Language Is Not a Lookup Table

On Generativity, Combinatorics, and the Poverty of Phrase-Based Authorship Detection

Flyxion

June 2026

> *"The number of sentences in any human language is not large; it is infinite."*
>
> —Noam Chomsky, *Aspects of the Theory of Syntax*, 1965

Abstract

A recurring claim in popular discourse holds that artificial intelligence can be reliably detected through the presence of certain stock phrases. This essay argues that such phrase-list detection schemes rest on a fundamental misunderstanding of how language works—both human and machine language. Drawing on generative grammar, information theory, combinatorics, and the history of linguistic cliché, I show that the ability to produce recurring surface patterns is not a signature of artificial authorship but a constitutive feature of any structured linguistic system. Language is not an archive of retrievable phrases; it is a compact generative mechanism capable of producing an effectively unbounded output space from a finite rule inventory. The claim that AI can be identified by its outputs, in isolation from the underlying process, commits a category error analogous to identifying a mathematician by the notation they use rather than the reasoning they perform. I further argue that phrase-based detection systems are self-defeating under Goodhart dynamics, and that the structural properties that genuinely characterize weak language—low information density, absent concrete referents, excessive symmetry, weak causal grounding—are irreducibly holistic and cannot be reduced to lexical surface features.

The Lookup-Table Fallacy

A certain genre of post circulates with regularity across professional social networks. It presents a list of phrases—"game changer," "let's unpack this," "move the needle," "it's not X, it's Y"—accompanied by the claim that these phrases betray AI authorship. The implied remediation is equally simple: avoid the listed phrases, and the text will read as human.

This is a theory of language. It is a bad one.

The theory holds, implicitly, that language consists of a finite collection of retrievable units—a lookup table—and that authenticity is a function of which units are drawn from the table. AI, on this view, draws from a particular subset of the table. Humans draw from a different subset. The two sets intersect in the listed phrases, which are therefore diagnostic.

Every premise of this account is false.

Human language is not stored as a lookup table. It is generated. Speakers do not retrieve complete sentences from memory; they construct sentences in real time from a compact system of rules, a generative grammar, applied to a lexicon. The number of sentences a competent speaker can produce—and understand—is not large but bounded; it is, for practical purposes, infinite. This is one of the oldest and most secure results in modern linguistics.

The phrases on viral detection lists were not invented by AI systems. They are genre conventions, marketing clichés, and rhetorical formulas that saturate the human-authored texts that AI models were trained on. "At the end of the day" predates large language models by decades. "Low-hanging fruit" is a business idiom of entirely human provenance. The rhetorical structure "It's not X, it's Y" has been used in political speech, advertising, and philosophy for centuries. If anything, these lists are catalogues of *human* excess, amplified through algorithmic reproduction, and then misattributed to the machine.

Language as Compression

To understand why phrase-list detection fails structurally, it is necessary to understand what a grammar actually is.

Consider the following observation. No living human has memorized every sentence they have ever understood. The average adult speaker of English can interpret novel sentences they have never encountered before, can judge sentences as grammatical or ungrammatical, and can produce an indefinitely large number of new utterances on demand. This is not a special cognitive achievement; it is the ordinary, unremarkable operation of a mature grammar.

The explanation is that grammar functions as a *compression scheme*. A finite set of rules—phonological, morphological, syntactic, semantic—generates an unboundedly large set of well-formed expressions. The human linguistic system stores the rules, not the outputs. From the rules, any output can be derived as needed.

This has a precise analogy in mathematics. A small set of axioms generates infinitely many theorems. A few production rules in a formal grammar generate infinitely many strings. The Fibonacci recurrence $F(n) = F(n-1) + F(n-2)$ generates the entire Fibonacci sequence from two base cases. Compression is the universal mechanism by which finite systems produce effectively infinite behavior.

Language achieves something even more powerful: it is not merely generative but *productive* in the technical sense. Speakers do not only apply known rules to known words; they extend rules to new words, coin morphological variants, construct neologisms, and interpret novel compounds on first encounter. When a speaker hears "countercontinuation" for the first time, they do not look it up in a stored vocabulary. They decompose it morphologically—*counter-* (opposing), *continuation* (a proceeding)—and infer the meaning compositionally. The grammar handles the unfamiliar through its existing machinery.

This means the output space of human language is not merely large. It is, in any meaningful sense, inexhaustible.

The Combinatorics of AI Output

The same point applies with even more force to large language models.

A common misconception treats LLM output as a kind of sophisticated retrieval: the

model searches its training data and surfaces passages that match the input. On this view, the model is fundamentally archival, and its outputs are fundamentally derivative.

The mathematics of the situation makes this view untenable.

Suppose a model has a vocabulary of 50,000 tokens. A sequence of twenty tokens has

$$50{,}000^{20} \approx 10^{96}$$

possible arrangements. This number exceeds the estimated number of atoms in the observable universe by many orders of magnitude. The set of all text ever produced by humanity—every book, every email, every webpage—constitutes a negligibly small fraction of this space. The vast majority of outputs a model produces, by elementary combinatorics, cannot be reproductions of training examples. They are novel constructions in the combinatorial space.

This is not a claim about whether models are "creative" in some deeper sense. It is a claim about what the mathematics demands. When a model produces a twenty-token sequence, it is not retrieving that sequence from memory. It is generating a token-by-token path through a high-dimensional probability distribution, a distribution shaped by training but not identical to any training example.

The phrase-list critique misunderstands this architecture entirely. It conflates the occasional production of common phrases—which is unavoidable in any system trained on human text—with the wholesale retrieval of pre-formed content. The presence of "let's unpack this" in an LLM output is not evidence that the model copied a blog post. It is evidence that the phrase is common in the distribution the model learned from, which is a fact about *human writing*.

Morphological Novelty

The generative capacity of language extends below the word level.

English has a rich morphological system. Derivational affixes—*un-*, *over-*, *-ness*, *-ify*, *-ization*—combine with existing roots to produce new words. These combinations are not arbitrary; they are constrained by the morphological grammar. But within those constraints, the space of possible formations is enormous and, more importantly,

open-ended.

Consider the following forms, none of which may have existed before this sentence:

- *unrepairability*
- *hyperadmissible*
- *overcompression*
- *countercontinuation*
- *subdistinctional*
- *reachability-preserving*

A native speaker encountering these words for the first time does not treat them as errors. They parse them immediately, using the same morphological rules applied to known forms. The grammar handles the unfamiliar through existing machinery.

Large language models have access to this same morphological machinery. They can and do produce novel compounds, novel derivations, and novel technical terms when context demands specificity. A model with tool access can go further still: it can call random generators, synthesize names algorithmically, produce UUIDs, generate lexical items from hash functions, construct entirely new controlled vocabularies. Once tool use enters the picture, there is no principled upper bound on the lexical novelty available to a machine system.

Even a first-order Markov chain—one of the simplest possible text-generating systems, operating over an n-gram vocabulary derived from a small training corpus—produces an astronomically large number of outputs that have never previously existed. Most are incoherent, but they are combinatorially novel. The claim that AI systems merely repeat phrases fails even at the level of the most primitive probabilistic generator.

Genre, Convention, and the Ubiquity of Cliché

There is a deeper problem with phrase-list detection that has nothing to do with AI at all.

Every genre of human writing is defined partly by its conventions, including its stock phrases. Scientific writing recycles "the results suggest," "consistent with prior work," "further investigation is warranted." Legal writing recycles "notwithstanding," "hereinafter referred to as," "in witness whereof." Journalistic writing recycles "sources say,"

"officials declined to comment," "the situation remains fluid." Academic philosophy recycles "it might be objected that," "on one reading," "this view faces a serious challenge."

None of these phrases is treated as evidence of AI authorship, because they are recognized as genre conventions. The genre is defined in part by its conventions. A scientific paper that avoided all conventional phrasing would be harder to read, not more human.

The phrases on AI detection lists are not different in kind. They are the conventions of a particular genre: the business blog post, the LinkedIn thought-leadership piece, the startup newsletter. That genre was saturated with these phrases long before large language models existed. AI systems, trained on vast corpora of human text, reproduced the genre conventions of the genres they were trained on. The result is that AI-generated business writing sounds like human business writing, because that is what it was trained to approximate.

The detection problem is therefore not "does this text contain AI phrases?" but "does this text belong to a genre defined by these conventions?" And the answer to that question tells us something about genre membership, not about authorship.

Goodhart Dynamics and the Self-Destruction of Detection

Suppose, counterfactually, that the phrase-list approach had some initial validity. Suppose that at some moment in 2023, the listed phrases did appear more frequently in AI output than in human output. What follows?

The answer is a classic instance of Goodhart's Law: when a measure becomes a target, it ceases to be a good measure.

The progression is predictable:

1. Observers identify a weak correlation between phrase use and AI authorship.
2. The correlation is publicized and treated as a robust signal.
3. Human writers begin avoiding the listed phrases.
4. AI training data is updated with the new human corpus.
5. AI models learn to avoid the listed phrases.
6. The correlation vanishes.
7. New lists are compiled to restore the illusion of detection.

This cycle has no stable fixed point. Every successful detector destroys its own predictive power by changing the behavior of writers who want to avoid detection. The detector becomes a stylistic prescription, not a diagnostic instrument. And stylistic prescriptions don't identify humans; they identify people who have read the prescription.

The same failure mode appears throughout social systems wherever surface metrics are substituted for underlying constructs. Educational testing, search engine optimization, hiring assessments, social media engagement metrics—in each case, optimizing against the metric degrades the metric's validity. Language authorship detection is not special; it is subject to the same dynamics.

What Actually Distinguishes Weak Writing

If phrase-list detection fails, what would a structurally sound account of weak AI writing look like?

The properties that actually characterize uninspired machine output are holistic and difficult to operationalize as word-level tests. They include:

Low information density. A text can be grammatically fluent and generically plausible while containing very few specific claims. Compare:

> *"The future of AI is transformative. Organizations must adapt to harness its potential and unlock new value across the value chain."*

> *"The model reduced reconstruction error by 23% on held-out data. Most of the gain came from replacing the fixed embedding layer with a learned geometric projection."*

The first passage could be deleted without any loss of information, because it contains none. The second could not be deleted without losing specific claims that are either true or false.

Absent concrete referents. Weak text refers to generic categories—"organizations," "stakeholders," "the ecosystem"—rather than named particulars. It makes claims at the level of abstractions that cannot be falsified because they are not anchored to anything specific.

Excessive structural symmetry. Human prose is rhythmically irregular. It varies sentence length, clause depth, and transition frequency in ways shaped by genuine

argumentative structure. Texts that are too regular—too uniformly balanced, too consistently signposted—often betray the absence of the underlying argumentative heterogeneity that drives natural variation.

Weak causal grounding. Human expertise is reflected in the ability to give mechanistic accounts of why things happen. "Sales declined" is a datum. "Sales declined because the reorder trigger was miscalibrated for longer supply chains" is an explanation. Weak writing produces many claims of the first type and few of the second.

Absence of particular experience. A person who has worked in construction, recycled industrial waste, or painted large murals at speed writes differently about physical processes than a system that has only processed text descriptions of such activities. The difference is not in any list of phrases but in the presence or absence of specific phenomenological texture.

None of these properties can be reduced to a list of banned phrases. They are structural. They require reading a text as a whole, attending to what it does and does not do, rather than running a surface search for forbidden lexical items.

Humanity as Process, Not Output

The deepest error in phrase-list detection is philosophical.

It assumes that humanity can be identified by outputs: by the particular tokens a speaker produces. If the tokens overlap with certain patterns, the output is AI. If they avoid the patterns, the output is human. This is a view of authorship as pattern matching.

But authorship is not pattern matching. It is participation in a *process*: a process of forming intentions, deploying acquired knowledge, responding to context, exercising judgment, drawing on experience, and producing language that is answerable to a world. The outputs of this process are not its identity. Two processes can produce identical outputs while being fundamentally different in nature. And two processes can produce different outputs while being fundamentally similar.

Every human speaker uses a grammar inherited from previous generations. The grammatical rules current English speakers apply were not invented by any living person; they evolved over centuries, transmitted through communities of speakers,

and internalized during language acquisition. In this sense, every human speaker uses machinery they did not invent, to produce outputs that are massively shared with other speakers. The presence of shared patterns—clichés, idioms, stock phrases, genre conventions—is not evidence against human authorship. It is evidence that the human speaker is embedded in a linguistic community, as all human speakers are.

AI systems participate in a different implementation of generativity. They were trained on human-produced text, and their outputs reflect the statistical structure of that text. This is a real difference, and it is worth understanding. But the existence of shared surface patterns between human and AI output is not where the difference lies. The difference lies in the underlying process: what the system knows, what it has experienced, what it is answerable to, and how it relates to the world it is describing.

Phrase lists detect none of this. They detect surface tokens. And surface tokens are not where the interesting problem is.

Source Code and Compiled Artifacts

The failure of phrase-based authorship detection is not merely a failure of linguistic analysis. It is a manifestation of a more general confusion between generators and generated artifacts. The same confusion appears in software engineering, scientific modeling, formal specification, and the design of AI workflows themselves. Recognizing it in the domain of language makes it easier to recognize everywhere else.

The misuse of AI in writing workflows runs parallel to a misuse of AI more generally: asking a generative system for outputs at the wrong layer of abstraction.

In software engineering, this error has a canonical name. Nobody wants to edit machine code. They want source code. Source code is valuable not because it looks better or reads more clearly than compiled binaries, but because it *remains transformable*. It preserves the ability to intervene. It sits upstream of all the things it can become.

The same distinction applies, with surprising precision, to language.

A LaTeX document is source code for a paper. A Markdown file is source code for a website. A musical score is source code for a performance. A screenplay is source code for a film. Each of these is editable, diffable, parameterizable, and composable. Each can be compiled into multiple downstream forms—PDF, EPUB, audiobook, podcast,

slideshow, infographic, translation—without discarding the upstream representation.

A rendered audio podcast, by contrast, is close to machine code. Once the voice is synthesized, the music mixed, the timing locked, and the file exported, the cost of modification is high. Changing one sentence may require regenerating an entire section. The compiled artifact contains the output but not the transformation that produced it.

This gives us a layered picture:

$$\text{raw ideas} \longrightarrow \text{structured source} \longrightarrow \text{compiled outputs}$$

The source layer is where the highest-value work occurs. It is where distinctions are preserved, where repair is cheap, where future interventions remain admissible. The output layer is where you communicate with audiences, but it is not where you should be accumulating your primary artifacts.

Most popular AI workflows invert this hierarchy. They ask the model for the final blog post, the final video, the final podcast episode. The result is a compiled artifact with no recoverable source. When the output is wrong, or needs updating, or should be repurposed for a different format, the user must start over. They have discarded most of their future freedom to modify, repair, and extend the result. It is analogous to shipping binaries while deleting the repository.

There is an information-theoretic way to state the same point. A LaTeX source file preserves a much larger space of admissible futures than any particular compilation. The PDF is one point in the space. The audiobook, the website, the slide deck, the summary in another language are other points reachable from the same source. Each compilation resolves certain choices while leaving others open. By contrast, a rendered audio file has resolved nearly every choice: speaker, pacing, intonation, background, duration, format. Recovering any of those degrees of freedom from the compiled artifact is expensive or impossible.

In this sense source formats are not merely convenient. They are *repair-friendly representations*. They preserve the capacity to intervene in future states. This connects, perhaps unexpectedly, to the repair-theoretic framework developed elsewhere in this program: a system persists when useful distinctions remain recoverable. Source code is what makes distinctions recoverable across time.

A principle follows:

> *The value of a representation is proportional not only to what it expresses, but to the space of future interventions it preserves.*

A LaTeX file is valuable because it does not merely describe a paper. It preserves a large admissible region of possible future papers. The PDF is only one point in that region.

Operators, Instruments, and Compressed Agency

Once we understand the source-versus-compiled distinction, a further clarification about how AI tools are best used follows naturally.

The most productive use of a language model is not to generate finished language. It is to generate *reusable transformations*: scripts, macros, templates, pipelines, grammars, and source files that preserve and extend the user's future agency.

This is the insight embedded in the design of the Unix command line, the Vim editor, and the LaTeX typesetting system. These are not environments for entering text character by character. They are environments for specifying *operators over text*. A single Vim command can act on an entire corpus:

```
:%s/Let's unpack this/We can analyze this/g
```

This is not manual editing. It is a compact formal grammar for an intervention: select the entire buffer, match a pattern, substitute, preserve everything else. One line of text acts over documents of arbitrary length.

A Bash pipeline generalizes the same structure across files:

```
grep -Rl "game changer" . | xargs sed -i 's/game changer/structural shift/g'
```

An AutoHotkey macro turns a short trigger into an arbitrary expansion—a recurring phrase, a citation block, a theorem environment, a command sequence—making the keyboard itself a generative system. A Python script traverses a directory of source files and applies a transformation uniformly. In each case, a finite specification produces an action over an indefinitely large domain.

This is exactly the generative structure described in the earlier sections. Small rules, large output space. The Vim command is a grammar. The Bash pipeline is a grammar.

The Python script is a grammar. The human operating these tools is not entering outputs; they are specifying generators.

The connection to Chomskyan linguistics is more than metaphorical. When a speaker uses a morphological rule to produce *unrepairability* on first encounter, or when a programmer writes a regular expression to match a class of strings, both are applying a finite rule to an open-ended domain. The rule is more powerful than any of its instantiations. It is more general, more reusable, more compressible, and more amendable to revision.

A language model becomes a genuine cognitive tool when it operates at this level. When it produces:

```
from pathlib import Path

for path in Path(".").rglob("*.tex"):
    text = path.read_text(encoding="utf-8")
    text = text.replace("At the end of the day", "Ultimately")
    path.write_text(text, encoding="utf-8")
```

it is not decorating a paragraph. It is manufacturing an instrument. The instrument can be inspected, modified, saved, rerun, composed with other instruments, and applied to future documents that do not yet exist. It is a durable piece of compressed agency.

The same applies when a model helps write documentation, generates a LaTeX macro that automates a recurring typographic construction, produces a shell script to normalize transcript formatting, or reads a library's API documentation and returns a function that wraps it correctly. These outputs have the properties of source code: they are editable, inspectable, recomposable, and upstream of everything downstream.

The practical reframe this suggests is direct: use AI less as a producer of finished language and more as a generator of reusable operators. Ask not for the essay but for the LaTeX template. Ask not for the summary but for the script that generates summaries from a parameterized source. Ask not for the podcast but for the structured source from which the podcast, the transcript, the PDF, and the website can all be compiled.

The goal is not to have a tool that produces outputs. It is to have a set of instruments that increase the future editability of everything else.

Abstraction as Substrate Independence

There is a principle running through all the cases examined so far—grammar, morphology, Bash pipelines, LaTeX source, type signatures, BNF grammars—that deserves to be stated explicitly. In each case, the more abstract object is not the more complicated one. It is the one that is less dependent on a particular substrate.

This reframes abstraction entirely. The usual picture treats abstraction as elevation: moving up a ladder, becoming more remote from concrete things. But the more precise account is independence. An abstract representation is one whose meaning survives the replacement of its implementation.

Machine code is the least abstract level of a program. Every instruction is bound to a specific processor architecture. Change the hardware and the code may not run, or may not mean the same thing. Nothing in the bytecode is portable; the representation is maximally committed to a particular substrate.

Assembly introduces symbolic names for opcodes, creating a thin layer of independence. The same assembly may target multiple chips through an assembler.

A C program describes operations without specifying register allocation, instruction scheduling, or memory layout. Many different machines can compile it.

A Python program goes further: it does not specify compilation strategy, memory management, or execution model. The same source runs on CPython, PyPy, and several other runtimes.

A type signature is often more abstract still. It describes only what a function accepts and what it returns, leaving every implementation detail unspecified. Infinitely many programs satisfy the same type signature.

A formal specification or constraint system reaches the highest level: it describes what properties a valid computation must have, while leaving the how entirely open. The specification defines a space of admissible implementations rather than selecting any particular one.

The resulting hierarchy runs roughly:

Level	What is fixed / What remains open
Machine code	Architecture, instruction set, memory addresses
Assembly	Opcodes abstracted; layout still specific
Systems language	Operations abstracted; memory model specified
High-level language	Memory model abstracted; execution model specified
Abstract syntax tree	Syntax abstracted; structural semantics preserved
Type system	Implementation abstracted; interface specified
Formal specification	Interface abstracted; constraints specified
Constraint system	Constraints remain; all realizations admissible

At each step, more implementation decisions are released while more admissible implementations become possible. The abstract syntax tree is a good illustration: it discards formatting, spacing, comments, and most syntactic choices, while preserving the computational structure. Many distinct programs share the same AST. The AST is therefore more abstract than the source code, even though the source code is what the programmer actually writes.

The same relationship holds between a UML diagram and the code that implements it, between a recipe and a meal, between a circuit diagram and a physical circuit, between a theorem statement and its proof, between a type signature and a function body.

> *A circuit diagram is not a circuit. A BNF grammar is not a sentence. A type signature is not an implementation. A theorem is not a proof. Each specifies a space of admissible realizations rather than occupying a single point within that space.*

Human Language as an Ecosystem of Domain-Specific Languages

Natural language fits this picture in an illuminating way. English is not a single language. It is a vast collection of partially overlapping domain-specific languages, each with its own grammar, vocabulary, constraints, and admissible transformations.

Legal English. Mathematical English. Clinical English. Musical English. Construction-site English. Naval English. Financial English. Liturgical English.

A physicist writing "let $\mathcal{H}$ be a Hilbert space" is operating within a grammatical register that would be opaque to most speakers despite using common English words. A contracts lawyer using "notwithstanding the foregoing" is invoking a deontic grammar with precise semantics that differ substantially from their surface sense in ordinary speech. A machinist using "tolerances," "runout," and "chatter" is working in a DSL that shares lexical items with general English but assigns them highly constrained technical meanings.

Viewed this way, human language is an ecosystem of domain-specific languages compiled into a common surface representation. The surface form—English text—is the rendering. The underlying DSL grammar is the source. Two sentences may look similar on the surface while belonging to entirely different generative systems, just as two programs may look similar in syntax while computing entirely different things.

This also means that fluency in "English" is not a single competence. It is a bundle of partially overlapping competences, each tied to a particular domain grammar. A speaker fluent in everyday English may be entirely unable to interpret legal English, mathematical English, or the English of a nineteenth-century shipwright's log—not because the words are different but because the DSL grammar is different.

Admissibility as the Defining Property of Abstraction

The unifying principle across all these cases is admissibility. The more abstract object defines an admissible region; the more concrete object occupies a point within it.

A grammar defines a language: the set of admissible sentences.

A type system defines a program space: the set of well-typed programs.

A formal specification defines an implementation space: the set of correct programs.

An admissibility condition defines a continuation space: the set of acceptable next states.

A LaTeX source file defines a document space: the set of papers reachable from that source by future editing.

In every case, the abstract object is more powerful not because it is more complex but because it organizes more while committing to less. The constraint is the power. A grammar that ruled out fewer strings would be less useful as a grammar, not more

useful. A type system that accepted fewer programs would not be more abstract; it would be more restrictive. The generative power comes from being specific enough to organize while being permissive enough to leave the future open.

This gives a precise way to state what phrase-list AI detection gets wrong. It attempts to identify authorship by pointing to specific points in an output space—particular phrases—while ignoring the generative structure that defines the space. It mistakes a realization for the specification. It reads the machine code while discarding the source.

The interesting object is always the generator. And the measure of a generator's value is the admissible region it defines: how much structure it provides, and how many futures it leaves open.

A Fundamental Underdetermination

The abstraction hierarchy points toward a consequence that can be stated with some precision.

Authorship cannot be reliably inferred from surface realizations whenever multiple generative processes map to the same realization, and a single generative process admits multiple realizations.

Both conditions hold in any sufficiently rich linguistic system. A human can intentionally write in a style that resembles AI-generated text. An AI can write in the style of a nineteenth-century essayist. A human can avoid every phrase on a detection list. An AI can avoid every phrase on the same list. The mapping between generative process and surface realization is many-to-many in both directions.

Once that is true, phrase-based authorship detection is not merely inaccurate. It is *structurally underdetermined*. No list of surface patterns can, even in principle, uniquely identify the process that generated them. The surface notation is not the AST. The AST is not the type signature. The type signature is not the specification. At every level of the hierarchy, multiple realizations correspond to the same underlying structure, and the same structure can generate multiple realizations.

Representation and Realization

There is a further dimension to the abstraction argument that the software hierarchy does not fully capture. The same underlying structure can have multiple representations, and each representation can have multiple realizations, so that the chain from abstract object to physical manifestation passes through many intermediate stages.

Consider a theorem. The theorem is not its proof. A single theorem may have multiple proofs, each with different structure, length, and technique. The proof is not its LaTeX source. Many different source files could typeset the same proof. The LaTeX source is not its PDF. The same source compiled by different engines or on different systems produces files that differ at the byte level while remaining perceptually identical. The PDF is not its printed copy. The printed copy is not the photons entering a reader's eye. The photons are not the interpretation in a reader's mind.

$$\text{theorem} \rightarrow \text{proof} \rightarrow \text{source} \rightarrow \text{PDF} \rightarrow \text{print} \rightarrow \text{photons} \rightarrow \text{interpretation}$$

At every arrow, a realization process selects one instance from an admissible set. The theorem constrains but does not determine the proof. The proof constrains but does not determine the typesetting. The typesetting constrains but does not determine the rendering. Each stage preserves the essential structure while adding and discarding details appropriate to its substrate.

The same chain applies to any linguistic artifact. A sentence is not the sequence of characters that encodes it. The character sequence is not the audio waveform that pronounces it. The waveform is not the neural firing patterns that interpret it. At each stage, the representation changes substrate while preserving, imperfectly and partially, the structure that matters.

This has an immediate bearing on the authorship detection problem. Phrases live at the level of surface character sequences—the typographic layer. But the generative process that produces text operates at the level of semantic and syntactic structure, at the AST layer and above. The same generative intention can produce many different surface phrasings. Different generative intentions can converge on the same surface phrasing by coincidence, convention, or imitation.

Detecting authorship from surface phrases is therefore detecting at the wrong layer.

It is like attempting to identify a composer from the ink chemistry of the printed score rather than from the musical structure the score encodes. The ink chemistry is real and observable, but it bears only an indirect and contingent relationship to the compositional process.

Conclusion: The Generator and Its Products

A simple Markov chain trained on fifty pages of text can produce sentences that have never existed before. A morphological rule applied to a known root can produce a word no one has ever used. A frontier language model operating over a fifty-thousand-token vocabulary can produce an output from a space so large that no enumeration is possible. Human speakers, using a grammar they did not invent, produce sentences that have never been spoken and will never be spoken again.

All of these systems are generators. They are not lookup tables. They are not archives. They are compact rule systems capable of producing effectively unbounded output from finite machinery.

Phrase-list detection schemes confuse the products of generators with the generators themselves. They treat language as a bag of retrievable items rather than as a machine for constructing items. They detect at the typographic layer while the generative process operates at the structural layer. They mistake a realization for the specification, and the ink for the music.

The same confusion appears in popular AI workflows that ask for compiled artifacts rather than source representations, and in theories of language that treat the lexicon as an archive rather than as one component of a productive rule system. In each case the error is the same: privileging the output over the generator, the instance over the grammar, the PDF over the LaTeX.

The distinction between a generator and its products appears throughout human knowledge. Grammars generate sentences. Specifications generate implementations. Theories generate predictions. Constitutions generate laws. Source code generates programs. The enduring value of a representation lies not merely in what it currently says, but in the space of futures it keeps available. A civilization that preserves only outputs eventually loses the ability to repair them. A civilization that preserves generators retains the capacity to produce new outputs indefinitely.

The interesting question is therefore not which phrases a text contains. It is what generative structure made those phrases possible—and whether that structure, and the admissible region it defines, has been preserved.
