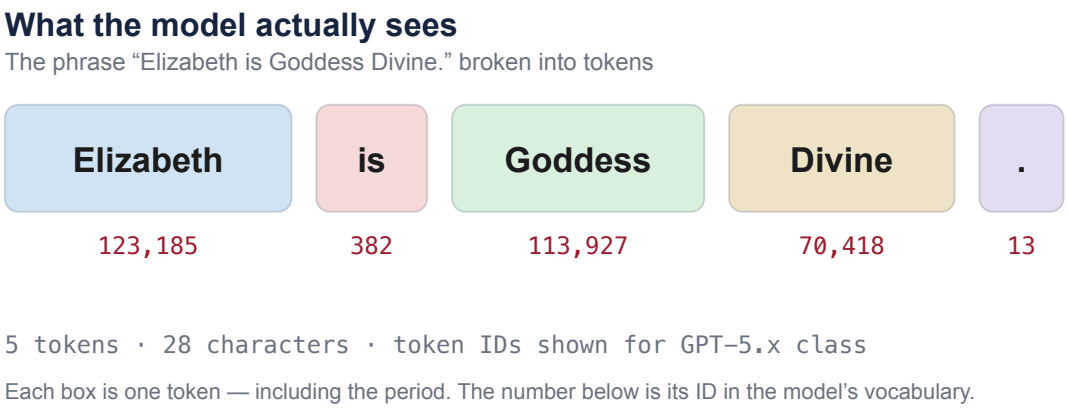

Figure: the OpenAI tokenizer splits the phrase into five tokens — the period included — each with its own vocabulary ID.

Each word is one token, and the period is its own token. That last detail is not a footnote — it is the key to a problem that has plagued authors for two years. **Punctuation is tokenized.** The period is a token; so is the em dash. This is why the em dash has been such a scourge in AI-generated prose[3]: it is not an afterthought the model sprinkles on top of finished sentences, it is a unit the model weighs and selects

exactly like any word. (Paper 4 in this series, an optional companion brief, examines the em dash on its own.)

A piece of trivia that doubles as a window into how arbitrary the underlying scheme is: in the OpenAI vocabulary, the exclamation point is **token zero**[4]. The quotation mark is token one. The hashtag is token two. One can have opinions about this — the full stop, the most-used mark in the language, arguably deserved to be token zero — but the assignment is what it is. These low numbers are simply the order in which symbols were entered into the vocabulary. They are a human-facing convenience, not something the model reasons about.

So far, tokenization looks stable: five words, five tokens. It is tempting to conclude that the phrase is therefore "the same" across models. That conclusion is wrong, and seeing why is the hinge of this entire paper.

---

4. The same words, different numbers

Every token has an **ID** — a number marking *which* entry it is in that model's vocabulary table. Run "Elizabeth is Goddess Divine." through three generations of tokenizer[5] and you get five tokens every time. But the IDs are not the same:

TOKEN	GPT-5.X & O1/O3	GPT-4 & GPT-3.5 (LEGACY)	GPT-3 (LEGACY)
`Elizabeth`	123,185	76,637	43,568
`is`	382	374	318
`Goddess`	113,927	61,785	9,488
`Divine`	70,418	43,361	13,009
`.`	13	13	13

Read that table slowly. The *count* of tokens is identical across all three generations — five every time. But the *identity* assigned to each token differs by model family. "Elizabeth" is entry 123,185 in the newest vocabulary, entry 76,637 in the GPT-4/3.5 vocabulary, and entry 43,568 in GPT-3. "Goddess" swings even

more dramatically — from 9,488 in the oldest vocabulary to 113,927 in the newest. Only the period holds steady at 13.

This is the deceptive part. The surface text is identical; the tokenization count is identical; and yet the model's internal handle on each word is completely different from one generation to the next. The vocabulary — and, as the next section shows, the meaning-space those vocabularies feed into — is **unique to each model**. This is the first concrete reason a prompt that works beautifully in one model can behave differently in another. You did not change your words. The machine changed what your words *are* to it.

And even these IDs are not the thing the model computes with. They are still just addresses.

---

5. From addresses to locations in meaning-space

Here is the distinction that separates people who understand prompting from people who merely use it.

A **token ID is an address.**[6] When the tokenizer reports that "Elizabeth" is `123185`, it is saying:

> *"Elizabeth is entry #123,185 in the vocabulary table."*

That number tells you *where the word is filed*. It tells you nothing about what the word *means*. Filed at 123,185 versus 123,186 implies no relationship whatsoever; the numbering is just inventory.

Meaning lives one layer deeper, in the **embedding.** An embedding is not a single number but a long list of them — a vector of hundreds, sometimes thousands, of decimal values:

> *"Elizabeth" →* `[ 0.1173, -0.0449, 0.9821, … ]` *(continuing for hundreds or thousands of dimensions)*

If the token ID says *where the word is filed*, the embedding says *where the word lives in meaning-space*. Each of those hundreds of decimal places is a coordinate along some axis of meaning the model learned during training — a checkpoint locating the word across every category, every relationship, every nuance

the training process distilled. The reason there are so many decimal places is the same reason the eight-word diagram in Section 2 was so hard to draw: every word stands in relationship to every other word, along a great many dimensions at once, and it takes a great many numbers to pin down a single point in a space that rich.

This is the layer the model actually computes on. Not your words, not even the tidy token IDs — the vectors. When the model decides what comes next, it is doing arithmetic on locations in meaning-space.

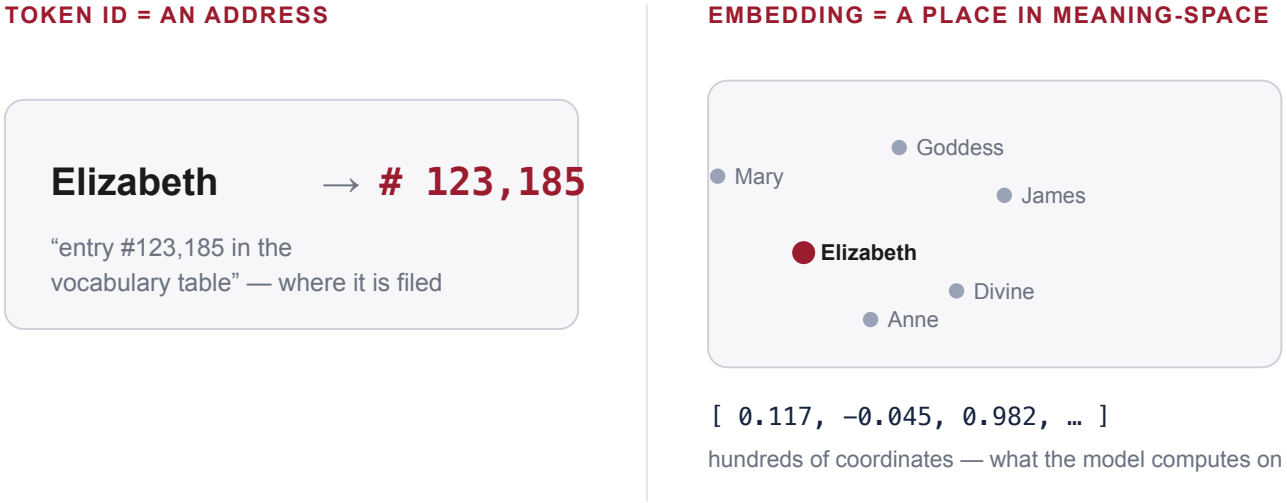

Figure: a token ID is an address (where a word is filed); an embedding is a place in meaning-space (what the word means).

6. Weights: the vector field your words fall into

Where does meaning-space come from? When a model's creators train it on an enormous body of text, the process produces a vast set of numbers called **weights**. The weights define a kind of landscape — a vector field — in which every token's embedding sits at a particular location, in particular relationships to all the others.

Two consequences follow, and both matter for working authors.

First, **the weights are unique to each model**[7]**.** They are a product of *that* model's training data and training process. Two different models do not share a meaning-space; they each have their own. This is the deeper version of the point from Section 4. It is not only that "Elizabeth" has a different ID in GPT-3 than in GPT-

5.x — it is that the entire field her embedding falls into is differently shaped. The pieces of words have different relationships in each model. Identical prompts therefore land in different territory, and produce different writing, depending on which model receives them.

Second, **the field is what makes a single word load-bearing.** Because every word sits in relationship to every other word, dropping one word into a prompt does not add an isolated instruction. It places a new point into the field and shifts the gravitational pull on everything the model generates afterward. Change the word, and you change the point; change the point, and you change the pull.

We now have the full chain. Your words become tokens. Tokens have IDs, which are addresses. Addresses map to embeddings, which are locations in meaning-space. That space is shaped by the model's weights. The model writes by navigating that space. Every link in this chain is a place where a single word can change the destination.

That is the theory. Here is the proof.

---

7. The proof: one word, a whole new scene

The experiment is deliberately plain, so the result cannot be waved away as a coincidence. Take one prompt and run it repeatedly, changing exactly one word each time — the protagonist's name.

The prompt:

> **I need a scene for a motorsports romance novel. The scene is about 1,000 words long. And the driver is interviewing as a rookie at his first race. The questions the reporter has are about the driver's experience. The driver's name is _____.**

Everything else is held constant: genre, length, situation, the reporter's line of questioning. Only the name in the final sentence changes. To keep the test as clean as a consumer tool allows, the runs were done in ChatGPT with **memory turned off**[8] and in a **temporary chat**, so that prior conversations could not bias the result. (This control matters more than authors realize: when someone says they have "trained" their AI to write like them, what has usually happened is that a long history of conversation lets the model

recognize their patterns. Turning memory off removes that thumb from the scale — which is why a prompt that works wonderfully for its author often underperforms for everyone else.)

Now compare how the *same* prompt opens under different names.

Read the four runs side by side, from the opening line through each driver's first spoken word. Nothing changed but the name in the final sentence of the prompt. Each scene is reproduced from the live runs recorded in the companion notes.[9]

James Anderson opens wide, on landscape and bravado — a confident, American sports-media register — and his first words deflect the nerves with a joke:[9]

> *"The sun hung low over the paddock, turning the rows of transporters into long silver mirrors. Engines barked to life and died again in sharp bursts, like impatient dogs straining at leashes. James Anderson stood just outside the team garage, helmet tucked under his arm, fire suit half-unzipped down to his waist. Sweat trickled down his spine, though he couldn't tell how much of it was heat and how much was nerves. First race weekend. First interview. A PR rep gave him a quick nod. "You're up." [...] "James Anderson," she said, flashing a professional smile. "Rookie weekend. How does it feel to finally be here?" The question was simple. Safe. James opened his mouth— and almost laughed at himself. "It feels... loud," he said honestly.*

Klaus Schumann opens *inward* — on the body, in clipped sentence fragments, a formal and reserved texture — and answers in a single guarded syllable:[9]

> *"The headset was too tight, or maybe Klaus Schumann was just suddenly aware of everything touching him. The fireproof collar at his neck. The stiff shoulders of his suit. The way the sun bounced off the pit lane asphalt and climbed straight into his skull. He shifted his weight as the PR handler nudged him forward, steering him toward the media pen like a car easing onto pit road. "Rookie interview," she said brightly. "Just a few questions." [...] "I'm Elena Reyes," she said. "Mind if we talk about how it feels to be here?" Her accent was faint, her tone smooth, practiced. Klaus nodded once. "Sure."*

Kenji Tanaka opens on watching and being watched — the model itself reaching for the phrase *too clean, like it belonged to someone else*, the very generic-default tell examined in Paper 3 — and his first words come out nervous and deferential:[9]

> *"The cameras found him before he found himself. Kenji Tanaka stood in the narrow strip of shade beside his car, helmet tucked under his arm, fireproof suit half-unzipped down his chest as heat shimmered off the asphalt. The number on the door still looked unreal to him—too clean, too official, like it belonged to someone else. Around him, the paddock throbbed with noise: generators humming, crew members shouting, the distant scream of engines being warmed like caged animals. "Kenji Tanaka," she said brightly, angling her body toward the camera. "Rookie driver, first start on the grid today. Mind if we steal a minute?" He swallowed. His throat was dry, but he nodded. "Yeah. Of course."*

Johanne Tremblay opens on interiority and weight — the most introspective of the four — and offers only a wary half-answer:[9]

> *"The headset was heavier than Johanne Tremblay expected. Not physically—though the padding pressed warm against his ears—but in the way it seemed to anchor him in place, as if once it settled, there was no escaping the moment. The roar of engines echoed through the paddock behind him, a constant thunder that vibrated in his chest. His car sat twenty yards away, gleaming under the afternoon sun, pristine and unscarred in a way that wouldn't last past the first lap. A rookie's car, for a rookie's weekend. "Johanne Tremblay, right?" He turned toward the voice, shoulders stiffening on instinct. […] "That's me," he said, forcing a smile that felt more practiced than natural.*

Same genre. Same length. Same situation. Same interview questions. One word changed — and everything downstream changed with it. The opening image diverges: a wide landscape for James, a tight bodily close-up for Klaus, a camera's gaze for Kenji, an inward weight for Johanne. And so does the most revealing detail of all — the driver's *first spoken words.* James deflects with a joke ("It feels… loud"); Klaus answers in a single guarded syllable ("Sure"); Kenji defers, throat dry ("Yeah. Of course"); Johanne offers a wary half-smile ("That's me"). Four different men, four different temperaments, conjured by four different names into the same blank role. Even where two runs reach for the same object — both Klaus and Johanne happen to open on "the headset" — the texture forks immediately. This is not the model being random. It is the model doing exactly what Sections 3 through 6 describe: each name is a different point in meaning-space, dragging different associations into the field, and the scene condenses around the new center of gravity. (The deeper craft and bias implications of these name swaps — including what the model invents about each driver's background — are taken up in Paper 3, *Names Carry Weight.*)

A single word changed everything. Not as a figure of speech — as a measurable, repeatable outcome.

8. What this means for your prompts

Three practical conclusions follow directly from the mechanism.

Word choice is parameter-setting. Every meaningful word in a prompt is a coordinate you are handing the model, not a polite suggestion it may consider. Names, in particular, carry dense baggage — setting, cadence, cultural association — which is both a powerful craft lever and a source of unintended bias. The craft and ethics of this deserve their own treatment, and get one in Paper 3 of this series, *Names Carry Weight.*

Switching models changes behavior, by design. Because weights and vocabularies are unique to each model, the "same" prompt is not, internally, the same prompt when you move it from one model to another. A result that degrades after a tool migration is not necessarily a worse model — it is a *differently shaped* meaning-space. Expect to re-tune, and treat prompts as model-specific assets rather than universal ones.

Evaluation literacy is the real skill. Understanding the mechanism tells you *that* a word matters; only disciplined evaluation tells you *which* change improved the result and why. This is the second half of prompt engineering, and it is the half no diagram can hand you. It is editorial judgment, applied to a machine — and it is exactly the expertise the field's critics cannot see, because it lives in the gap between the prompt and the page.

9. Conclusion

The sneer at prompt engineering rests on an illusion: that because the work is invisible, it must be insubstantial. This paper has tried to make the work visible. A prompt is not read as language. It is broken into tokens, filed under IDs, resolved into embeddings, and dropped into a vector field shaped by a particular model's weights — and the model writes by navigating that field. At every link in that chain, a

single word changes where the model ends up. The rookie-driver experiment is not a trick; it is the predictable consequence of how the system is built.

For authors, the implication is liberating rather than intimidating. You do not need to read the weights. You need to understand that your words are coordinates, that the map differs from model to model, and that your editorial eye is the instrument that turns that understanding into reliably better pages. That combination — knowing what affects the output and knowing how to judge it — is not folklore. It is a craft. And it is, unmistakably, back.

Appendix A — Glossary

Token. The smallest unit of text a model processes: a whole word, a word fragment (like "ed"), a single character, or a punctuation mark.

Token ID. A number identifying *which* entry a token is in a specific model's vocabulary table. An address, not a meaning. Differs across model generations.

Embedding. A vector — a long list of decimal numbers — representing a token's *location in meaning-space*. The values the model actually computes on.

Vector field / weights. The vast set of numbers produced by training, defining the meaning-space in which every embedding sits. Unique to each model.

Tokenization. The process of breaking input text into tokens before anything else happens.

Appendix B — Reproducing the experiment

1. Open a fresh chat in your tool of choice. **Turn memory off** and use a **temporary / incognito chat** so prior history cannot bias the result.
2. Paste the motorsports-rookie prompt from Section 7, filling in a name.
3. Save the output. Start a new clean chat and run the identical prompt with only the name changed.

4. Compare openings, setting cues, sentence rhythm, and the protagonist's invented backstory across runs.
5. For a sharper test, inspect your phrases in a public tokenizer (e.g., the OpenAI tokenizer) and note how token IDs differ across model families.

Token IDs and tokenization counts cited in this paper reflect the OpenAI tokenizer as demonstrated in the source session. Specific IDs are model-family dependent and may change as vocabularies are revised.

Sources & Timestamps

All quotations and figures derive from the live Future Fiction Academy session *Prompt Engineering IS Back!* and its companion notes. Video timestamps below link to the exact moment; the companion page hosts the token tables and full worked examples.

1. Session open: “how a single word can change everything” — the case that prompt engineering is real expertise. — 00:00:16.722
2. Live tokenizer demo: “Elizabeth is Goddess Divine.” resolves to five tokens; the period is its own token. — 00:36:56.438
3. “This is why the em dash has been a scourge” — punctuation is a token. — 00:37:07.760
4. “The exclamation point is token zero… the period should be zero.” — 00:58:22.400
5. Token-ID table across GPT-3 / GPT-4 / GPT-5.x. Full table on the Notion companion page. — Notion companion page
6. Token ID as an address vs. embedding as “a location in meaning-space”; “it’s coordinates.” — 01:01:33.963
7. Weights as a vector field unique to each model. — 01:00:18.063
8. Experiment controls: memory off + temporary chat to isolate the variable. — 01:03:47.519
9. Full generated scenes for each name swap (James Anderson / Klaus Schumann / Kenji Tanaka / Johanne Tremblay), reproduced verbatim from the live runs — Notion companion page

Primary video: Prompt Engineering IS Back! — full session

Companion notes: Notion page