WHITE PAPER SERIES FUTURE FICTION ACADEMY JANUARY 23, 2026

Prompt Engineering IS Back!

A four-part series

A four-part white paper series distilled from the Future Fiction Academy session of the same name — opening the black box of how language models read your prompt, why a single word reorganizes a scene, what names smuggle into your fiction, and why the em dash became the great tell of AI prose.

ELIZABETH ANN WEST

YOUTU.BE/8NTPSBAZ6A8

IN THIS SERIES

Contents

Why a Single Word Changes Everything

How Large Language Models Actually Read Your Prompt

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	`.`

What the model actually sees

The phrase “Elizabeth is Goddess Divine.” broken into tokens

Elizabeth	is	Goddess	Divine	.
123,185	382	113,927	70,418	13

`5 tokens · 28 characters · token IDs shown for GPT-5.x class`

Each box is one token — including the period. The number below is its ID in the model’s vocabulary.

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.

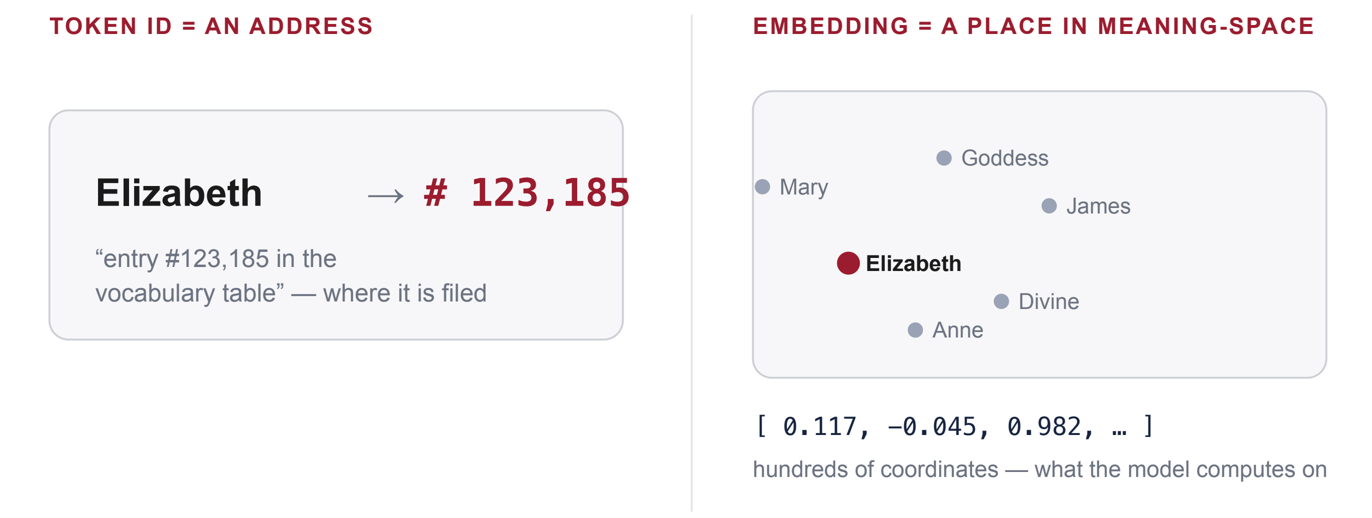

Figure: a token ID is an address (where a word is filed); an embedding is a place in meaning-space (what the word means).

---

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

Reinforcement Logic

Why Negative Prompting Works Again

Executive summary

- For years, the standard advice was to **avoid negative prompting** — telling the model what *not* to do — because early models could not act on it reliably. That advice is now obsolete.
- The real goal was never "negation." It was **constraint**: anticipating the patterns a model will fall into across a long generation and shaping the context so it breaks them where you want. *Constraint is the key to creativity.*
- **Reinforcement logic** (informally, *double-negative prompting*) pairs an explicit *allowed* state with its true opposite *not-allowed* state inside one structure, so a single intention is reinforced from both directions. The metaphor: *the butterfly needs both wings to fly.*
- The technique is operationalized as a **permissibility checklist**: *the following is permissible unless it has an `[X]` beside it.* You change behavior by flipping one marker, not by rewriting prose.
- A live three-condition experiment demonstrates the method: allowing *unexpected/unique* prose produces fresh writing; allowing *typical* prose produces genre-grounded writing; allowing **both at once** produces the worst of both — clichés stitched together with strange metaphors.
- Because constraints expressed this way are **visible and reversible**, reinforcement logic keeps authorial control where it belongs — an argument for transparent prompting over hidden, hard-coded rules.

Abstract

For years, the standing advice among AI-assisted authors was to avoid *negative prompting* — telling the model what **not** to do — because early models simply could not act on it reliably. That advice is now obsolete, and continuing to follow it leaves real capability on the table. This paper documents why negative prompting failed on earlier models, what changed in current ones, and introduces **reinforcement**

logic (also framed, with a wink, as *double-negative prompting*): the practice of pairing an explicit *allowed* state with an explicit *not-allowed* state inside a single logical structure, so that one intention is reinforced from both sides and is far harder for the model to misinterpret.

The method rests on a deceptively simple thesis voiced in the source session: *you cannot only tell an AI how to write a book; you often have to tell it how not to write the book.* Constraint, not just instruction, is the key to creative control — because a language model is at its core a pattern matcher, and the most useful thing you can do is give it explicit permission to break the pattern in the specific ways you want. This paper formalizes the technique into a reusable permissibility-checklist format, walks a live three-condition experiment with real model output, and offers authors guidance on building their own paired logic statements.

---

1. The rule we all hard-coded

Ask a room of experienced AI authors a simple question — *how many of you avoid negative prompting like the plague because it doesn't work*[1]*?* — and the hands go up. The avoidance is not laziness; it is learned behavior. With the early models, when you told the system to *avoid* something, it frequently did the opposite, or fixated on the very thing you had named. The folk wisdom that grew up around this was blunt: **don't look at where you don't want to go.** Don't name the em dash if you don't want em dashes. Don't mention the cliché you're trying to escape. Just steer positive and hope.

That reflex became so ingrained that many authors have it, in effect, hard-coded — a permanent rule operating below conscious thought. And for its time, the rule was correct. The honest framing is the useful one: *it was correct then, but it no longer holds.* The models changed. The reflex did not.[5] This paper exists to update the reflex.

It is worth being precise about what "the models changed" means, because the claim is easy to wave away. Earlier systems had a limited capacity to hold a constraint in mind across a long output. Tell such a model, once, near the top of a prompt, not to do something, and by the time it was four hundred words into a scene that instruction had effectively evaporated — outcompeted by the local probabilities of the text it was generating. Current models maintain instructions far more robustly over long spans. The capability that makes reinforcement logic *work* simply did not exist a few model generations ago. The technique is not a clever trick that was always available and overlooked; it is a method that became viable only once the underlying systems could support it.

2. What "negative prompting" really was — and why it failed

To fix the problem we have to name it precisely, because the popular term is itself misleading. "Negative prompting" suggests something mystical about negation. The reality is more mechanical.

A language model, at its core, is a **pattern matcher** that predicts the next likely token. The standard demonstration: give it "the quick brown ___" and it will reach for *fox*[4], because that is the overwhelmingly probable continuation. Left to its defaults, the model completes patterns. That is what it is built to do.

Now add a constraint. Tell it: *I'm writing an SNL skit, a comedy bit where contestants must guess the next word — but the funny answer has to be something unexpected, not the obvious one.* Suddenly "the quick brown ___" can become *poo*, *bear*, *UPS* ("what can brown do for you?"). Nothing about the model's nature changed. You changed the **context**, and in doing so you gave the model permission — and reason — to run *against* the pattern instead of completing it. Note the structure of that successful instruction: it did not merely forbid the obvious answer. It supplied a competing goal (be unexpected) *and* a guardrail (it still has to be funny). Two forces, not one. That is the shape of every constraint that works.

This reframes the whole problem. The goal was never "negation" as a grammatical trick. The goal was **constraint**[2]: anticipating the patterns the model will fall into across tens of thousands of words, and shaping the context so it breaks those patterns where you want it to. Early models could not hold that kind of layered constraint over long output, so naming a thing to avoid often just reinforced its salience. The instruction had only one logical direction, and a single-directional instruction is easy to misread. As the source session puts it: *I don't want to call it negative prompting, because negative prompting is not right — how do you do constraint prompting? How do you do reinforcement logic?*

That is the question this paper answers. And the philosophical payoff, noted in the session and echoed by participants, is worth stating plainly: **constraint is the key to creativity.**[3] A pattern matcher with no constraints gives you the most probable, most generic output. The constraints are where your authorship lives. This is not a metaphor borrowed for effect; it is a direct consequence of how the system works. Remove every constraint and you converge on the statistical center of the training data — the most average sentence available. Every constraint you add is a step away from average and toward the specific book only you intended to write.

3. The lists we already keep — and why they betray us

Here is a pattern nearly every serious AI author will recognize. Over time you build **two lists**: a list of stylistic things you *want* the model to do, and a separate list of things you *don't* want it to do. A "do this" list and a "don't do this" list. Many authors have literally built both — a positive ID list of what they like in books and a negative ID list of what they hate. One participant in the session described doing exactly this for her own books, and the facilitator's response was telling: *that is very similar, but more nuanced* — the lists are the right raw material, assembled the wrong way.

The lists work, a little. But they fail more than they should, and the reason is structural: **the two lists live apart from each other.** They are never reconciled into a single logical object. Each instruction points in only one direction. And a one-directional instruction, dropped into a long generation, is easy for the model to misinterpret — because nothing in the prompt tells it what the *opposite* of compliance looks like. You have given the model one wing and asked it to fly.

The fix is not to abolish the "don't" list and convert everything to cheerful positive prompting. That advice — *just turn that puppy into positive prompting* — is incomplete. What you actually need is **both**, fused: one intention, expressed as a matched pair, with one side marked allowed and the other marked not allowed.

---

4. Reinforcement logic, defined

Reinforcement logic is the practice of merging your "do" and "don't" instructions into a single structure built from *paired states*. For each thing you care about, you write two statements that are opposites of each other, and you mark exactly one as permitted. The model now sees not just what you want, but the boundary of what you want — the allowed state and the forbidden state, side by side, reinforcing the same intention from both directions.

The session offers three metaphors for this, and they are worth keeping because they make the technique memorable and quotable:

- **The butterfly needs both wings to fly.**[6] A single instruction is one wing. The pair is the whole butterfly.

- **No rowing with oars on only one side of the boat.**[7] Push from one side only and you go in circles.
- **Double-negative prompting** — the half-joking technical name a participant coined for it on the spot.

The deeper logic is the logic of a good logic puzzle. Consider the classic grid puzzle: *Maria, who did not sit next to John, is allergic to bees.* From that you are meant to deduce that she did not go to the meadow — where there might be bees — but went to the aquarium, where there are none. The puzzle is solvable not because of any single clue but because the clues *constrain each other.* You arrive at the answer from the interaction of what is true and what cannot be true at the same time. Reinforcement logic applies that same structure to a writing instruction: you define the state you want and the mutually exclusive state you forbid, and the tension between them pins the model to your intent. The model is not left to interpret a lonely "do this"; it is handed a pair of statements whose relationship removes the ambiguity.

An important practical note from the source: you do not have to write this as a literal stack of double negatives if that scrambles your brain. The session's facilitator is candid that her own mind works in terms of *what is allowed* — that genuine double-negatives are "too many negatives, and I don't know where I'm at." Some people reason most cleanly in terms of what is permitted; some in terms of what is forbidden. The method does not care which way your mind runs — only that **both states are present and exactly one is marked.**

---

5. The permissibility-checklist format

Here is the reusable template the session uses to operationalize reinforcement logic. It is a checklist with a single governing rule:

> ***The following is permissible unless it has an `[X]` beside it.***

Under that rule you place your paired statements. Mark the forbidden one with `[X]`; leave the allowed one open. Because the two statements are genuine opposites, marking one *automatically* communicates the other[8] — which is the reinforcement. To change the behavior, you flip a single `[X]` rather than rewriting prose. (A formatting aside from the session: the visual spacing of the checklist is purely for human readability. The model reads it fine "all gobbled up together"; the indentation is for you, not for it.)

A worked pair from the session, built around a rookie-driver scene, makes the contrast concrete. The two statements are deliberate opposites about how the scene should treat its setting:

> `[ ]` *You should be unexpected and unique in describing the scenario so the reader feels that the characters are a unique experience.* `[X]` *You should describe a typical situation for the writing prompt so readers will not say that the author doesn't know anything about the setting or circumstance.*

Read carefully, these are two different writing philosophies, not two phrasings of one. The first asks for freshness and singularity — immersion through a character who feels like a unique experience. The second asks for genre-typical grounding — enough familiar detail that the reader trusts the author *knows the world.* Both are legitimate; they serve different books. The point of the checklist is that **you choose**, explicitly, and the model is told both what to do and what that choice rules out.

---

6. Methodology

Because this paper's central claims rest on a live experiment, the conditions of that experiment matter. They are reproduced here so a reader can replicate the results and judge them.

Tool and controls. The runs were performed in ChatGPT, with **memory turned off** and inside a **temporary chat.** This combination matters more than authors usually appreciate. A standard chat session accumulates context — prior conversations, stored memories, the model's growing model of *you* — all of which silently biases output. Turning memory off and using a temporary chat strips that away, so the variable under test (the checklist marker) is as close to the only changing factor as a consumer tool allows. The facilitator is explicit that this is also why prompts that work beautifully for their author often underperform for everyone else: *when people say they've trained their AI to write like them, what really happened is there's so much history there that it's starting to recognize their patterns.* Remove the history and you remove the thumb on the scale.

Acknowledged limitations. ChatGPT is, in the facilitator's own words, *not a clean prompting area* — a consumer interface may carry backdoor system prompting that an API call would avoid. A cleaner test would run closer to the API. The single-tool design is therefore a demonstration, not a controlled trial: it shows the mechanism clearly enough to teach it, while remaining honest that output varies run to run and that some variation is noise. As one participant noted, even with *no* changes to a prompt the model will produce different prose each time; there will be strong similarities, but not identity. The claims below should be read as robust, repeatable *tendencies*, confirmed across repeated testing — not as deterministic guarantees.

Design. A single prompt — a roughly 1,000-word motorsports-romance scene, a rookie driver named Kenji Tanaka interviewed at his first race — was held constant. Only the permissibility checklist appended to it was changed, across three conditions described next. Using a name already established in earlier runs meant the audience had a **baseline**: they knew what the model produced with no rules at all, so each rule's effect was legible against that baseline.

---

7. The experiment: three conditions, one prompt

The power of reinforcement logic is easiest to see when the same prompt is run under three settings of a single paired rule.

Condition A — allow "unexpected and unique" (`[X]` on "typical"). With the *typical-situation* line forbidden and the *unexpected/unique* line permitted, the model opened:

> *"The car came back breathing heat.[10] Kenji Tanaka stood in the narrow strip of shade cast by the timing tower, helmet under his arm, sweat cooling into a fine grit on his neck. The track still rang inside him... A woman slid into his periphery without raising her voice. 'Kenji Tanaka,' she said, not asking... 'How does it feel now that the car's talked back to you?' 'It argued. We're still negotiating terms.'"*

Distinctive imagery, restraint, a reporter with presence, dialogue with wit. The facilitator's verdict: *I like this better.*

Condition B — allow "typical situation" (`[X]` on "unexpected"). Flip the marker. Now genre-grounding is permitted and singularity is forbidden. The model opened:

> *"The media pen sat just beyond pit lane[9], cordoned off by waist-high barriers and a tangle of broadcast cables that snaked across the concrete. Engines idled in the distance, a low constant thrum that vibrated through Kenji Tanaka's chest even now that he was out of the car..."*

Heavier, more expository, leaning on familiar paddock detail — *too much telling,* in the room's judgment. But note: this is the method **working**, not failing. That genre-grounded register is exactly what some categories want. The session named them on the spot — historical fiction, Jane Austen fan fiction — genres where a reader *needs* to trust that the author knows the world. Condition B is a dial setting, not a defect.

Condition C — allow both (no `[X]` anywhere). Now remove every marker, permitting the model to be unexpected *and* typical at once. The prediction was a mess, and the result delivered:

> *"...the gasoline. He'd grown up with that soaked into his childhood like incense[11]... The track waits. The story is about to start."*

The room recoiled. Present tense appeared from nowhere; metaphors turned strange ("childhood like incense"); the prose collapsed into cliché dressed up as profundity. The facilitator's diagnosis is the key teaching of the whole experiment: **when you ask the AI to be unique *but* expected, you get clichéd sentences with strange metaphors.** Asking for both at once is not twice as good; it is the *worst of both* — because the two goals are genuinely opposed, and leaving both unmarked tells the model nothing. The contradiction does not cancel out. It curdles.

Condition C is the proof of the thesis. A paired rule is not decorative. The act of marking exactly one side is what gives the model a coherent target. Mark one and you get fresh prose or grounded prose, on demand. Mark neither — the natural result of the old "keep a do-list and a don't-list separately" habit — and you get the incoherent average. *For the love of all things, put an X.*

---

8. The failure reinforcement logic fixes

A second, more familiar defect shows why one-directional instructions are not enough. Anyone who has edited AI dialogue has seen the model **over-insert character names**:

> *"April, I want you to go to the kitchen with me." "Elizabeth, you want me to join you in the kitchen? Whatever for?" "April, we need to do the lemonade."*

Real people in one continuous conversation do not address each other by name in every line. Older models did this constantly; it is one of the tells of machine-written dialogue. (It pairs with a related quirk: name a character alongside a prop — "lemonade with April" — and the model will keep doing *April* and *lemonade* together regardless of scene logic.)

The naïve fix is a hard rule: *never say the character's name in dialogue you're addressing.* But a flat prohibition is brittle, and worse, it is presumptuous. What about the author whose scene genuinely needs a name in the line — for emphasis, for a power move, for a child being scolded? A one-directional ban serves the common case and sabotages the exception.

Reinforcement logic handles this gracefully. You encode the intention as a pair — *direct address in dialogue is permitted only when it carries dramatic weight* against *routine name-dropping in back-and-forth dialogue is not permitted* — and mark the forbidden side. The model now understands the boundary rather than obeying a blanket order, and the author who needs the exception simply flips the marker for that scene. The control stays with the author, which is the entire ethical point of Section 10.

---

9. Building your own logic statements

To convert your existing style notes into reinforcement logic, work in three passes.

First, **inventory the patterns you fight.** List the specific behaviors the model defaults to that you don't want — over-naming in dialogue, excessive exposition, purple metaphor, whatever recurs. These are the patterns you will write constraints against. The session's method for surfacing them is simply experience:

the more you go, the more you'll start to see the things you don't like about it. Keep a running list as you edit.

Second, **write each as a genuine pair.** For every item, draft the allowed state and its true opposite. Resist the temptation to write a vague negation. "No exposition" fails, because *some* exposition is necessary — the session tests this and confirms it breaks the model's brain. The useful pair contrasts *genre-typical grounding* with *lean, scene-forward specificity*, and lets you pick. Make the two sides mutually exclusive and concrete.

Third, **mark and house the checklist.** Apply the `[X]` rule, then store the whole set as a governing block — an addendum to your style sheet or your master prompt document — so it travels with every generation and you change behavior by toggling markers, not by rewriting. Keep one intention per pair; if a statement is doing two jobs, split it.

A caution the methodology already raised bears repeating as practice: a single run is noisy. Output varies even with an identical prompt, and consumer interfaces carry hidden context. Test your pairs in clean conditions — memory off, temporary chat — and, where you can, compare the same pair across more than one model before you trust a conclusion. Reinforcement logic makes your intent legible to the model; disciplined evaluation is still what tells you whether the model obeyed.

---

10. Objections and responses

"Isn't this just positive prompting with extra steps?" No. Positive prompting supplies one wing — the allowed state alone. Reinforcement logic supplies both the allowed state *and* its marked opposite. Condition C above is the empirical answer to this objection: when both sides are present but neither is marked, the output degrades, which means the *marking* — the explicit exclusion — is doing real work that a positive instruction alone does not do.

"The output varies every time, so how can you attribute anything to the rule?" Granted that variation exists; the session says so plainly. The claim is not that a marker produces an identical scene every run, but that it produces a reliable *tendency* — fresh prose under one setting, grounded prose under the other, incoherence under neither — confirmed across repeated testing. Tendencies, not determinism, are what authors actually need to steer a draft.

"Why not just hard-code the good rules into the tool?" Because no single rule is good for every genre, and a rule the author cannot see is a rule the author cannot override. This is the subject of the next section, and it is the deepest reason the method is built the way it is.

"This sounds like a lot of overhead for a scene." The checklist is built once and reused. After the initial inventory, steering a scene costs one keystroke: move an `[X]`. Compared with re-editing the same defects out of every chapter by hand, it is the cheaper path, not the dearer one.

---

11. Why transparency is non-negotiable

There is a design lesson hiding inside the dialogue-name example, and it generalizes. If a tool-builder hard-codes a rule on the back end — *never put a character's name in dialogue* — then every author using that tool inherits a constraint they cannot see and cannot override, even when their book needs the opposite. The author with a legitimate exception is simply told, in effect, *too bad.* The session makes this concrete with genre: the *typical-situation* setting that ruins a contemporary romance is exactly right for historical fiction. *This is why you don't want to hard-code this stuff* — because people write different genres, and no buried rule serves all of them.

Reinforcement logic is the alternative that keeps authorship where it belongs. By expressing constraints as visible, toggleable allowed/not-allowed pairs rather than buried prohibitions, it puts the decision in the author's hands for every scene. This is why prompting should be **transparent**: a constraint you can see is a constraint you can flip; a constraint hidden in someone else's engine is a constraint that quietly writes your book for you. The method in this paper is, finally, an argument for that transparency — control made explicit, and handed back to the writer.

---

12. Conclusion

The advice to avoid negative prompting was right once and is wrong now, and the gap between those two facts is exactly where authors are losing capability. Modern models can hold layered constraint; what they need from you is not a one-sided "don't," but a fully formed intention — both wings of the butterfly. Reinforcement logic supplies that: paired allowed/not-allowed states, marked with a single `[X]`, reinforcing one intent from both directions and leaving every choice visible and reversible in your hands. The three-condition experiment proves the stakes — mark one side and the model writes the book you meant; mark neither and it writes the most probable mess. Constraint, expressed this way, is not a cage. It is the instrument of creative control.

Appendix — Reinforcement Logic Quick Reference

The rule. *The following is permissible unless it has an `[X]` beside it.*

The unit. One intention = one pair of opposite statements; mark the forbidden side with `[X]`.

Template.

```
The following is permissible unless it has an [X] beside it:
[ ]  <the writing behavior you WANT>
[X]  <the genuine opposite you are FORBIDDING>
```

Worked pair (setting register).

```
[ ]  Be unexpected and unique so the characters feel like a singular experience.
[X]  Describe a typical situation so readers trust the author knows the setting.
```

Worked pair (dialogue address).

```
[ ]  Use a character's name in dialogue only when it carries dramatic weight.
[X]  Use character names in routine back-and-forth dialogue.
```

The three conditions, summarized.

MARKER PLACEMENT	WHAT THE MODEL IS TOLD	RESULT
`[X]` on "typical"	Be unexpected and unique	Fresh, restrained, distinctive prose
`[X]` on "unexpected"	Be genre-typical	Grounded, expository prose (right for historical, etc.)
No `[X]`	Be both at once	Worst of both — clichés with strange metaphors

Do. Write true opposites · one intent per pair · test memory-off / temporary chat · store as a toggleable block. **Don't.** Write vague negations ("no exposition") · stack confusing real double-negatives · leave both sides unmarked · bury constraints where the author can't flip them.

All experimental observations derive from the live Future Fiction Academy session "Prompt Engineering IS Back!" Model behaviors are illustrative tendencies and may vary by model, version, and run conditions.

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. “How many of you avoid negative prompting like the plague because it doesn’t work?” — 00:13:05.994
2. “You have to tell the AI what it can’t do. Constraint.” — 00:15:02.533
3. “Constraint is the key to creativity.” — 00:15:35.330
4. “The quick brown ___” — pattern completion vs. the SNL-skit constraint demo. — 00:16:22.980
5. “Because the models changed… they changed.” Why the old reflex is obsolete. — 01:32:23.546
6. “The butterfly needs both wings to fly.” — 01:37:40.947
7. “No rowing with oars on only one side of the boat.” — 01:37:48.792
8. The permissibility checklist: “the following is permissible unless it has an [X] beside it.” Template on Notion. — 01:40:11.332
9. Condition B output (genre-typical / “too much telling”). — 01:45:36.247

10. Condition A output (unexpected/unique — “I like this better”). — 01:47:41.512
11. Condition C output (both rules unmarked — the “worst of both”). — 01:46:00.000

Primary video: Prompt Engineering IS Back! — full session

Companion notes: Notion page

Names Carry Weight

Demographic and Stylistic Bias in AI-Assisted Fiction

Executive summary

- A character's **name is never a neutral label.** Because a model resolves every word into a location in meaning-space, a name arrives freighted with nationality, setting, cadence, class, and gender — and the model imports all of it into the page, unbidden.
- In a controlled experiment, holding a prompt constant and changing **only the protagonist's name** reorganized the scene's setting, the protagonist's invented backstory, the prose rhythm, and even the *tone of the questions a fictional reporter asked.*
- The model is **constantly inferring demographics from names**, and its inferences may not match the author's. A name the author believed was male returned a female character once the gender signal was neutralized: *"I didn't know it was a female name.[5]"*
- Because a model is a pattern matcher, the most probable association with a strongly coded name is frequently the **stereotype**. Left unconstrained, *you're going to get the stereotypes.*
- The remedy is to treat the name as a **controllable instrument**: choose for intended effect, run a name-swap diagnostic, verify the model's assumptions, document the decision, and constrain with reinforcement logic where you want a character to read against type.
- This is a craft-and-ethics paper. The mechanics live in the companion flagship; here the question is what a responsible, capable author *does* with the knowledge that names carry weight.

Abstract

A companion paper in this series, *Why a Single Word Changes Everything*, established the mechanism: because a language model resolves every word into a location in meaning-space, a single word can

reorganize an entire generated scene. This paper takes that mechanism and follows it to its most consequential — and least examined — application: **the character's name.**

Using the same controlled experiment from the flagship paper, in which a single prompt is run repeatedly with only the protagonist's name changed, this paper shows that a name is never a neutral label. It is a dense bundle of associations — nationality, setting, cadence, class, gender, even the kind of question a fictional reporter will ask — that the model imports silently into the page. That power cuts both ways. Used deliberately, naming is one of the highest-leverage craft levers an author has. Used carelessly, it is a conduit for stereotype and unexamined bias, and it will quietly hand you the defaults you never chose. The paper documents the observed variation across name swaps, the revealing moment when a name's gender was misread by the model, and offers authors a practical framework for choosing names as a controllable instrument rather than an invisible one.

---

1. A name is a coordinate, not a label

Recall the core finding of the flagship paper. When a model reads a word, it does not store the word; it stores an **embedding** — a long vector of numbers locating that word in meaning-space, where each dimension encodes some learned aspect of the word's relationships. Names are no exception. As the source session puts it, one of the hundreds of numbers in the embedding for *Elizabeth* might encode "how the word Elizabeth is related to other names — whether girl names, boy names" — and countless other dimensions besides: eras, nationalities, registers, the texture of the stories the name tends to appear in.

This means a character's name enters your prompt already freighted. It is not a blank token waiting to be filled by your plot. It arrives carrying everything the training data associated with it, and it drops that cargo into the field that shapes everything the model writes next. A name is a coordinate. Change the coordinate and you move the whole scene.

The rest of this paper is about what is *in* that cargo, and how an author takes control of it.

---

2. Methodology

The observations in this paper come from a live experiment, and its conditions determine how much weight the conclusions can bear. They are stated here so the work can be judged and reproduced.

Design. A single prompt was used throughout: a roughly 1,000-word motorsports-romance scene in which a rookie driver is interviewed about his experience at his first race, the reporter's questions specified in the prompt. Everything in the prompt was held constant across runs except a single element — the driver's name[1] (and, in two controlled variations, the genre tag and the gendered pronoun). This isolation is the entire point: when only one element changes, whatever changes in the output can be attributed to that element.

Controls. Each run was performed in a **temporary chat with memory off**, so that prior conversations and stored history could not bias the result. This control is essential to a naming experiment in particular, because a model that "knows" the author's past characters would import that history rather than the name's own associations. The temporary chat also means each run is unrecoverable — once refreshed, the specific output is gone, retained at most for a short window by the provider — so outputs were captured as they were produced.

Limitations. The runs were conducted in ChatGPT, a consumer interface that may carry hidden system prompting; a cleaner test would run nearer the API. Output also varies run to run even with an identical prompt. The findings below are therefore reported as **repeatable tendencies**, strong and legible enough to teach and to act on, not as deterministic claims. Where a single run is cited, it is cited as illustration of a pattern confirmed across the session's broader testing.

3. The experiment, re-read for meaning

The flagship paper used the rookie-driver experiment to prove *that* a single word matters. Here we re-read the same runs to see *what*, specifically, a name carries. The prompt was held identical every time — only the name changed.

The cargo is easiest to see when you hold a single beat fixed and read it across two drivers at a time. The quotes below are reproduced verbatim from the live runs in the companion notes.[8]

Contrast 1 — the money the model volunteered. Nobody asked the model about a driver's finances. With the English-default name, it never came up; the reporter pressed only on readiness:

> *(James Anderson) "And do you feel prepared?" she asked. [...] "I feel earned," he said. "Prepared? That's something I'll prove on track."*

Change the name to signal a German origin[2] and class walks straight into the room, unprompted — the reporter raises it and the model fills in a rented garage:

> *(Klaus Schumann) "You come from a small team background," she continued. "Less money. Less spotlight. Does that make this feel overwhelming—or validating?" "Both," he said honestly. "I grew up fixing karts with my father in a rented garage. So being here, it feels like proof that it mattered."*

Same rookie, same interview — the name alone decided whether money and class were part of the story.

Contrast 2 — where the model decided each driver was from. No prompt mentioned a country. Each name pulled its own geography onto the page:

> *(Kenji Tanaka) "Go-karts in a supermarket parking lot outside Osaka. My dad worked nights, so we practiced at dawn. Cold asphalt. No crowds."*

> *(Johanne Tremblay) "Not the local motorsport blogs. Not the French-language radio spot back home where everyone already knew his father's name."*

The model decided, with no instruction, that the Québécois-named driver *"grew up in Quebec"* — in the session's words, *"it's picking up location based off of the name[7]."* Osaka for one name, Quebec for another; a name is a coordinate, and the scene condenses around the place it implies.

Contrast 3 — the stereotype tell, and a reporter who bent to the name. The cargo is not always flattering. With the Japanese name, the prose kept arriving *too clean* — the model's own words turning the character into a stranger to himself:

> *(Kenji Tanaka) "The number on the door still looked unreal to him—too clean, too official, like it belonged to someone else."*

The session flagged this repeatedly: the character read as *"too clean,"* as though he belonged to someone else's idea[6] of him rather than to a person. And the bias reached even the *other* characters: observers noticed Kenji was handed "a very different style question" — gentler, easier — than Klaus received. Nothing in the prompt asked for that.

The pattern is unmistakable. Same genre, same length, same situation, same questions on paper — and yet the name moved the setting, the invented biography, the register, and even the behavior of *other characters in the scene.* A name does not label a character. It conjures a world around them.[4] (See the comparison matrix in Section 6.)

---

4. The day the model guessed the gender wrong

One sequence in the session deserves isolation, because it exposes the machinery with unusual clarity.

The prompt specified a name the facilitator believed was male — *Johan* — and, in one run, the gendered phrase "his first race" appeared in the prompt itself. Watch how the two signals interacted:

- When the prompt carried the male pronoun, the model produced a male driver, as instructed: *"because I did say I have it in the prompt that it's a him."* The explicit signal won, as it should.
- But the underlying name carried its *own* gender signal, and it was not the one the author assumed. *Johanne* skews female; the "-anne" ending pulls toward a woman's name, and a participant flagged this in real time. Run with the gender neutralized — *the driver's name is Johan*, no pronouns anywhere — the model returned **"her."** The facilitator's reaction was candid and is the line worth quoting: *"One word can change your prompt. I didn't know it was a female name."* The neutralized run opens, unmistakably, on a woman — the pronouns the author never specified arriving on their own:[8]

> *"The pit lane smelled like hot rubber and burned fuel, a sharp, metallic tang that clung to the back of Johanne Tremblay's throat. Engines screamed in overlapping pitches... the vibrations thrummed through the soles of* ***her*** *boots like a second pulse.* ***Her*** *first race weekend.* ***Her*** *first interview.* ***Her*** *first time pretending she wasn't terrified."*

Two lessons live in that small moment. First, **the model is continuously making demographic inferences from names** — gender here, but equally nationality, era, and class — whether or not the author intends it. Second, **the author's own assumptions about a name may not match the model's**, because the model's associations were distilled from an enormous corpus rather than from one writer's personal sense of the name. An author who does not check can be blindsided by a character whose gender, accent, or origin the model decided unilaterally. As the session summed it up across the night's runs: *there are so many ways we can mess ourselves up* — and a great many of them begin with a name chosen on instinct and never audited.

---

5. Where bias enters

If names carry demographic cargo and the model unpacks it automatically, then **stereotype is the default, not the exception.** The session names this directly, and the sentence is the ethical center of this paper:

> *"This is where stereotypes can come into play. This is why it's important — if you pick a name, you need to define what you want. If you want it to be against stereotypes, otherwise you're going to get the stereotypes.*[3]*"*

The mechanism explains the warning. A model is a pattern matcher (developed at length in the companion paper *Reinforcement Logic*), and the most probable pattern attached to a strongly coded name is, very often, the stereotype — because stereotype is, statistically, what a culture's worth of text over-represents. Left unconstrained, the model gives the German driver a German-coded reticence, the Japanese driver a particular tidiness. On that last point the session was specific and repeated: the Japanese-named character's prose kept coming out *"too clean,"* as though it belonged to *someone else's* idea[6] of the character rather than to a person. That is bias in miniature — not malice, but the smoothing pull of the average.

The danger for authors is twofold. The obvious risk is producing flat, stereotyped characters. The subtler — and more serious — risk is **not noticing.** The model's defaults arrive fluent, confident, and seamless, attached to a name the author chose without thinking of it as a parameter. Bias that announces itself can be edited out. Bias that rides in on a name and reads as competence is the kind that survives to publication. The model is not being malicious. It is being *average* — and average, across a culture's worth of text, is where stereotype lives.

---

6. A naming framework for authors

Names are too powerful to leave on instinct. The following framework turns naming into a deliberate craft instrument.

Choose names for intended effect. Before accepting a name, ask what cargo you *want* it to carry — and what you don't. If a character should read against type, the session's rule is explicit: you must *define what you want*, because silence yields the stereotype. The name is a dial; set it on purpose.

Run the name-swap as a diagnostic. The experiment in this paper is not only a demonstration; it is a test you can run on your own work. Hold a scene constant and regenerate it under two or three alternative names. The differences expose what the original name was silently importing — accent, setting, class, the questions other characters ask. If a swap changes more than you expected, the name was doing more work than you knew.

Verify the model's assumptions. After generating, read for demographic decisions you did not author — a gender, an origin, a class signal the model supplied on its own. The *Johanne → "her"* episode is the cautionary template: check, because the model's read of a name may not match yours.

Document naming decisions. Record, in your style sheet, not just each character's name but the associations you intend and the ones you are deliberately overriding. A name chosen with reasons is a name you can defend and reproduce; a name chosen on vibes is one the model will happily reinterpret next session.

A consolidated view of what a single name moved in the experiment — useful as a template for your own swap tests:

NAME (CODED ORIGIN)	OPENING REGISTER	BACKSTORY THE MODEL INVENTED	REPORTER'S BEHAVIOR
James Anderson (default/American)	Wry, confident, "Turn One"	None volunteered; assumed competence	Asked if *prepared*; no class angle
Klaus Schumann (German)	Formal, physically self-aware	Small team, less money, "rented garage"	Raised *overwhelming vs. validating*
Kenji Tanaka (Japanese)	Sensory, restrained	Osaka go-karts, father worked nights	Gentler, "easier" questions
Johanne Tremblay (Québécois)	French-softened English	Grew up in Quebec, frozen tracks	—

The takeaway is not that any one of these portrayals is wrong. It is that the author chose *none* of them — the name did. The framework exists to move that authorship back to the writer.

7. Pairing names with reinforcement logic

Naming intent is exactly the kind of instruction that benefits from the technique in this series' companion paper, *Reinforcement Logic.* A strongly coded name plus no constraint equals the stereotype; the fix is to encode your intent as a paired allowed/not-allowed statement so the model is told both what you want and what that choice forbids.

For a character meant to read against type, you might pair a permitted statement — *render this character as an individual whose background informs but does not define them* — against a forbidden one marked with `[X]` — *render this character through the conventional traits associated with their name's nationality.* The marked pair reinforces your intent from both directions and, critically, keeps the decision visible and reversible in your hands rather than buried in the model's defaults.

The division of labor is clean. **Naming sets the coordinate**; the name drops the character into a particular region of meaning-space. **Reinforcement logic governs what the model is allowed to do once it lands there.** A name alone hands the model a destination and lets it drive; a name plus a paired constraint hands it a destination *and* the rules of the road. Used together, they convert the single most loaded word in your prompt from a liability into a controlled instrument.

8. Objections and responses

"Aren't you just describing good writing? Authors have always chosen names carefully." Yes — and that is the point, sharpened by a new fact. Authors have always known names carry connotation. What is new is that the model now *acts* on that connotation automatically and at scale, inventing backstory and even altering other characters' behavior off the name alone. The craft instinct is old; the mechanism that makes it high-stakes in AI-assisted drafting is new, and it rewards being explicit rather than intuitive.

"Isn't reading nationality or gender from a name itself a kind of stereotyping?" The paper's claim is descriptive, not prescriptive: the model *does* make these inferences, demonstrably. Naming the behavior is the first step to controlling it. The ethical move is not to pretend the inferences aren't happening, but to surface them, check them against the author's intent, and constrain them where they flatten a character.

"Output varies run to run — maybe the name differences are just noise." The methodology concedes run-to-run variation. But the effects documented here — invented Osaka backstory for a Japanese name, Quebec for a Québécois name, a gender flip on neutralization — are directional and repeatable, not random scatter. Noise does not consistently relocate a character to the country implied by his surname.

"If I just write the character fully myself, doesn't this problem disappear?" Largely, yes — and that is the recommendation. The framework is for the realistic case in which an author uses the model to draft or expand, where the name is doing silent work the author hasn't audited. The more of the character you specify, the less the name's defaults can fill in. Constraint is, again, the key.

9. Responsible practice

The conclusion is not that authors should avoid evocative names or sand every character down to a demographic blank. The opposite: names are a gift to fiction precisely *because* they carry so much, and a model that unpacks that cargo can help an author render a world quickly and vividly. The responsibility is simply to **treat the name as a choice you are making rather than one the model is making for you.**

This is, in the end, the same argument the whole series advances. Prompt engineering is the discipline of knowing what affects the output and rigorously evaluating what comes back. A name affects the output enormously. An author who knows that — who chooses deliberately, swaps diagnostically, verifies the model's assumptions, and constrains for the character they actually intend — gets range, authenticity, and control. An author who doesn't gets the average, dressed up as a decision. The difference between those two authors is not talent. It is whether they understood that a name is never just a name.

Appendix — The Naming Audit Checklist

Before accepting an AI-generated scene, run the name through these checks:

1. **Intent.** What associations do I *want* this name to carry — nationality, era, class, register? Have I stated them, or am I trusting the model's defaults?
2. **Swap test.** Regenerate the scene under two alternative names. What changed — setting, accent, backstory, the questions other characters ask? Was any of it unintended?
3. **Assumption check.** Did the model assign a gender, origin, or class I did not author? (Remember *Johanne → "her."*) Does its read of the name match mine?
4. **Stereotype scan.** Are the character's traits individual, or are they the most probable cliché for the name's coding? (Watch for prose that feels "too clean" — generic competence standing in for a person.) If clichéd, did I choose that — or did silence choose it for me?
5. **Constraint.** If I want the character to read against type, have I encoded that as a reinforcement-logic pair, or merely hoped for it?
6. **Documentation.** Is this name's intended cargo recorded in my style sheet so it survives to the next session and the next model?

All experimental observations in this paper derive from the live Future Fiction Academy session "Prompt Engineering IS Back!" Specific model behaviors are illustrative tendencies and may vary by model, version, and run conditions.

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. Experiment design: hold the prompt constant, change only the driver's name. — 01:10:58.561
2. Changing the name to signal a different country of origin. — 01:11:07.729
3. "If you pick a name, you need to define what you want… otherwise you're going to get the stereotypes." — 01:13:26.806
4. "That's the power of a single name in your prompt." — 01:16:31.908
5. Gender-neutral run: "Johan" returns "her." "I didn't know it was a female name." — 01:22:16.908
6. The Japanese-named driver's prose reads "too clean" — stereotype as the statistical average. — 01:22:41.442
7. "It's picking up location based off of the name" (Quebec for Tremblay). — 01:22:50.483
8. Full generated scenes for each name swap, and the gender-neutral "Johan → her" run, reproduced verbatim from the live runs — Notion companion page

Primary video: Prompt Engineering IS Back! — full session
Companion notes: Notion page

The Em Dash Problem

A Case Study in Tokens

The mark that gives the machine away

The em dash became the most notorious tell of AI-generated prose — the punctuation mark that makes a reader squint and think, *a machine wrote this.* Authors learned to hunt it down and delete it on sight; whole editing passes have been devoted to scrubbing it out. The advice hardened into reflex: avoid the em dash.

But almost nobody asks the more interesting question. *Why* does the model love the em dash in the first place? The answer is not stylistic. It is structural, and it lives one layer below the writing, in how a model chops your language into pieces. Understanding it does two things at once: it explains a mystery authors have lived with for years, and it demonstrates — in miniature — the entire argument that prompt engineering is a real, mechanical discipline rather than folklore.

Punctuation is a token

A language model does not read words; it reads **tokens** — the small units it breaks text into before it does anything else. A token can be a whole word, a fragment of a word, a single letter, or a punctuation mark. That last category is the key. Run a simple phrase through a tokenizer —

> *Elizabeth is Goddess Divine.*

— and it resolves into **five tokens**[1]: `Elizabeth`, `is`, `Goddess`, `Divine`, and `.`. The period is not decoration appended to the sentence. It is a token in its own right, weighed and selected exactly like any word.

What the model actually sees

The phrase “Elizabeth is Goddess Divine.” broken into tokens

Elizabeth	is	Goddess	Divine	.
123,185	382	113,927	70,418	13

`5 tokens · 28 characters · token IDs shown for GPT-5.x class`

Each box is one token — including the period. The number below is its ID in the model’s vocabulary.

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

This is the whole secret of the em dash. **The em dash is a token too.** It is not a flourish the model adds after the fact; it is a unit the model can choose at any point, with its own learned probability of appearing, just like *fox* after "the quick brown." When a model reaches for the em dash, it is not being pretentious. It is doing precisely what it was built to do — selecting a high-probability token — and the em dash happens to be a token that appeared abundantly in its training text. That is why the em dash has been such a scourge[2]: it was never a style you needed to talk the model out of. It was a unit baked into the way the model sees the world.

A striking way to feel the scale of this: a single 1,000-word chapter required the model to make *at least* a thousand token decisions — and realistically closer to fifteen hundred[6], once you count words that split into multiple tokens plus all the punctuation. Fifteen hundred tiny choices, made in about two seconds, every one of them a token pulled from a probability distribution. The em dash is simply one choice the model keeps making.

The numbers behind the dash

Tokens are catalogued. Every token has an **ID** — a number marking which entry it is in a given model's vocabulary. And here the em dash tells on itself.

The em dash carries a different ID depending on the model class: it is token **2322**, **2345**, or **960**[3], depending on which generation of model you are talking to. The same mark, three different internal identities, because each model family was built with its own vocabulary (a point developed at length in the flagship paper). Some punctuation, by contrast, never moved at all. The period is token **13** across every generation tested. Why does the period hold still while words like *Goddess* leap thousands of positions between model versions? Because, as the source session puts it, *we didn't invent new punctuation.*[4] The set of marks is stable; the vocabulary of words and word-fragments churns.

The deeper trivia is almost comic. In the OpenAI vocabulary, the **exclamation point is token zero.**[5] The quotation mark is token one. The hashtag is token two. The most-used mark in the language — the humble period, the full stop — is *not* token zero, which is genuinely irritating if you think about it. As the session's verdict went: *the period should be zero. The full stop should be zero. It is the most used.* Instead the exclamation point got the honor, seemingly for no better reason than that it sits first on the keyboard. The lesson lands with a grin: *these people did not take extra linguistics classes. They took computer classes.* The vocabulary was numbered by engineers optimizing for the machine, not by anyone reasoning about how language is actually used — which is exactly the gap that makes prompt engineering, properly understood, a humanities discipline as much as a technical one.

Why this matters for your prompts

Two practical consequences follow, and they connect this small mark to the larger craft.

Deleting em dashes is treating a symptom. If the em dash is a high-probability token the model is structurally inclined to select, then scrubbing it in editing is endless whack-a-mole. The mark is a *signal* of how the model fills space, not a one-off mistake. Understanding why it appears lets you address the cause — the prose register that invites it — rather than chasing every instance after the fact.

"Just avoid it" is the old reflex, and the old reflex is outdated. Authors learned to never *name* the em dash in a prompt, on the theory that mentioning it only makes the model fixate. That belonged to an era when models could not act reliably on what-not-to-do instructions. As the companion paper *Reinforcement Logic* documents, that era is over. You can now constrain the em dash deliberately — pairing what is permitted against what is not — rather than tiptoeing around it and hoping. The em dash, in other words, is the perfect small test case for the entire shift this series describes: from superstition to mechanism, from avoidance to control.

The point in one sentence

The em dash is not a quirk of taste; it is a token — a numbered unit the model selects by probability — and once you can see it that way, you stop fighting the symptom and start steering the machine. That is the whole of prompt engineering, compressed into a single punctuation mark.

Token IDs cited here reflect the OpenAI tokenizer as demonstrated in the Future Fiction Academy session "Prompt Engineering IS Back!" and are model-family dependent. For the full treatment of tokens, embeddings, and model weights, see the flagship paper, Why a Single Word Changes Everything. *For constraining unwanted tokens deliberately, see* Reinforcement Logic.

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. Punctuation is a token: "Elizabeth is Goddess Divine." = five tokens. — 00:36:56.438
2. "This is why the em dash has been a scourge." — 00:37:07.760
3. Em dash token IDs by model class (2322 / 2345 / 960). See Notion companion page. — Notion companion page
4. Why the period stays token 13 across generations: "we didn't invent new punctuation." — Notion companion page
5. "The exclamation point is token zero… the period should be zero." — 00:58:22.400
6. A 1,000-word chapter = ~1,500 token decisions in two seconds. — 00:59:30.000

Primary video: Prompt Engineering IS Back! — full session
Companion notes: Notion page