

The Hitchhiker's Guide to Monoculture

Gordon Burtch – Boston University

gburtch@bu.edu

Abstract. Large language models (LLMs) often produce homogeneous outputs, raising concerns that AI coding assistants may lead to convergence in the software artifacts that developers create. Whether this occurs in practice is unclear because developers interactively prompt, evaluate, modify, and reject model outputs, and because outputs vary with prompt and repository context. I examine code homogenization using Kaggle contest submissions from 2019 to mid-2026. I first document widespread convergence toward the random seed value 42, consistent with LLMs reinforcing a longstanding convention in programming culture. I then study homogenization more broadly, at two levels of aggregation and abstraction. At the submission level, I measure the average pairwise similarity of submissions within contests. At the contest level, I measure the conceptual span of submitted code, motivating distinct measures for each: TF-IDF representations, which capture surface syntax, and Voyage 3 code embeddings, which capture code intent and semantics. The results demonstrate substantial syntactic homogenization at both the individual and collective levels: individual submissions have become more alike in literal syntax and code structure, while the latent dimensionality of syntactic variation has narrowed. In contrast, I find little evidence of semantic homogenization, individually and collectively. Average semantic distance remains essentially flat, and the contest-level latent dimensional span of semantic approaches remains stable, with evidence suggesting it has even expanded modestly. These findings suggest that AI coding assistants are certainly standardizing implementation details, yet they have not yet produced evidence of homogenization in the approaches and problem-solving strategies coders employ.

Keywords: Digital Monoculture, LLMs, AI coding assistants, Generative AI, Software Programming

Draft: July 12th, 2026

Introduction

For decades, scholars have argued that reliance on shared technological components can create systemic fragility, akin to the risks associated with monoculture, or lack of biodiversity, in agricultural contexts. Since the 1980s, these risks have been discussed from multiple perspectives. First, shared reliance on technological artifacts can increase exposure to single points of failure, a phenomenon known as software, IT, or, more generally, technological monoculture (Stamp 2004). Recent outages at Amazon Web Services and CrowdStrike provide notable examples: when services that form part of the backbone of the modern digital economy fail, the broader digital ecosystem can grind to a halt. Second, the use of shared algorithms in decision-support systems can increase homogeneity in human decision-making, a phenomenon known as algorithmic monoculture (Kleinberg and Raghavan 2021; Messeri and Crockett 2024). The canonical example concerns shared resume-screening tools across employers, which may lead recruiters to screen in, or screen out, increasingly similar job candidates. The emergence of large language models (LLMs) has recently introduced a third, related concern: generative monoculture, or the tendency of LLMs to produce homogeneous outputs across models, prompts, or repeated samples (Wu et al. 2025; Jiang et al. 2026).

These three notions of monoculture span different levels of abstraction. Generative monoculture concerns convergence in model outputs. Technological monoculture concerns convergence in the artifacts humans create and rely upon, including software implementations, libraries, defaults, and other concrete design choices. Algorithmic monoculture concerns the convergence of the underlying ideas, strategies, or conceptual approaches that humans employ in problem-solving and decision-making. AI coding assistants plausibly link all three: homogeneous model outputs may encourage similar code implementations, which may in turn encourage similar problem-solving approaches. Yet these links are neither mechanical nor guaranteed. Evidence that LLMs produce homogeneous outputs does not, by itself, imply that human work products will homogenize. In most real production settings, LLMs do not operate autonomously; they are used collaboratively. Users prompt them, evaluate their suggestions, reject or revise their outputs, combine them with prior knowledge, and integrate selected pieces into broader workflows. Whether generative monoculture survives this human production process and appears in the artifacts people ultimately create is therefore an empirical question.

Understanding whether such homogenization occurs in code is important because software convergence carries both benefits and risks. Standardized implementations may be easier to read, maintain, and debug. But repeated reliance on common defaults and implementation patterns can also create systemic fragility, including unexpected forms of correlated exposure. The choice of random seed values provides a useful example, and one I examine empirically below. Random seeds are used to support replicability in machine-learning and statistical software by governing stochastic elements such as initialization, resampling, and data splitting. Developers typically treat seed values as incidental, and in many contexts they are. However, recent work shows that seed choice can have important consequences in some settings, particularly for the quality and stability of training and fine-tuning large language and computer vision models (Picard 2021; Zhi et al. 2023; Bui et al. 2025; Li et al. 2025). If AI coding assistants coordinate developers around the same arbitrary seed value, then a seemingly minor implementation default may become an ecosystem-level source of correlated behavior.

At the same time, AI-induced homogenization in code is not inevitable. Developers retain agency: they may accept, reject, edit, combine, or iteratively refine LLM-generated code. Moreover, coding-assistant outputs are homogeneous only conditional on the prompt and conversational context. To the extent that developers prompt heterogeneously, use different models or settings, and adapt their prompts in response to feedback, monoculture may fail to arise. Recent work on prompt adaptation reinforces this point: users’ ability to adapt their interactions with generative AI systems can act as a dynamic complement to model capability, especially in tasks with clear objectives and feedback (Jahani et al. 2026). With the right expertise, incentives, and feedback structure, LLMs may even foster greater diversity by lowering the cost of experimentation and allowing users to explore approaches they would otherwise not have attempted.

This raises the empirical questions I address in this paper: To what extent do LLMs, particularly AI coding assistants, homogenize human-produced software code? Does homogenization appear primarily in surface implementation details, or does it extend to higher-level algorithmic approaches? Put differently, I examine whether generative monoculture propagates into technological monoculture, and whether technological monoculture in turn propagates into algorithmic monoculture.

Kaggle contests provide a useful setting for studying these questions because they define a narrow and shared task environment. In broad creative domains, such as art, writing, or design, the relevant task space is weakly bounded. If AI-assisted artifacts appear to occupy a larger region of concept space, this may reflect genuine exploration, but it may also reflect the importation of already-existing concepts, motifs, or styles from the model’s external training distribution into the focal dataset. Conversely, if artifacts become more similar on average, it may be difficult to distinguish convergence in ideas from changes in style, user composition, prompt conventions, or platform incentives. Kaggle competitions mitigate these concerns. Within a contest, participants work with the same data, pursue the same objective, observe the same leaderboard, and are evaluated by the same scoring rule. Within-contest similarity is therefore a meaningful object of study: submissions differ not because contestants are solving unrelated problems, but because they pursue different approaches to the same problem.

I evaluate my questions using large-scale data on Kaggle contest submissions between 2019 and mid-2026. I begin with a concrete case study of convergence in an implementation default: the choice of random seed values. I show broad convergence in developers’ seed choices toward the value 42, a long-standing meme in hacker culture derived from *The Hitchhiker’s Guide to the Galaxy*. The share of seeded Kaggle contest submissions employing 42 began to rise sharply in late 2022, and by mid-2026, more than 95% of seeded submissions use this value. In a broad sample of machine-learning repositories from GitHub over the same period, I observe a similar pattern: more than 70% of seeded repository files now use 42. These patterns are consistent with AI coding assistants reinforcing and amplifying a preexisting default, transforming a common convention into a dominant software practice.

I then examine monoculture at different levels of abstraction and aggregation. Regarding the former, I construct two representations of each Kaggle submission: a TF-IDF n-gram representation, which captures surface-level syntax, and a semantic-retrieval-focused Voyage code embedding, which captures more abstract code intent or semantics. Appendix D provides validation evidence that these two embedding approaches reliably capture the intended aspects of code. Using these representations, I construct two within-contest measures at different levels of

aggregation. First, I compute the average pairwise distance between submissions to the same contest, separately in the TF-IDF and Voyage embedding spaces. This measure captures whether the typical submission is becoming more similar to its contemporaneous peers. Second, I compute contest-level effective spectral rank, a measure based on spectral entropy.¹ This measure captures whether submissions collectively span many independent directions in embedding space, or whether variation has collapsed onto a smaller number of dominant axes.

The results suggest that homogenization is occurring, but only in certain respects. Following the rise of AI coding assistants, average pairwise novelty among Kaggle submissions has declined in literal syntax, yet evidence of such homogenization is generally lacking at higher levels of abstraction, i.e., semantics. Similarly, the contest-level latent dimensional span of code submissions has declined, but again only in syntax. When it comes to semantics, conceptual span remains essentially unchanged.² Thus, while AI-era code production exhibits clear evidence of homogenization, that homogenization is most apparent in implementation details and surface structure, not in problem-solving approach.

These findings offer a counterpoint to the growing literature documenting homogenization in LLM outputs and in AI-assisted human writing, ideation, creativity, and work product. Much of this literature implicitly treats similarity across outputs, artifacts, and decisions as jointly determined. The results here suggest that these are distinct objects of study. Homogeneity in model outputs can propagate into human-created artifacts without producing commensurate convergence in the underlying solution space. In software, the use of similar code structures, libraries, boilerplate, or defaults does not necessarily imply similar modeling strategies; conversely, different surface implementations may still embody similar algorithmic approaches.

The broader theoretical point is that LLM-induced homogenization is hierarchical. AI coding assistance may coordinate developers around common defaults, templates, libraries, and implementation patterns while having relatively little influence on the strategies they employ in problem-solving. Conversely, conceptual convergence may increase even when surface-level implementations appear diverse. Distinguishing these layers is therefore necessary for understanding the ecosystem-level consequences of generative AI. The relevant question is not simply whether AI makes work products more similar, but where and how that similarity arises. This perspective also helps reconcile two competing views of AI's influence: that it pulls users toward average, conventional output, and that it enables experimentation by lowering the cost of execution. Both may be true simultaneously, but at different levels of abstraction.

The findings also have practical implications for the governance of AI coding assistants. Existing discussions often focus on whether generated code is correct, secure, maintainable, or legally usable. Those concerns are important, but they treat each generated suggestion largely in isolation. The results here point to a broader ecosystem-level issue: even when individual AI-

¹ I elaborate on this measure in the methods section. Briefly, however, the effective spectral rank is closely related to principal components and eigenvalues. Intuitively, the measure reflects the number of principal components one must retain to explain most of the variation in the set of embeddings. If this measure declines, the embeddings can be effectively represented by a smaller number of latent dimensions.

² In Appendix D, I demonstrate that this null result is neither an artifact of the measures being insensitive, nor of contests themselves being so narrow as to prevent contraction (or a rise) in the span of solution approaches.

generated suggestions are harmless on their own, repeated adoption across many developers can create new forms of dependence on common defaults and implementation patterns. The seed-42 result illustrates this externality. A single seed choice may appear innocuous, but widespread convergence on the same value can reduce the variation that random seeds are meant to support in initialization, resampling, splitting, and replication. Similar dynamics may arise for package imports, hyperparameter grids, evaluation routines, testing patterns, and other apparently minor implementation choices.

At the same time, the results suggest that AI-induced monoculture depends on context. Kaggle contests combine several forces that should promote homogenization: a common task, shared data, public examples, discussion forums, and visible performance rankings. Yet they also provide strong incentives for differentiation and rapid feedback about performance. The evidence suggests that these incentives may preserve conceptual diversity even as implementation details become more standardized. A general implication is therefore that AI coding assistance is likely to homogenize software unevenly across settings. In routine production environments, internal tools, or low-incentive coding tasks, conceptual convergence may be more pronounced. In competitive or high-feedback environments, developers may remain active participants in the production process, using AI to accelerate implementation without fully surrendering the exploration of alternative solution strategies.

Related Literature

A rapidly growing body of literature examines the effects of large language models on work. Early studies emphasize three broad themes: exposure, productivity, and task reallocation. LLMs are potentially relevant to a large share of knowledge-work tasks (Eloundou et al. 2024). Consistent with this view, experimental and field evidence indicate that generative AI can, on average, improve worker productivity across a range of professional tasks, though effects vary substantially across tasks, workers, and organizational contexts (Noy and Zhang 2023; Brynjolfsson et al. 2025; Dell’Acqua et al. 2026).

Software development is a particularly important setting for understanding these effects. Programming is one of the domains in which LLMs have been most heavily deployed, through tools such as GitHub Copilot, Codex, Cursor, Claude Code, and related code-generation systems. These tools help developers generate code snippets, functions, tests, and documentation, and can facilitate debugging. However, developers ultimately decide whether and how to incorporate generated code into functioning software systems. The effect of coding assistants, therefore, depends not only on the quality of generated code but also on how developers prompt, evaluate, modify, and integrate that code.

Early empirical evidence suggests that AI coding assistants can substantially affect software-development productivity. Peng et al. (2023), in a controlled experiment with GitHub Copilot, find that developers given access to Copilot complete a programming task substantially faster than those in the control condition. Subsequent field-experimental evidence from firms similarly suggests that access to Copilot can improve measured developer productivity, though effects vary across contexts and workers (Cui et al. 2026). These findings are consistent with the view that LLMs

reduce the cost of producing code, especially for well-specified programming tasks where the relevant solution space is familiar and generated output can be evaluated relatively quickly.

Other work emphasizes that AI coding assistants change the nature of software work, not merely the speed with which code is produced. Hoffmann et al. (2025) find that Copilot access shifts open-source developers’ task allocation toward core coding work and away from non-core project-management activities. At the same time, usability and practitioner studies show that developers must still understand, edit, debug, and integrate generated code, making AI assistance dependent on human evaluation and project-specific judgment (Vaithilingam et al. 2022; Zhang et al. 2023).

Work has also begun to examine how AI coding assistants affect innovation in open-source communities. Yeverechyahu et al. (2026) study the launch of GitHub Copilot, exploiting Copilot’s selective support across programming languages. They find that Copilot increased overall open-source contributions, but that these gains were concentrated more strongly in maintenance-related and iterative contributions than in code development or capability-expanding contributions. They interpret this pattern as evidence that LLMs are especially useful for interpolative, “inside-the-box” work, where developers build on an existing codebase and the relevant context is well defined, but less useful for extrapolative, “out-of-the-box” innovation. This distinction is directly relevant to the present paper because it suggests that LLMs may alter not only how much software is produced, but also the kinds of solutions software communities pursue.

A smaller but growing body of literature points to the limitations and risks of AI coding assistants. Pearce et al. (2025) show that GitHub Copilot often generates insecure code across a range of security-relevant programming tasks. Similarly, Fu et al. (2025) provide evidence from GitHub projects that AI-generated snippets frequently contain security weaknesses. Vaithilingam et al. (2022) find that although programmers often value Copilot as a useful starting point, they can struggle to understand, edit, and debug its generated code. Zhang et al. (2023) similarly document integration difficulties as a central practical challenge in using Copilot. Collectively, these findings suggest that generated code is not a costless input into software production. Developers must evaluate whether it is correct, secure, maintainable, and consistent with project-specific requirements.

Recent evidence also complicates the simple claim that AI coding assistants reliably increase developer productivity. Becker et al. (2025), in a randomized controlled trial with experienced open-source developers working on mature repositories, find that access to AI coding tools slows or extends completion time rather than reducing it. They attribute this pattern, in part, to the costs of prompting, waiting, reviewing, and correcting AI output. This result is not inconsistent with earlier evidence of productivity gains in more bounded programming tasks. Rather, it suggests that the returns to AI coding assistance depend on task complexity, developer expertise, codebase maturity, and the costs and incentives associated with evaluating generated output.

LLMs may also be changing the broader knowledge infrastructure surrounding software development. Burtch et al. (2024) study the growth of ChatGPT and document substantial declines in Stack Overflow traffic and question posting, with the largest declines concentrated in programming topics where ChatGPT is likely to perform well. They also show that similar declines are not observed in matched Reddit developer communities, suggesting that the effects of LLMs depend on the social and institutional structure of the knowledge community. This evidence is relevant because Stack Overflow has historically served as a diverse, distributed source of

programming knowledge, examples, and debugging advice. If developers increasingly substitute away from such communities toward a smaller number of general-purpose LLMs, the sources of programming guidance may themselves become more centralized.

Taken together, prior literature provides clear evidence that LLMs are changing software development. However, this work has focused primarily on individual productivity, task allocation, developer experience, generated code quality, security risks, and knowledge-community substitution. Much less is known about whether LLMs change the distribution of software artifacts across the ecosystem. This is the gap I address here. If many developers rely on a small number of foundation models trained and aligned in similar ways, AI coding assistants may coordinate developers around common templates, libraries, parameter choices, defaults, and implementation patterns. Such convergence may be beneficial when it diffuses best practices, but it may also create technological monoculture by increasing dependence on shared implementations and common failure modes.

Digital Monoculture

Although prior work has focused heavily on the productivity effects of AI tools, scholars have also begun to raise concerns about digital monoculture. The most direct evidence comes from work on generative monoculture, referring to the tendency of LLMs to produce homogeneous outputs in response to user prompts. Jiang et al. (2025), for example, document substantial homogeneity in language models’ open-ended responses, showing that repeated samples from the same model and outputs from different models can converge on similar content. Wu et al. (2025) document related concerns in code generation for software development. These studies establish the potential for AI-driven monoculture: if many users rely on models that produce similar outputs, they may be exposed to similar templates, defaults, phrasing, and solution patterns.

However, the potential for homogeneity does not imply its actual manifestation in practice, nor does it dictate the specific details of any such manifestation. In most economically relevant settings, generative AI systems are used by humans who prompt, evaluate, reject, revise, and integrate model outputs. A growing literature on human-AI creativity, therefore, provides an important bridge between model-output homogenization and artifact-level monoculture. Zhou and Lee (2024), for example, study the adoption of text-to-image generative AI by digital artists and show that AI reshapes the geometry of creative production: average content novelty declines, peak content novelty rises, and visual novelty declines. Their findings suggest that generative AI does not make human creative work novel in any uniform sense. Rather, it compresses some dimensions of output while expanding others. Relatedly, Zhou, Lee, and Gu (2025) examine whether AI-assisted creative-frontier expansion is driven by a small number of highly productive “masterminds” or by a broader “hive mind” of contributors. Their evidence highlights that ecosystem-level novelty can depend not only on the novelty of the typical artifact, but also on how contributions are distributed across producers.

These studies are closely related to the present paper because they examine human-AI production rather than model outputs alone. They also highlight an important measurement issue. In broad creative domains such as art, writing, or design, the relevant task space is weakly bounded. If the observed frontier of artifacts expands after AI adoption, this may reflect genuine exploration of new creative territory, but it may also reflect the importation of already-existing concepts, styles, or

motifs from the model’s external training distribution into the focal platform. Conversely, if artifacts become more similar on average, it may be difficult to distinguish convergence in underlying ideas from convergence in surface style, prompt conventions, platform incentives, or user composition. The present paper addresses this challenge by studying software contests, where the task environment is more sharply defined. Within a Kaggle competition, participants work with the same data, pursue the same objective, and are evaluated by the same scoring rule. This makes within-contest similarity and within-contest conceptual span more directly interpretable as variation in approaches to a shared problem.

The possibility that shared AI systems shape human artifacts is also related to older concerns about technological and algorithmic monoculture. Concerns about algorithmic monoculture initially arose from the observation that many organizations increasingly rely on common machine-learning systems for screening and evaluation, such as in lending and labor-market contexts. Kleinberg and Raghavan (2021), proposing this notion, present an analytical model of the consequences for social bias and welfare in applicant screening. More recently, Toups et al. (2023) and Bommasani et al. (2026) provide empirical evidence of related dynamics in practice. In these settings, monoculture arises because shared decision-support systems can coordinate human or organizational decisions, reducing diversity in outcomes and potentially amplifying correlated errors.

With the rise of LLMs, concerns about shared technologies and shared decision-making have expanded to a wider range of contexts. Messeri and Crockett (2024), for example, raise concerns about the potential emergence of scientific monoculture. Related empirical work documents the homogenizing effects of LLMs on content production and work product (Dell’Acqua et al. 2026; Padmakumar and He 2024), as well as on ideation and creativity (Anderson et al. 2024; Sourati et al. 2026). Liang et al. (2024) further show that LLMs are affecting the content of academic peer review. Taken together, this literature suggests that LLMs may shape not only what individuals produce but also the space of ideas and evaluations they consider.

The extent of such homogenization, however, likely depends on the context and manner of LLM use. One reason is that users adapt to model behavior. Jahani et al. (2026) study prompt adaptation, or the way users adjust their inputs in response to evolving model capabilities, and show that such adaptation can be an important complement to generative AI performance, especially when tasks have clear objectives and feedback. This point is directly relevant to AI-induced monoculture: shared models may expose users to similar initial suggestions, but users need not respond to those suggestions in the same way. When users have clear goals, repeated feedback, and incentives to improve, they may learn to prompt, evaluate, reject, and revise AI-generated output in ways that preserve variation in final artifacts.

Incentives may further shape whether AI assistance homogenizes or diversifies human production. Gartenberg et al. (2026), analyzing article submissions and peer review at a major academic journal around the emergence of LLMs, argue that academic “publish or perish” incentives may encourage greater output while reducing writing quality and narrowing topic span. This point is important for the present paper because Kaggle contests combine forces that should promote homogenization with forces that may counteract it. Participants share a narrow task, public examples, and visible performance benchmarks, all of which should make convergence easier to observe. At the same time, leaderboard incentives and rapid feedback reward differentiation, potentially preserving conceptual diversity even as implementation details standardize.

This paper contributes to the digital monoculture literature by distinguishing among levels of abstraction in human-AI software production. Prior work establishes that LLM outputs can be homogeneous and that AI-assisted human artifacts can change in novelty, frontier span, and distribution across producers. Much less is known about whether generative monoculture propagates into the code developers ultimately submit, or whether artifact-level convergence extends to the underlying algorithmic approaches embodied in that code. By measuring within-contest similarity in both syntactic and semantic embedding spaces, I examine whether AI coding assistance induces technological monoculture in software implementations and whether that convergence is accompanied by algorithmic monoculture in higher-level solution strategies.

Methods

I assembled two large code corpora to study the diffusion of AI coding assistance on developer practice: Kaggle contest kernels and a panel of public GitHub repositories. Both span January 2019 through mid-2026, permitting a pre/post comparison around the public release of ChatGPT (November 30, 2022) and the general availability of GitHub Copilot (June 21, 2022).

Kaggle Contest Submissions. I obtained Kaggle kernel source code and metadata from the public Meta-Kaggle and Meta-Kaggle-Code data dumps in early July of 2026. My primary sample comprises publicly posted Python kernels submitted to Kaggle competitions. I link each kernel to a contest via the Kernel Version Competition Sources table. I then restrict the sample to competitions in Kaggle's Featured, Research, Recruitment, and Analytics categories, excluding Playground (prize-free) and 'getting-started tutorial' competitions, given they lack the incentive structure of genuine competitions. I also exclude essay and report competitions, where submissions are primarily prose rather than code (e.g., the 2023 Kaggle AI Report). To mitigate concerns that pairwise similarity may simply reflect copying via Kaggle's forking mechanism, I exclude forked kernels and project-language-template starter kernels. The resulting panel comprises 79,735 unique contest submissions to 473 distinct competitions, from 39,034 unique users. Among these submissions, 32,011 (40.1%) explicitly set a random seed. Of those, 57% (18,298) set the literal value 42 at least once, with the vast majority of these occurring in recent months and years.

Within-contest sample. The seed-choice analysis considers the full set of described kernels, with kernels timestamped by their submission date. For more general measures of code homogenization, namely, within-contest similarity and contest-level code span measures, I further limit the sample in three ways. First, I retain only completed competitions as of the data collection date (July 2, 2026). I remove in-progress competitions primarily to address the concern that a constructed measure of contest-level code span may be premature and mechanically lower than those of completed contests. Second, within each retained competition, I retain only kernels created on or before the contest deadline while the competition was active. Roughly one quarter of contest-linked kernels are published *after* the contest deadline. For long-lived, popular competitions, these post-deadline kernels accumulate over many years, such that many kernels published in 2026 remain attached to competitions that closed as early as 2015; including these post-deadline kernels complicates the timestamping of kernels used to construct contest-level measures of code span. Finally, because a single participant may publish many distinct, yet incremental kernels associated with a single contest, I take the participant, rather than the kernel, as the unit of analysis, representing each contestant by their most recent pre-deadline submission to the competition. Each retained competition is time-stamped by its submission deadline. These

restrictions yield a final sample of 252 competitions where at least 20 participants are available to construct within-contest pairwise-distance measures, and 205 competitions where at least 50 participants are visible to construct the contest span (spectral entropy) measure, over the 2019 to 2026 window.

GitHub ML Repositories. For the seed-choice analysis on GitHub, I assemble a panel of public Python files from repositories whose code uses standard machine-learning or scientific-computing libraries (i.e., that import any of numpy, scikit-learn, tensorflow, pytorch, keras, jax, xgboost, lightgbm, catboost, or pandas). Repositories were sampled through the GitHub Search API, stratified by buckets of repository star counts and time, from January 2019 through mid-2026. The repositories were cloned locally, and the code was then parsed to identify explicit seed-setting calls. Note that each repository file is dated based on its repository’s month of creation. Because files may be added to a repository after it was created, these date stamps are coarser than the submission timestamps I employ for the Kaggle sample. The resulting panel includes 19,938 distinct repositories with 303,101 Python files. Approximately 2.7% (8,303) of these files set an explicit seed value. Of those, 2,485 set a seed of 42, with the majority concentrated in recent months and years.

Random Seed Extraction

I extract explicit seed-setting calls from each Python source file via regular-expression matching of canonical patterns from common machine-learning libraries, including `random.seed(N)` and `seed = N` kwargs for Python’s built in random module, `np.random.seed(N)` for NumPy, `random_state=N` for scikit-learn, `tf.random.set_seed(N)` and `tf.set_random_seed(N)` for TensorFlow, `torch.manual_seed(N)` and `torch.cuda.manual_seed(N)` for PyTorch. The pattern extractor also identifies NumPy Generator seeds (`np.random.default_rng(N)`), `everything(N)` and `set_seed(N)` helper wrappers (including PyTorch Lightning’s), as well as capitalized seed constants (e.g., `RANDOM_SEED = N` or `RANDOM_STATE = N`), as long as they are defined outside a comment. Prior to matching, I strip Python comments and triple-quoted docstrings so that seed values only count if they exist outside a comment block. A file is marked as using “seed = 42” if any pattern captures the literal integer 42 anywhere in its uncommented source. For aggregate share figures, I compute the share of seeded files (or kernels) using each captured seed value within a given month or quarter.

Code Embeddings

For the Kaggle contest panel, I construct two embeddings for each kernel that capture, respectively, surface-level syntactic and higher-level semantic similarity. For surface-level syntactic embeddings, I construct a sparse TF-IDF representation of each kernel over unigrams and bigrams of code tokens, following standard practice (Salton and Buckley 1988). I apply sublinear term-frequency scaling, inverse-document-frequency weighting, minimum and maximum document-frequency thresholds (a minimum of 10 documents, and a maximum of 95% of documents), and retain the top 20,000 scaled features. I then reduce the resulting sparse matrix to 1,024 dimensions via truncated singular value decomposition (Halko, Martinsson, and Tropp 2011).

For higher-level semantic embeddings, I obtain embeddings of the same contest submissions from Voyage AI’s voyage-code-3 model (Voyage AI 2024a) via their paid API. These are 1,024-dimensional embeddings derived from a code-pretrained embedding model designed specifically for code retrieval tasks. Each contest submission is embedded as a single vector; submissions longer than the model’s 32,000-token input limit are truncated to the first 32,000 tokens. Notably, voyage-code-3 is trained with Matryoshka representation learning (Kusupati et al. 2022), which nests smaller

embeddings within larger ones so that the leading dimensions carry the most information, loosely analogous to the variance ordering that the truncated SVD imposes on the TF-IDF space; this makes the two representations structurally parallel. The similarity and rank measures I employ are invariant to rotations of the embedding basis, however, so no result depends on this ordering. Unlike TF-IDF, Voyage-code-3 is trained on a large code corpus with a retrieval objective, so it places submissions near one another in the embedding space when their code accomplishes more similar tasks, even if they differ substantially in surface syntax.

In Appendix D, I evaluate a series of alternative embedding approaches, including the focal approaches, in terms of the aspects of code that they capture. I demonstrate there that the two focal embedding representations are best suited to capturing the aspects of code I have suggested above: syntax (literal code, as written) versus semantics (intent or solution approach). There, I construct a controlled corpus of 20 code snippets that reflects the cross-product of five code tasks and four variable-naming conventions. Importantly, all function and variable names are ‘neutralized’ such that they reveal neither the problem nor the algorithm. Considering eight distinct embedding approaches, I show that those from Voyage code embeddings are the *only* representation that successfully assigns two snippets to similar positions in the embedding space when they share a common high-level intent, i.e., when they are solving the same problem, and when they employ distinct variable naming conventions. By contrast, TF-IDF embeddings provide the most syntactic treatment of code, placing snippets closest together in embedding space primarily based on surface syntax, i.e., shared variable naming conventions, even when the code pursues a different solution approach.

Measures of Code Homogenization

For each embedding type, I construct two complementary measures of within-contest homogenization.

Submission-level pairwise distance. For each kernel, i , submitted to contest, c , I compute novelty as the mean cosine distance between i and all other kernels submitted to the same contest. Because learned embedding models, particularly Voyage-code-3 in this case, are known to exhibit anisotropic embedding geometry, i.e., by default, the raw vectors share a strong common-axis component that mechanically inflates raw pairwise cosines, I first globally center each embedding, subtracting the corpus mean, and then re-normalize all embeddings before computing cosine distances. Note that this is a standard practice in the embedding analysis literature (Mu and Viswanath 2018; Ethayarajh 2019). Additionally, to avoid noise in the resulting measure, I enforce a minimum threshold of 20 public contest submissions; that is, I exclude contests with fewer than 20 public submission kernels from this analysis.

The resulting pairwise distances reflect residual content-discriminating dissimilarity. Finally, I aggregate these pairwise distances per contest. Applying within-contest aggregation in this manner ensures that the resulting measures are based solely on within-contest code similarity, not cross-contest similarity. Were I to construct pairwise comparisons *across* contests, my measure would be influenced by variation over time in the similarity versus novelty of contest objectives. My approach ensures this is not the case here. A decline in mean within-contest pairwise distance over time indicates that submissions are becoming less distinctive relative to their contemporaneous peers.

Contest-level effective rank. Following Roy and Vetterli (2007), I define the effective rank of a contest's kernel embedding set as $\exp(H)$, where $H = -\sum_k p_k \log p_k$ and $p_k = s_k^2 / \sum_j s_j^2$. Here, s_k are the singular values of the contest-mean-centered embedding matrix. Effective rank captures the number of independent directions along which kernels spread within the contest: lower values indicate that variation concentrates along fewer directions; higher values indicate that variation spans many directions. Effective rank is a within-contest geometric measure that complements pairwise distance because contest submissions can be moderately close to one another, on average, yet still vary along many independent directions in the embedding space, or vice versa. Because effective rank increases with sample size (a larger number of kernel embeddings can increase the likelihood that the embeddings span more orthogonal components merely by chance), I randomly sample 50 submissions per contest and use the resulting embeddings to calculate the rank measure. I repeat this process 3 times, resampling 50 kernels per contest without replacement, and ultimately report the mean result per contest. Additionally, to ensure a fixed embedding count per contest sample, I exclude contests with fewer than 50 public kernel submissions from this analysis.

Statistical Tests

For each monthly or contest-level series depicted graphically, I also estimate an interrupted time-series (ITS) regression of the outcome, modeling the dependent variable as a function of a linear time trend (in months, centered at ChatGPT's public release on November 30, 2022), an indicator for the post-release period, and the interaction between the two, employing heteroskedasticity-robust (HC3) standard errors. Specifically, I estimate:

$$y_t = \beta_0 + \beta_1 t + \beta_2 Post_t + \beta_3 (t \times Post_t) + \varepsilon_t,$$

where t is time in months, centered at ChatGPT's public release, i.e., $t = 0$ on November 30, 2022, with $Post_t$ being an indicator equal to 1 when $t \geq 0$ and zero otherwise. Under this parameterization, β_1 captures the pre-release trend, β_2 captures any discrete level shift at the time of release, and β_3 captures the change in trend thereafter. Because t is centered on the release date, the intercept, β_0 , reflects the value of the counterfactual outcome implied by the pre-release trend, at the release date; the fitted level at the release itself is reflected by $\beta_0 + \beta_2$. Table 1 reports ITS estimates for the headline outcomes; other tests are noted in-line in the text and appendices. I also apply data-driven binary segmentation breakpoint detection (Killick and Eckley 2014), allowing for up to three mean-shift breaks under an asymptotic penalty at the 5% level. Finally, for analyses broken down by contestant tier (Figures 6 and 7), I estimate the ITS regressions jointly for both series, with standard errors clustered by competition. Unless noted otherwise, the fitted trend curves displayed in the figures are penalized-spline generalized additive model (GAM) smooths (Wood 2017).

Results

I begin by analyzing the convergence of random seed values over time, both in Kaggle kernel submissions and GitHub repositories. Figure 1 documents the share of seeds equal to 42 in Kaggle kernels (Figure A1 in the appendix presents a similar plot for the prevalence of seed 42 in new GitHub machine learning repositories over time). I observe clear evidence that the share of seeds equaling 42 began to rise sharply around the time of ChatGPT's introduction in late November of 2022. In Table 1, I find that the interrupted time-series regression, used to quantify the break, indicates that

the share of seeded Kaggle kernels employing 42 rises sharply post-ChatGPT. Despite drifting upward prior to ChatGPT’s release (at a rate of +0.42 percentage points per month), the increase roughly tripled post-release (by +0.81 percentage points per month, $p < 0.001$). It should be noted, however, that model-free breakpoint detection identifies post-2022 structural breaks in March 2023 and June 2024, approximately aligned with the releases of GPT-4 (March 2023) and GPT-4o (May 2024). The degree to which seed values have converged is quite extreme; nearly 100% of seed values in newly published Kaggle contest kernels equal 42, and more than 70% of seeds set in new GitHub repositories do as well.

Next, I turn to general measures of code homogenization across different levels of abstraction, using the two embedding types described in the Methods section. Results are again reported visually, in Figures 2 and 3. Figure 2 plots the within-contest mean pairwise cosine distance over time, computed separately in the TF-IDF (syntactic) and Voyage code-3 (semantic) embedding spaces. Each dot represents one contest; lower values indicate that submissions to that contest are more similar to one another. Two patterns are immediately apparent. First, I observe a pronounced decline in within-contest pairwise distance in the TF-IDF (syntactic) space, post-AI, with the pattern accelerating in recent years, whereas pairwise distance in the Voyage (semantic) space remains essentially unchanged. This indicates that contest submissions have become markedly more alike in their literal syntax, yet apparently no more alike in their overarching solution approach. Mean within-contest TF-IDF pairwise distance falls from approximately 0.80 in 2019 to roughly 0.71 by mid-2026. The Voyage code-3 distance, by contrast, hovers near 0.56 over the same period. Contest submissions are thus becoming increasingly similar in their literal syntactic structure, yet no comparable in their underlying functional or conceptual approach. In the ITS estimation, the post-release trend change in within-contest pairwise distance is -0.0021 per month in TF-IDF space ($p < 0.001$) but is not statistically significant in Voyage space (Beta = -0.0004, $p = 0.58$; Table 1). Notably, in the syntactic space, this convergence manifests not only at the mean, but also at the median and 75th percentile of the pairwise distance measure.

Figure 3 reports the corresponding contest-level effective spectral rank over time, again in each embedding space. Where pairwise distance summarizes the average distance between contest submissions, effective rank captures the ‘span’ of dimensions covered by all submissions at the contest level, with a higher rank implying greater novelty. In the TF-IDF space, effective rank again declines markedly post-AI, falling from roughly 33 around the time of ChatGPT’s release to about 28 by mid-2026. This result indicates that the dimensions of syntactic variation among contest submissions are collapsing. Again, in sharp contrast, the Voyage code-3 effective rank remains essentially flat throughout the study period, hovering near 26, if not mildly positive. The ITS estimates are consistent with this contrast: the post-release trend change is -0.16 ranks per month in TF-IDF space ($p < 0.001$) versus +0.05 ranks per month in Voyage space ($p = 0.089$; Table 1). Importantly, these results are robust to accounting for potential shifts in contest topics or objectives; when contest-level effective rank is first residualized, via cross-fitted regression, with respect to both the dimensions of a contest-description textual embedding obtained from voyage-3, Voyage AI’s general-purpose text-embedding model (Voyage AI 2024b), and structured contest features, results persist. In this adjusted estimation, I observe a post-release trend change of -0.13 ($p = 0.01$) in effective rank in TF-IDF space, yet a mildly positive but statistically insignificant change in Voyage space (Beta = +0.02; $p = 0.47$).

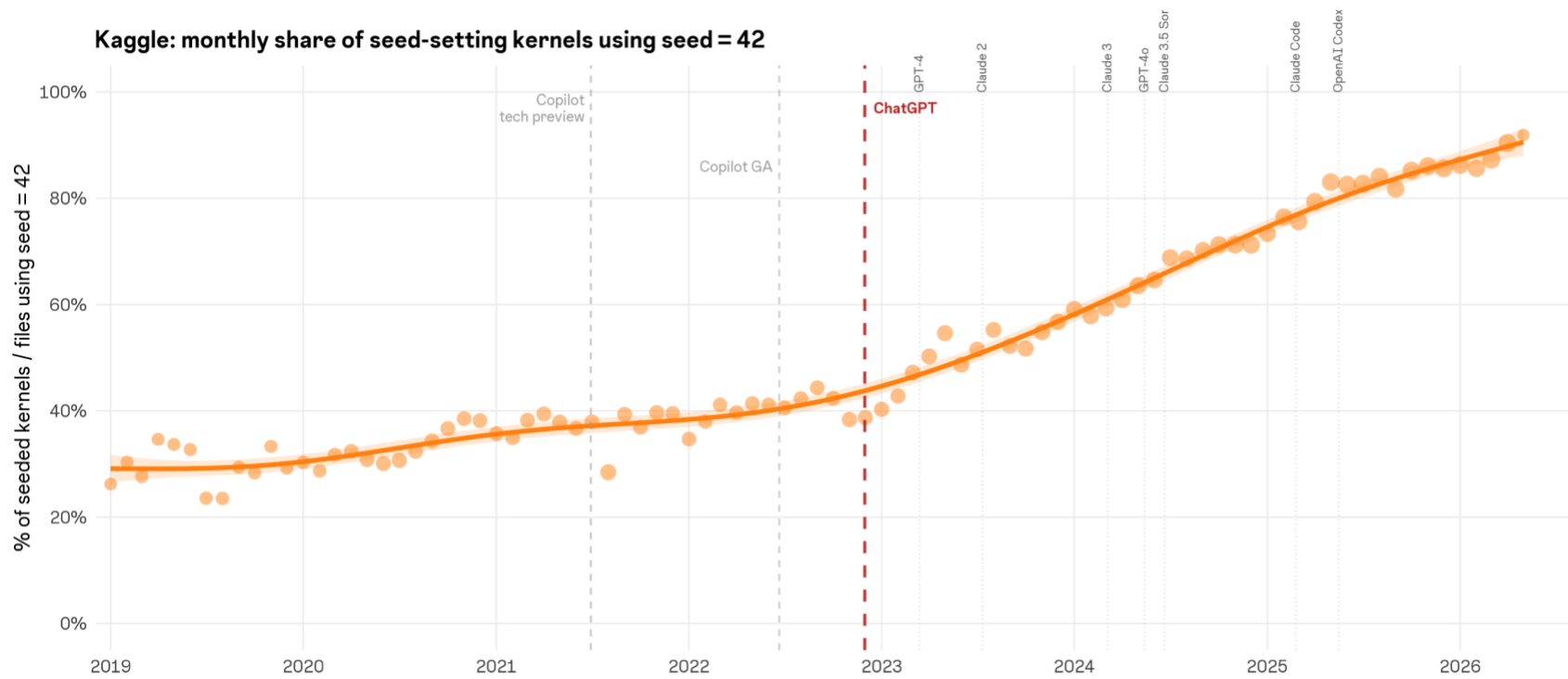


Figure 1. The percentage of seeded Kaggle kernels that employ seed = 42 at least once over time. The fraction of kernels employing this seed was relatively high prior to AI coding assistants, presumably due to the value’s prevalence in coding examples (e.g., on Stack Overflow) and in prior ‘forkable’ example notebooks on Kaggle. However, the share of seeded kernels using this value began to rise sharply following the introduction of AI coding assistants and now sits close to 95% of new Kaggle submissions.

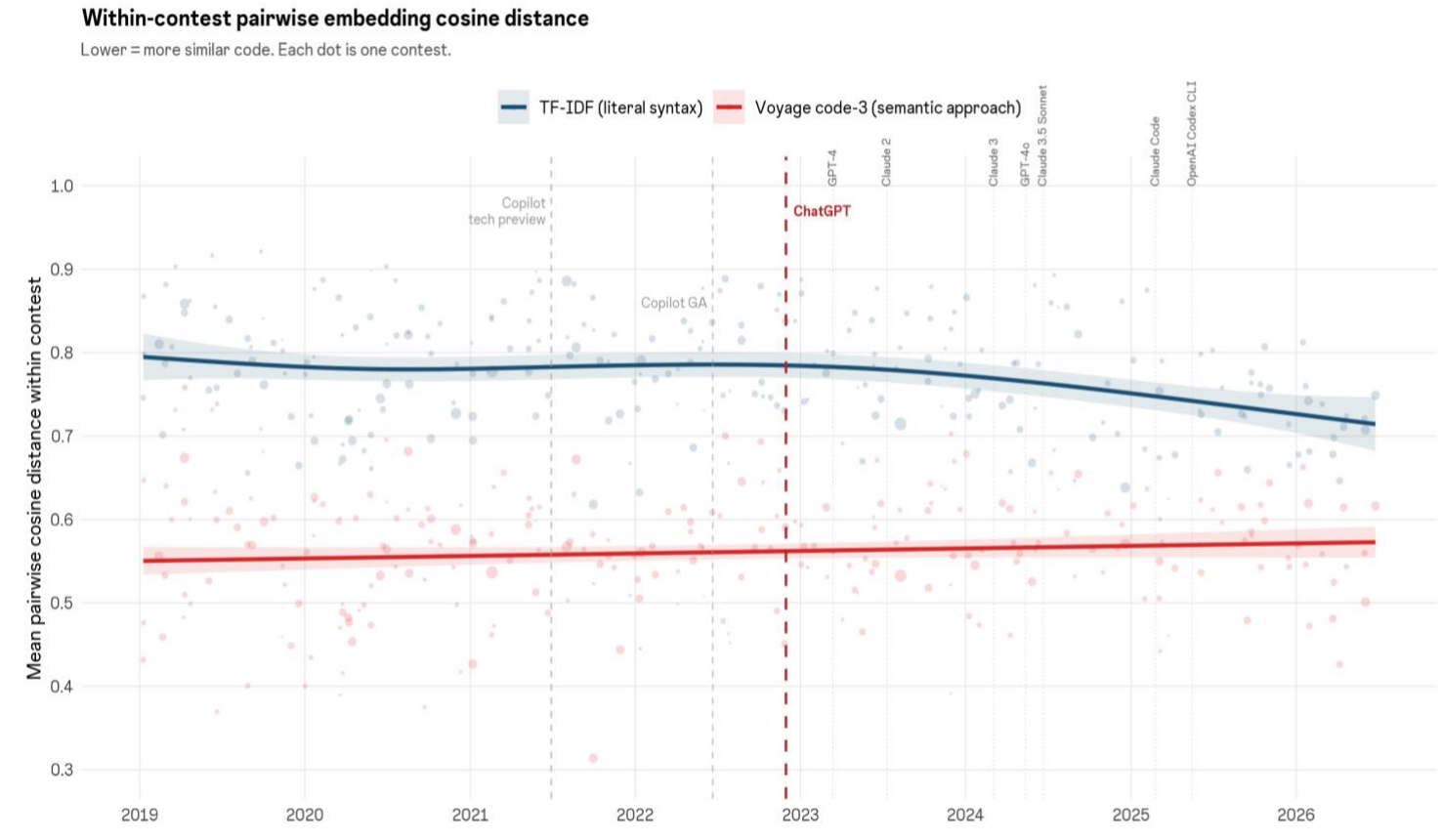


Figure 2. Within-contest mean pairwise cosine distance among Kaggle contest submissions over time. Calculations are computed in two embedding spaces: TF-IDF n-gram representations (capturing literal syntactic overlap) and Voyage code embeddings (capturing higher-level conceptual similarity). Each dot represents one contest, and curves reflect best-fit generalized additive models (GAMs). Lower values indicate that submissions to a given contest are more similar to one another. I observe a decline in pairwise distances over time in the syntactic embedding space, while the semantic space remains essentially flat, indicating that contest submissions have become more alike in surface code style but not in their underlying conceptual approaches. The vertical dashed line marks the release of ChatGPT in November 2022; other prominent LLM/code-assistant release dates during the observation window are shown along the top of the figure.

Per-contest effective spectral rank

Lower = variation concentrates on fewer directions. Each dot is one contest, subsampled to fixed N kernels.

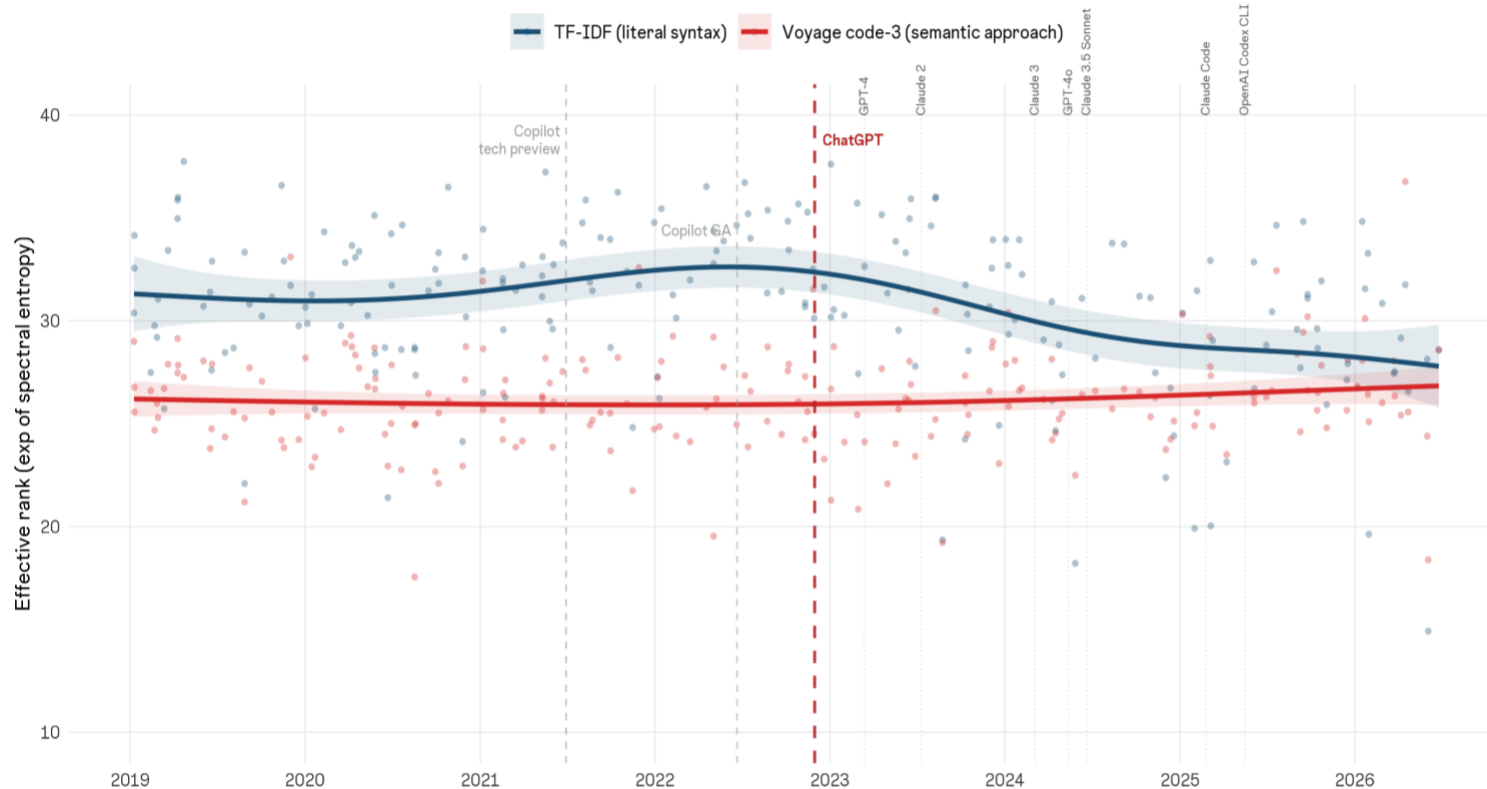


Figure 3. Per-contest effective spectral rank (the exponential of spectral entropy) among Kaggle contest submissions over time. Measures are computed in TF-IDF n-gram and Voyage code embedding spaces. Each dot represents one contest. Curves are generalized additive model (GAM) smooths. A random sample of 50 kernels per contest is used to enforce a fixed embedding count. Lower values indicate that within-contest variation concentrates along fewer independent components/directions in the embedding space, i.e., that submissions explore a narrower span of syntax or approaches. The TF-IDF effective rank begins to decline following ChatGPT’s release. The Voyage effective rank, by contrast, remains flat throughout the window and may even shift slightly positive, indicating that the breadth of conceptual approaches that contest participants collectively explore is largely unchanged. The vertical dashed line marks the release of ChatGPT in November 2022; other prominent LLM/code-assistant release dates during the observation window are shown along the top of the figure.

Table 1. Interrupted time-series estimates of code homogenization post ChatGPT’s release.

	Seed-42 share (pp)	Pairwise dist., TF-IDF	Pairwise dist., Voyage	Eff. rank, TF-IDF	Eff. rank, Voyage
Time (months)	0.418*** (0.016)	0.0000 (0.0004)	0.0001 (0.0005)	0.051* (0.021)	-0.001 (0.015)
Post-ChatGPT	-1.864† (1.121)	0.0167 (0.0160)	0.0173 (0.0162)	-1.061 (0.960)	-0.949 (0.712)
Time * Post-ChatGPT	0.815*** (0.039)	-0.0020*** (0.0006)	-0.0004 (0.0007)	-0.157*** (0.040)	0.052† (0.030)
Intercept	44.394*** (0.684)	0.7829*** (0.0108)	0.5573*** (0.0120)	32.898*** (0.514)	26.073*** (0.441)
Observations	133 months	252 contests	252 contests	205 contests	205 contests

Notes: Each column reports an OLS regression of the outcome on a linear monthly time trend centered at ChatGPT’s public release (November 30, 2022), an indicator for the post-release period, and their interaction. Column 1 reports monthly shares (percentage points) of seeded Kaggle kernels that set seed = 42 at least once; the remaining columns report one observation per completed competition, dated by its deadline: the mean within-contest pairwise cosine distance and the effective spectral rank, each computed in TF-IDF and Voyage code-3 embedding spaces. Heteroskedasticity-robust (HC3) standard errors in parentheses. † $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

Together, Figures 2 and 3 paint a nuanced picture. At the syntactic level, AI coding assistance appears to have reduced the average difference between any pair of submissions (Figure 2, TF-IDF) and the breadth of syntax across submissions to a given contest (Figure 3, TF-IDF). This is consistent with a rise in syntactic homogenization and, thus, technological monoculture. At the conceptual or functional level, however, the evidence for homogenization is less clear. Voyage pairwise embedding distances are essentially flat per contest, indicating no meaningful conceptual convergence on average. Further, crucially, the breadth of conceptual dimensions collectively explored across submissions to a contest (Figure 3, Voyage) does *not* appear to contract and even appears to expand mildly. Thus, contest participants collectively appear to be exploring at least as many independent conceptual directions in 2026 as they did in 2019, and the typical pair of submissions sits no closer together in conceptual space.

To better understand the full distributions of our two embedding-based measures over time, I present kernel density plots of the pairwise embedding distance and contest-level effective rank measures, by year (post-COVID), in Figures 4 and 5, respectively. In Figure 4, I present the density distribution of the pairwise embedding distance measure. I once again observe a systematic increase in pairwise similarity (a decline in pairwise novelty, seen as a shift to the left) among Kaggle contest submissions in the TF-IDF embedding space, beginning in 2023. The Voyage distribution, by contrast, is essentially stationary across years; the mean Voyage distance remains within a narrow band (roughly 0.54 to 0.58) with no systematic shift to the left, consistent with the lack of movement in the time series plot and with the ITS estimate reported above. In Figure 5, I similarly observe that the contraction in contest-level effective rank occurs only in the TF-IDF space and begins most clearly in 2024. The effective rank in the Voyage embedding space remains flat. Another apparent nuance that is apparent in Figure 5 is that the Voyage distribution widens in 2026. However, this ‘widening’ appears to be attributable primarily to a single competition with an unusually broad conceptual scope. Moreover, a Brown-Forsythe test of differences in variance in 2026 relative to prior years is not statistically significant, whether I include or remove that outlier contest.



Figure 4. Density of within-contest mean pairwise cosine distance among Kaggle submissions, per year. Densities (post-COVID) for TF-IDF n-gram representations (syntactic overlap) and Voyage code embeddings (conceptual similarity). I observe increases in pairwise similarity post AI, particularly in 2025-2026, following the uptake of tools like Claude Code.



Figure 5. Density of per-contest effective spectral rank among Kaggle submissions, per year. Densities (post-COVID) for TF-IDF n-gram and Voyage representations. I observe that spectral rank declines in the TF-IDF space, particularly beginning in 2023, yet holds or even rises in the Voyage embedding space.

These patterns suggest that while AI may be inducing convergence toward a smaller set of canonical syntactic templates, the underlying conceptual space that contest participants explore remains largely undiminished. Stated differently, AI-driven monoculture in this competitive context is real and substantial, but it has not affected the entirety of the ecosystem; many developers continue to explore sufficiently novel approaches in generating their contest solutions, such that, at the contest level, the breadth of ideas explored has persisted.

One factor that may contribute specifically to the decline in average pairwise novelty between contest submissions would be the entry of many new Kaggle participants who rely heavily on AI coding assistance tools, i.e., ‘vibe coders.’ To the extent that novices feel empowered to compete and begin participating on Kaggle in greater numbers, they may submit large volumes of AI-dominated code, driving up homogeneity. To examine this possibility, we reconstruct Figures 2 and 3, splitting the series based on Kaggle contestant tier; that is, we plot the series separately for Kaggle Novices and Contributors as compared to submissions more generally (including Experts, Masters, Grand Masters), where the latter group incorporates experienced contestants who have established a history of success on Kaggle. We present these split depictions in Figures 6 and 7. In general, we find that the headline results persist. As expected, we see that contestants with more experience and expertise (higher-tier contestants) exhibit higher code novelty generally, in both embedding spaces (in paired, within-contest comparisons, the Novice/Contributor set exhibits more submission-submission similarity than the broader submission set (the mean difference is 0.014 in TF-IDF space and 0.027 in Voyage space, both $p < 0.001$, with standard errors clustered by contest). Importantly, however, I observe roughly comparable post-release shifts in average pairwise novelty across all submissions (the novice submission set’s post-release shift does not differ significantly from the full submission set; $p = 0.70$ in TF-IDF space and $p = 0.62$ in Voyage space).

It is worth noting that the evidence I have reported to date associates code homogenization with the arrival of AI coding assistants solely on the basis of timing; my analysis does not directly measure the influence of AI usage. Thus, to better establish AI’s role, I undertake one final analysis, attempting to label AI usage per contest submission. I train an AI-usage classifier that detects LLM usage in code comments. This classification approach is based on prior work identifying that LLM coding assistants tend to produce distinct comments in code (Kang et al. 2024; Ye et al. 2026; Ji et al. 2026). I take a sample of human-generated Kaggle submissions from before November 2022 and a matched sample of one-shot generated code solutions for the same contests from four distinct LLMs (Claude 4.5 Haiku, Claude 4.6 Sonnet, Gemini 2.5 Flash, and OpenAI GPT-4o). Next, I extract the comments and markdown from each solution (stripping out actual code), and I use the comments to train a character-level n-gram-based classifier (3-, 4-, and 5-grams) to predict LLM usage. The resulting classifier is highly discriminative (see additional details in Appendix E).

Classifying all contest submissions in my dataset, I first observe that the share of Kaggle contest submissions that are classified as containing LLM-generated comments rises from roughly 1% before ChatGPT’s release to approximately 15% by mid-2026 (Figure E2). Using this AI score, I examine the association between AI use and code homogenization conditional on the contestant and contest; hence, my estimates are tied to within-contestant shifts over time, and within-contest differences across submissions, where different contestants exhibit higher versus lower AI usage after (versus before) AI’s emergence. Importantly, my broad findings hold.

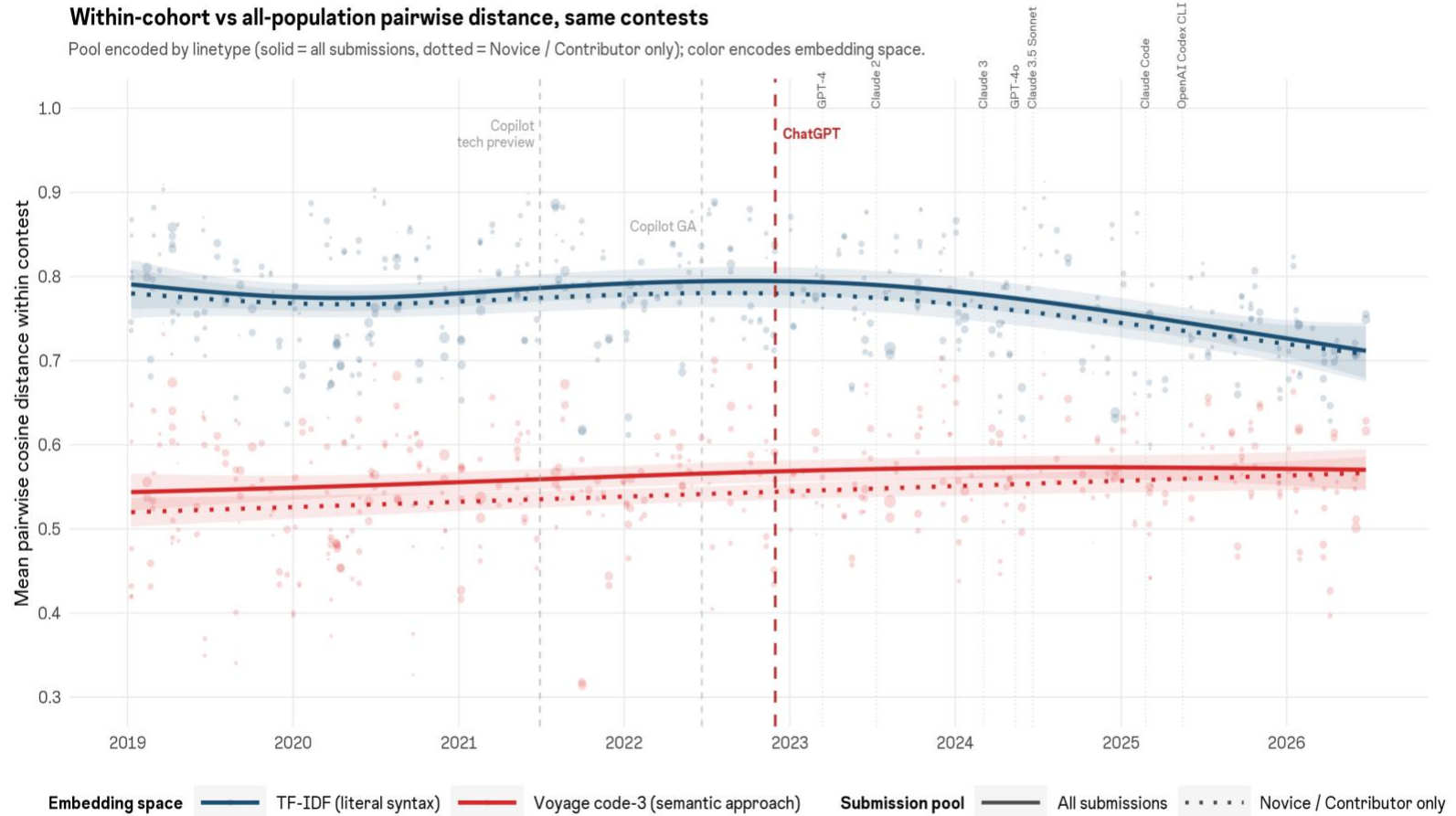


Figure 6. Within-contest mean pairwise cosine distance among Kaggle contest submissions over time, by contestant tier. Calculations are split by contestant tier (Novices/Contributors as dotted lines vs. everyone collectively as solid lines), computed in two embedding spaces: TF-IDF n-gram representations (capturing literal syntactic overlap) and Voyage code embeddings (capturing higher-level conceptual similarity). Each scatter dot represents one contest. Lower values indicate that submissions to a given contest are more similar to one another. The best-fit generalized additive model (GAM) curves show declines in pairwise distance over time in the TF-IDF space, regardless of contestant tier, indicating that submissions have become more similar in surface syntax. In the Voyage space, by contrast, pairwise distance stays essentially flat for both samples. Lower-tier contestants exhibit less novelty overall, but the decline over time is confined to syntactic space, consistent with AI-driven convergence in surface code style but not in underlying conceptual approach.

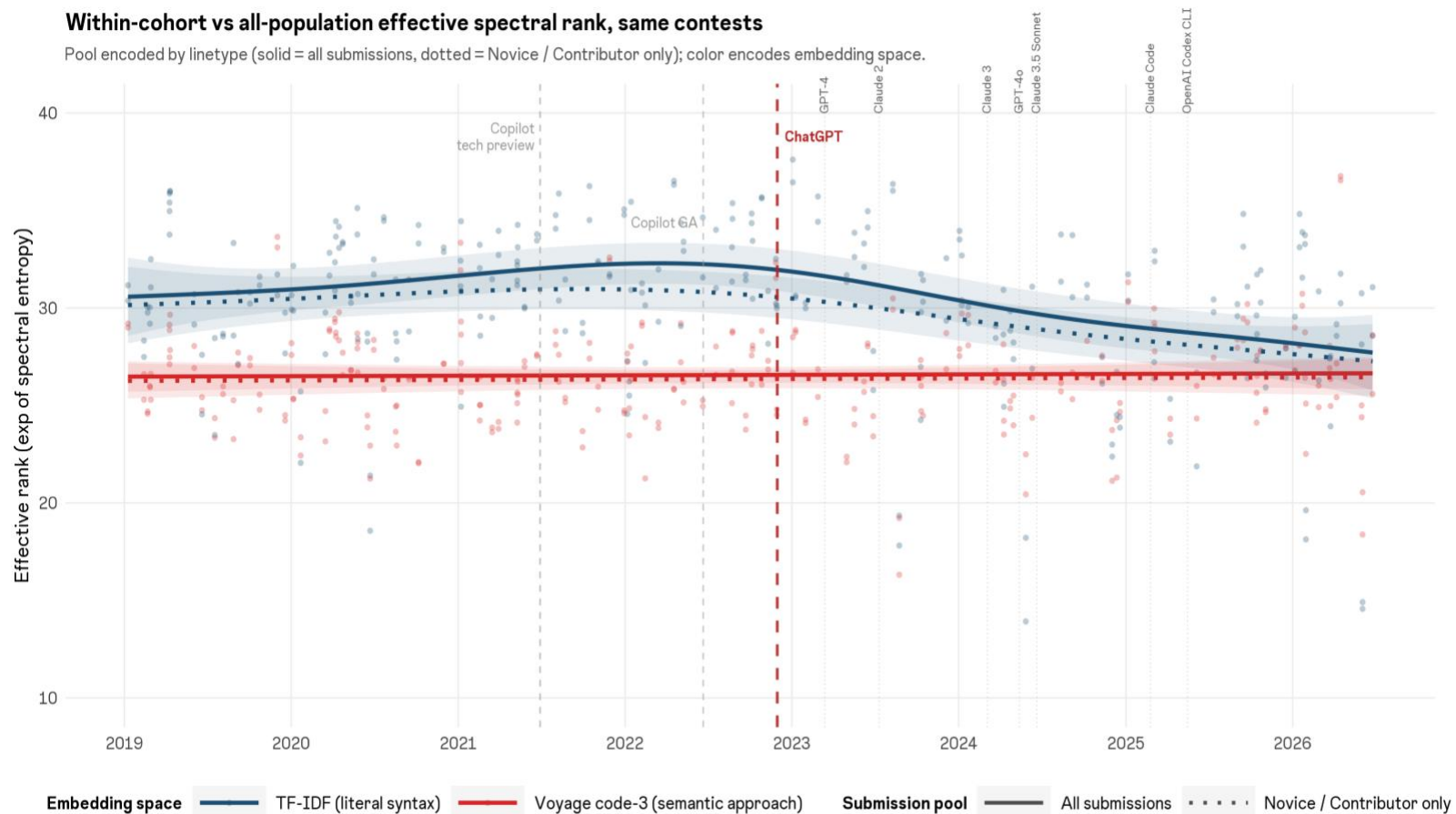


Figure 7. Per-contest effective spectral rank among Kaggle contest submissions over time, computed in TF-IDF n-gram and Voyage code embedding spaces, in total and specifically for Novice and Contributor contestants. Each point represents one contest, reflecting the average of three random samples of 50 submissions per contest. The best-fit curves are generalized additive models (GAMs). Lower values indicate that within-contest variation concentrates along fewer independent components/directions in the embedding space, i.e., that submissions explore a narrower span of syntax or approaches. The TF-IDF effective rank begins to decline following the emergence of AI coding tools, not merely for novices but for submissions, generally. The Voyage effective rank, by contrast, remains flat throughout the window, for novices and in general, indicating that the breadth of conceptual approaches that contest participants collectively explore is largely unchanged, even among relative novices.

Conditional on the contestant and competition, I find that a one-unit (full scale) increase in a submission's AI-comment score is associated with an 11.5-percentage-point increase in the probability that the submission sets a seed value to 42 ($p < 0.01$; Table E3). Aggregating to the contest level, the association between a contest's share of AI-labeled submissions and average pairwise similarity, I find a ten-percentage-point increase in AI share is associated with a 0.021 reduction in mean TF-IDF pairwise distance ($p < 0.001$) and a 1.4-point decline in TF-IDF effective rank ($p < 0.01$), yet no statistically significant change in either Voyage measure (Figure E3). Thus, I find that heavier AI use is systematically associated with greater code homogenization, though only in TF-IDF space. This evidence suggests that AI usage directly drives the patterns of code homogenization I have documented herein.

Discussion

This paper contributes to the literature on AI-driven homogenization in several ways. First, by measuring code similarity in both syntactic and semantic spaces, I show that AI's homogenizing effect on code is layered. The syntactic layer, what code literally looks like, has converged sharply. The semantic layer, what the code actually does, has not converged, neither in average pairwise similarity nor in the breadth of approaches collectively explored within a contest. Prior work has often treated AI-induced homogenization as a single phenomenon, often inferring algorithmic or conceptual convergence from surface-level similarities in generated outputs (Anderson et al. 2024; Padmakumar and He 2024; Wu et al. 2025). My results suggest that inference does not generalize. Convergence in code style can coexist with sustained breadth in conceptual approach, and treating these layers as separable enables more disciplined claims about where AI monoculture is and is not currently arising. The observed pattern is robust to the choice of embedding representation (Appendix F).

Second, the seed-42 finding is a clear illustration of what AI-driven technological monoculture looks like at the level of arbitrary implementation defaults. Random seed values are about the most innocuous-looking choice a developer can make: a single integer, often perceived as cosmetic, that nevertheless governs the stochastic behavior of an entire training pipeline. That one arbitrary value has come to dominate over 95% of newly seeded Kaggle contest submissions, and over 70% of seeded files in new ML-related GitHub repositories, suggesting that AI coding assistants are propagating their training-distribution defaults at scale.

The pattern is consistent with long-standing concerns about the ecosystem-level consequences of shared infrastructure (Stamp 2004; Tanfield-Taylor et al. 2026): systemic risk arises not from any individual choice but from the correlated structure of those choices across many actors. The substantive consequences here matter. Recent work has shown that random seed values meaningfully influence deep learning model performance and stability (Picard 2021; Zhi et al. 2023). To the extent that LLMs are synchronizing seed choices across developers, the resulting concentration may erode the independent resampling diversity that seeds were intended to support in the first place.

Third, the finding that conceptual breadth has been largely preserved in a competitive setting speaks to current debates about how AI changes the nature of human work (Becker et al. 2025; Hoffmann et al. 2025; Yeverechyahu et al. 2024). One view holds that LLMs flatten the distribution of human output by drawing everyone toward an "average" generated by the model's training distribution. A

competing view holds that LLMs are accelerants: by lowering the cost of execution, they expand the range of approaches that practitioners can feasibly explore. My results suggest both views describe parts of the truth, but at different levels of abstraction. At the syntactic level, the flattening view fits well, as code is converging on common templates, library calls, parameter choices, and arbitrary defaults. At the conceptual level, the accelerant view fits better in the competitive Kaggle context; contest participants collectively continue to explore a wide range of substantive approaches even as their code becomes stylistically uniform. This is most likely not because individual contestants are themselves more diverse in their own conceptual exploration, but because the conditions for exploration, particularly visible leaderboard incentives, financial prizes, and repeated low-cost experimentation, appear to support the survival of meaningful approach-level variation even as syntactic templates converge.

Some of the results I observe, particularly the rise in average pairwise similarity among contest submissions, may also be explained in part by a compositional shift in participants on Kaggle. Kaggle has become more accessible in the AI era; entry costs have fallen sharply, as LLM coding assistance allows novices to produce working code that would previously have required substantial human capital. Many of these new entrants likely rely heavily on AI-default templates. To the extent that these new entrants expand the pool of contestants while relying heavily on AI-generated code, the average similarity among submissions rises.

Importantly, however, my results are not solely driven by that dynamic. Repeating the analyses while splitting contestants by Kaggle performance tier (Figures 6 and 7) reveals heterogeneity. Contributors, the lowest tier among Kaggle contestants, exhibit the steepest declines in pairwise and contest-level code novelty. A compositional shift in contestants is thus playing some role here, though it does not explain the entirety of my results. More experienced contestants also exhibit declines in novelty, to a lesser degree. The contest-level effective rank in the Voyage code-3 space, most interestingly, remains essentially flat throughout the observation window for *both* groups of contestants (Figure 6, red lines). This is consistent with my central finding applying broadly, and not being driven solely by experienced developers: the breadth of conceptual approaches collectively explored within a contest has held fast, even as syntactic templates have converged, and this is not solely attributable to the persistence of experienced / expert coders.

Fourth, my findings have implications for the governance of AI coding assistants. The dominant policy and audit conversation around LLM-generated code has focused on individual-level concerns: whether the code is correct, secure, maintainable, and legally usable (Fu et al. 2025; Pearce et al. 2025; Vaithilingam et al. 2022). These concerns are important, but they treat each generated suggestion in isolation and miss the externality that emerges when many developers adopt the same suggestion. The seed-42 case is illustrative: each individual choice is harmless on its own, but the collective concentration and the resulting degradation of replication-relevant variation across the ecosystem create systemic risk. The same logic extends to other arbitrary implementation defaults: standard split-and-shuffle routines, default hyperparameter ranges in grid searches, default evaluation metrics, default error-handling idioms, and default library choices when multiple are functionally equivalent. Governance regimes that audit only individual-output quality will systematically underweight these ecosystem-level effects. Conversely, the comparative preservation of conceptual diversity I document on Kaggle suggests that incentive structures can shape the trajectory of AI-induced monoculture. Where developers face explicit rewards for substantive novelty, as in competitive programming, scientific research, or security work, the conceptual layer may be protected even as syntactic templates standardize. In settings without

such incentives, including routine production code, internal tooling, and configuration scripts, the conceptual layer may be more vulnerable.

Several limitations of this study deserve mention. First, my analysis focuses on Kaggle, which provides a uniquely well-defined unit of analysis (the contest) but is unusual in offering strong, visible, and financial incentives for substantive differentiation. The preservation of conceptual breadth I observe may not extend to non-competitive environments; replicating the analytical approach on contributions to shared codebases without comparable novelty incentives is an obvious next step. Second, my Kaggle panel reflects voluntary publication; contestants choose whether to publish their kernels and which versions to publish. I address this concern in part by restricting to original (non-fork) submissions and to non-Playground competitions, but selection on publication remains. Third, the two embedding spaces I employ, TF-IDF n-grams and Voyage code, are reasonable but not the only sensible choices. Alternatives such as structural embeddings derived from abstract syntax trees, embeddings of execution traces, or embeddings trained on functional equivalence might recover different patterns. Fourth, while my analyses establish a clear temporal correspondence between the rise of LLM coding assistants and the convergence patterns I document, the observational design cannot rule out alternative explanations for the timing.

Across measures and embedding spaces, the central pattern holds. Technological monoculture in code has risen sharply in the AI era, but algorithmic monoculture, at least at the contest level, has not kept pace. The methodology I develop here can be applied to other settings with well-defined coordinating objectives, such as academic replication packages, competitive-programming submissions, take-home interview problems, or coursework, to test whether the layered pattern I document is general or depends on the incentive structure of competitive contexts. The seed-42 finding opens a broader investigation of other arbitrary defaults that LLMs may be standardizing across the ecosystem. The within-contest measurement framework can be extended to investigate whether algorithmic monoculture eventually emerges as developers continue to adopt AI tools, model capabilities improve, and the contributor population shifts. Whether the conceptual frontier I observe today persists as those conditions evolve is an open question, and an important one for understanding the longer-term trajectory of AI in technical communities.

Conclusion

This paper has examined the extent to which documented homogeneity of LLM outputs, i.e., generative monoculture, propagates into the code that developers deploy and, ultimately, into the algorithmic approaches they explore. Using large-scale data on Kaggle contest submissions and public GitHub repositories from 2019 through mid-2026, I document a striking instance of LLM-driven technological monoculture in the choice of random seed values, with the seed "42" rising to dominate over 95% of seeded Kaggle contest kernels and over 70% of seeded ML-related GitHub files following the emergence of AI coding assistants. I then show, using complementary embedding-based measures of within-contest code similarity, substantial convergence in code syntax over time, more generally, both in terms of average pairwise distance among Kaggle submissions and in the dimensionality of contest-level code. However, regarding the latter, I show that contest-level convergence has occurred only in syntax, not in semantics. The number of independent conceptual directions explored within a Kaggle contest has remained essentially unchanged across the observation window. The broader point I make is that monoculture is hierarchical: convergence at lower levels of abstraction need not entail convergence at higher levels of abstraction. The

divergence in monoculture across levels of abstraction is plausibly driven by the institutional and incentive structure of the setting in which AI is used. For researchers and policymakers concerned with ecosystem-level fragility arising from workers' reliance on LLMs, these considerations are important to bear in mind: they speak to where and when AI homogenization is most likely to arise, at least for now, and they suggest approaches to managing this risk with respect to incentive design, measurement, and ecosystem-level auditing rather than a focus on evaluating individuals' work product.

Finally, although my evidence comes from software coding, similar results are plausibly arising in other domains. LLMs now mediate the production of prose, marketing copy, legal drafts, statistical analysis, and, increasingly, scientific research more generally. Early evidence indicates that LLMs homogenize creative writing and ideation much as they do code (Anderson et al. 2024; Padmakumar and He 2024), and that they may push research toward a narrower set of questions and framings (Traberg et al. 2026). The nuanced patterns I document here thus offer a broader lesson: while superficial text may converge, e.g., phrasing, structure, and word choice, the underlying space of ideas may not, at least in the presence of incentives that reward differentiation and novelty. My approach here also offers a path forward for measurement, enabling the study of these phenomena in other domains. A key requirement, however, is that workers must share a common coordinating objective, e.g., work on a particular legal question or case, applications to a particular grant call, a common research problem, or a logo design contest. Given a shared objective, it becomes possible to embed and compare work products at multiple levels of abstraction, and to ask whether the concrete or abstract aspects of work products are homogenizing and contracting, and when. The distinction here is quite important: convergence in form chiefly raises questions of style and attribution, whereas convergence in ideas would erode the diversity on which innovation, robustness, and scientific discovery depend.

References

- Anderson, B. R., Shah, J. H., & Kreminski, M. (2024, June). Homogenization effects of large language models on human creative ideation. In *Proceedings of the 16th conference on creativity & cognition* (pp. 413-425).
- Bommasani, R., Bana, S. H., Creel, K. A., Jurafsky, D., & Liang, P. (2026). Algorithmic Monocultures in Hiring. *arXiv preprint arXiv:2605.27371*.
- Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity. *arXiv preprint arXiv:2507.09089*.
- Brynjolfsson, E., Li, D., & Raymond, L. (2025). Generative AI at work. *The Quarterly Journal of Economics*, 140(2), 889-942.
- Bui, N. T., Savova, G. K., & Wang, L. (2025, December). Assessing the macro and micro effects of random seeds on fine-tuning large language models. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics* (pp. 41-46).
- Burtch, G., Lee, D., & Chen, Z. (2024). The consequences of generative AI for online knowledge communities. *Scientific Reports*, 14(1), 10413.

- Cui, K. Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2026). The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Management Science*.
- Dell'Acqua, F., McFowland III, E., Mollick, E., Lifshitz, H., Kellogg, K. C., Rajendran, S., ... & Lakhani, K. R. (2026). Navigating the jagged technological frontier: Field experimental evidence of the effects of artificial intelligence on knowledge worker productivity and quality. *Organization Science*, 37(2), 403-423.
- Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2024). GPTs are GPTs: Labor market impact potential of LLMs. *Science*, 384(6702), 1306-1308.
- Ethayarajh, K. (2019, November). How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 55-65).
- Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., & Chen, J. (2025). Security weaknesses of copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 34(8), 1-34.
- Gartenberg, C., Hasan, S., Murray, A., & Pierce, L. (2026). More versus better: Artificial intelligence, incentives, and the emerging crisis in peer review. *Organization Science*.
- Halko, N., Martinsson, P. G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2), 217-288.
- Hoffmann, M., Boysel, S., Nagle, F., Peng, S., & Xu, K. (2025). Generative AI and the Nature of Work. *Harvard Business School Strategy Unit Working Paper*, (25-021), 25-021.
- Jahani, Eaman, Benjamin S. Manning, Joe Zhang, Hong-Yi TuYe, Mohammed Alsobay, Christos Nicolaides, Siddharth Suri, and David Holtz. "Prompt adaptation as a dynamic complement in generative ai systems." *Information Systems Research* (2026).
- Ji, Y., Wang, J., Zhou, Y., Chen, F., & Yang, H. An Exploratory Study on LLM-Generated Code and Comments in Code Repositories. *Available at SSRN 5875028*.
- Jiang, Liwei, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. (2026). Artificial hivemind: The open-ended homogeneity of language models (and beyond). *Advances in Neural Information Processing Systems*, 38.
- Kang, S., Milliken, L., & Yoo, S. (2024). Identifying inaccurate descriptions in LLM-generated code comments via test execution. *arXiv preprint arXiv:2406.14836*.
- Killick, R., & Eckley, I. A. (2014). changepoint: An R package for changepoint analysis. *Journal of Statistical Software*, 58(3), 1-19.
- Kleinberg, J., & Raghavan, M. (2021). Algorithmic monoculture and social welfare. *Proceedings of the National Academy of Sciences*, 118(22), e2018340118.
- Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P., & Farhadi, A. (2022). Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35, 30233-30249.

- Li, S., Le, H., Xu, J., & Salzmman, M. (2025, May). Enhancing compositional text-to-image generation with reliable random seeds. In *International Conference on Learning Representations* (Vol. 2025, pp. 45444-45465).
- Liang, W., Izzo, Z., Zhang, Y., Lepp, H., Cao, H., Zhao, X., ... & Zou, J. Y. (2024). Monitoring AI-modified content at scale: A case study on the impact of ChatGPT on AI conference peer reviews. *arXiv preprint arXiv:2403.07183*.
- Messeri, L., & Crockett, M. J. (2024). Artificial intelligence and illusions of understanding in scientific research. *Nature*, 627(8002), 49-58.
- Mu, J., & Viswanath, P. (2018). All-but-the-top: Simple and effective post-processing for word representations. In *6th International Conference on Learning Representations, ICLR 2018*.
- OpenAI. (2024). New embedding models and API updates. <https://openai.com/index/new-embedding-models-and-api-updates/>.
- Noy, S., & Zhang, W. (2023). Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, 381(6654), 187-192.
- Padmakumar, V., & He, H. (2024, May). Does writing with language models reduce content diversity? In *International Conference on Learning Representations* (Vol. 2024, pp. 642-669).
- Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2025). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. *Communications of the ACM*, 68(2), 96-105.
- Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of ai on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590*.
- Picard, D. (2021). Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*.
- Roy, O., & Vetterli, M. (2007, September). The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference* (pp. 606-610). IEEE.
- Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5), 513-523.
- Sourati, Z., Ziabari, A. S., & Dehghani, M. (2026). The homogenizing effect of large language models on human expression and thought. *Trends in Cognitive Sciences*.
- Stamp, M. (2004). Risks of monoculture. *Communications of the ACM*, 47(3), 120.
- Tanfield-Taylor, J., Mirizio, A., Greene, B., & Ensmenger, N. (2026, April). Beyond Microsoft and Monsanto: Denaturing the Monoculture Metaphor in Computing. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (pp. 1-17).
- Toups, C., Bommasani, R., Creel, K., Bana, S., Jurafsky, D., & Liang, P. S. (2023). Ecosystem-level analysis of deployed machine learning reveals homogeneous outcomes. *Advances in Neural Information Processing Systems*, 36, 51178-51201.
- Traberg, C. S., Roozenbeek, J., & Van Der Linden, S. (2026). AI is turning research into a scientific monoculture. *Communications Psychology*, 4(1), 37.

- Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022, April). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI conference on human factors in computing systems extended abstracts* (pp. 1-7).
- Voyage AI. (2024a, December 4). voyage-code-3. *Voyage AI Blog*. <https://blog.voyageai.com/2024/12/04/voyage-code-3/>
- Voyage AI. (2024b, September 18). voyage-3 & voyage-3-lite: A new generation of embedding models. *Voyage AI Blog*. <https://blog.voyageai.com/2024/09/18/voyage-3/>
- Wang, C., Huang, K., Zhang, J., Feng, Y., Zhang, L., Liu, Y., & Peng, X. (2025, April). LLMs meet library evolution: Evaluating deprecated API usage in LLM-based code completion. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)* (pp. 885-897). IEEE.
- Wood, S. N. (2017). *Generalized additive models: An introduction with R* (2nd ed.). Chapman and Hall/CRC.
- Wu, F., Black, E., & Chandrasekaran, V. (2025, May). Generative monoculture in large language models. In *International Conference on Learning Representations* (Vol. 2025, pp. 33068-33107).
- Ye, H., Ran, F., Xu, W., & Zhou, M. (2026). The Readability Spectrum: Patterns, Issues, and Prompt Effects in LLM-Generated Code. *arXiv preprint arXiv:2605.13280*.
- Yeverechyahu, D., Mayya, R., & Oestreicher-Singer, G. (2026). The impact of large language models on open-source innovation: Evidence from GitHub copilot. *arXiv preprint arXiv:2409.08379*.
- Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using GitHub Copilot: An empirical study. *arXiv preprint arXiv:2303.08733*.
- Zhi, Y., Xie, X., Shen, C., Sun, J., Zhang, X., & Guan, X. (2023). Seed selection for testing deep neural networks. *ACM Transactions on Software Engineering and Methodology*, 33(1), 1-33.
- Zhou, E., & Lee, D. (2024). Generative artificial intelligence, human creativity, and art. *PNAS Nexus*, 3(3), pgae052.
- Zhou, E. B., Lee, D., & Gu, B. (2025). Who expands the human creative frontier with generative AI: Hive minds or masterminds? *Science Advances*, 11(36), eadu5800.
- Zhu, T., Wen, Z., Liu, S., Li, Y., Zhang, T., & Xia, X. (2026). Assessing the Capability of LLMs for Deprecated API Usage Updating from Natural Language Descriptions. *ACM Transactions on Software Engineering and Methodology*.

Appendix A: GitHub Seed 42 Share

An interrupted time-series regression of the monthly share of GitHub ML repository files using seed = 42 at least once ($n = 86$ months) shows the same pattern as Kaggle contest submissions: a modest pre-release upward trend ($+0.21$ percentage points per month, $p = 0.025$) that accelerates by a factor of five after ChatGPT's release. The estimated trend change following ChatGPT's release is $+0.89$ percentage points per month and is highly significant ($p < 0.001$).

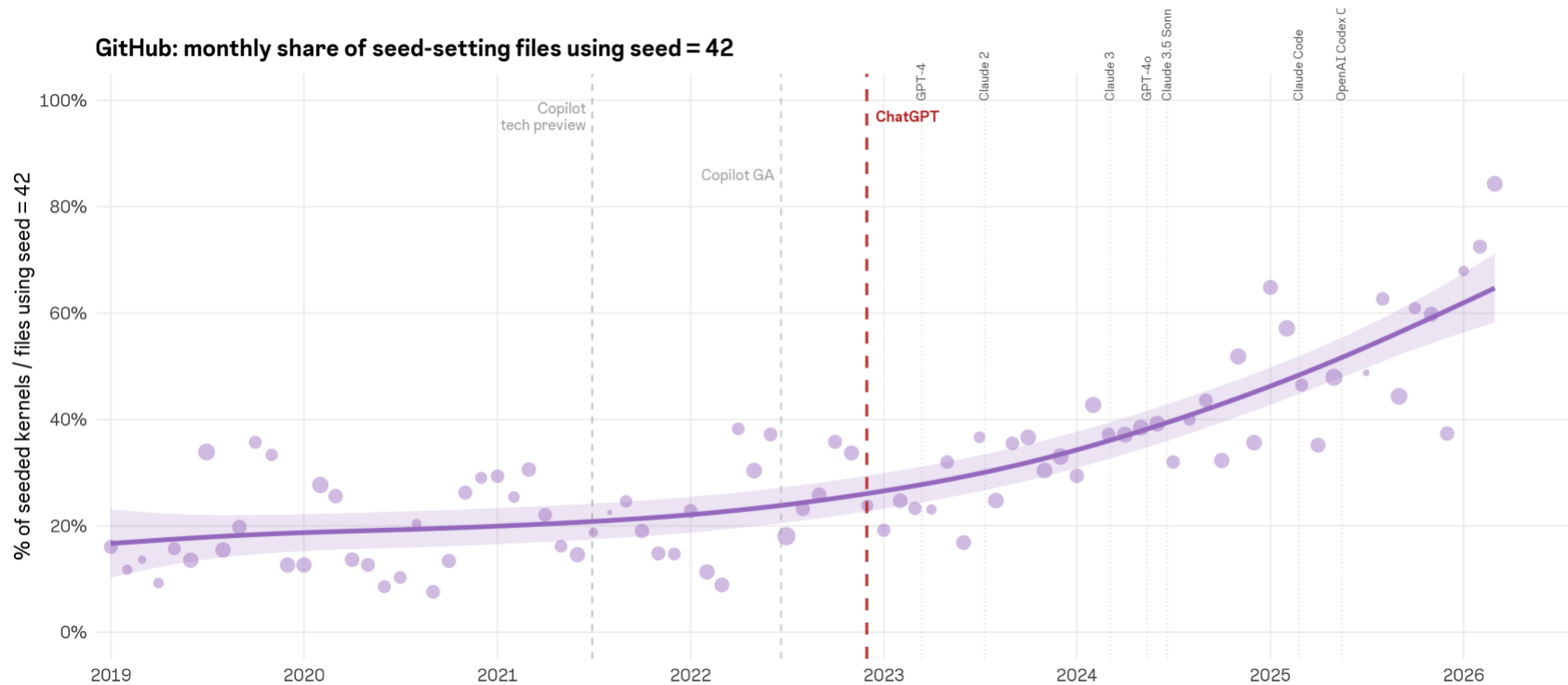


Figure A1. The percentage of seeded GitHub ML repository files employing seed = 42 over time. Each point represents a month of GitHub repository submissions, capturing the share of files including seed = 42 at least once. The fraction was relatively high prior to the emergence of AI coding assistants, though less so than on Kaggle. However, the share of seeded repository files with the value 42 began to rise sharply following the introduction of AI coding assistants and now stands at close to 70% of new GitHub repository submissions.

Appendix B: Kaggle Submission Forking Rate

An interrupted time-series fit of per-contest fork rates ($n = 253$ competitions) indicates that fork rates were rising before ChatGPT's release ($+0.22$ percentage points per month, $p = 0.004$) and that trend reversed subsequently (post-release trend change of -0.26 percentage points per month, $p = 0.065$), with no significant level shift at the point of ChatGPT release itself ($p = 0.99$). Rising fork rates, therefore, cannot explain post-release declines in average code novelty.

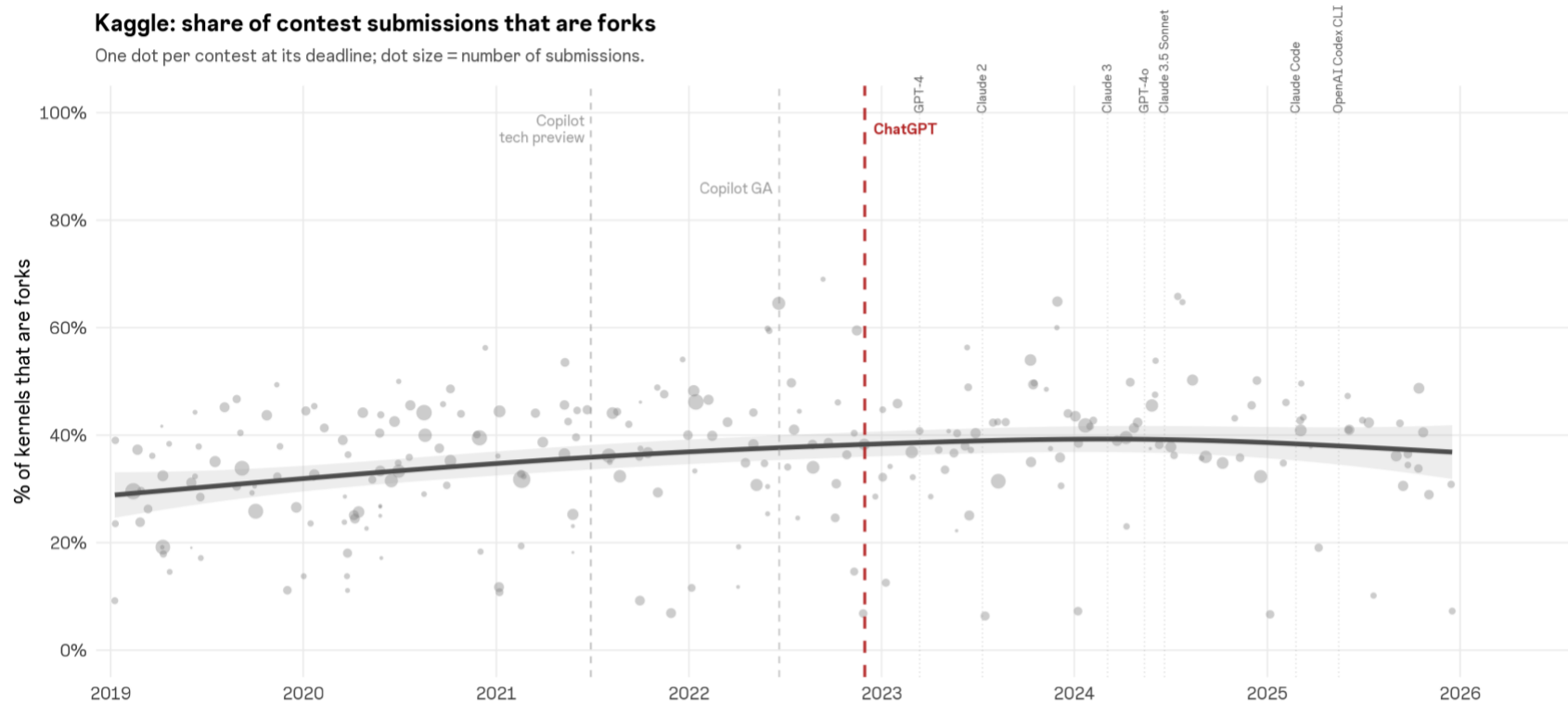


Figure B1. The percentage of Kaggle kernel submissions per contest that are forks of other kernels. The fraction rose through 2023, at which point it began to decline. This suggests that forking does not explain the recent decline in code novelty.

Appendix C: Kaggle Submission Publication Rate

An interrupted time-series regression indicates no significant change in per-contest ($n = 205$ competitions) publication rates around the release of ChatGPT. The estimated level shift is insignificant ($p = 0.89$), as is the estimated trend change ($p = 0.18$). These null results indicate that shifts in publication behavior are unlikely to explain my findings.

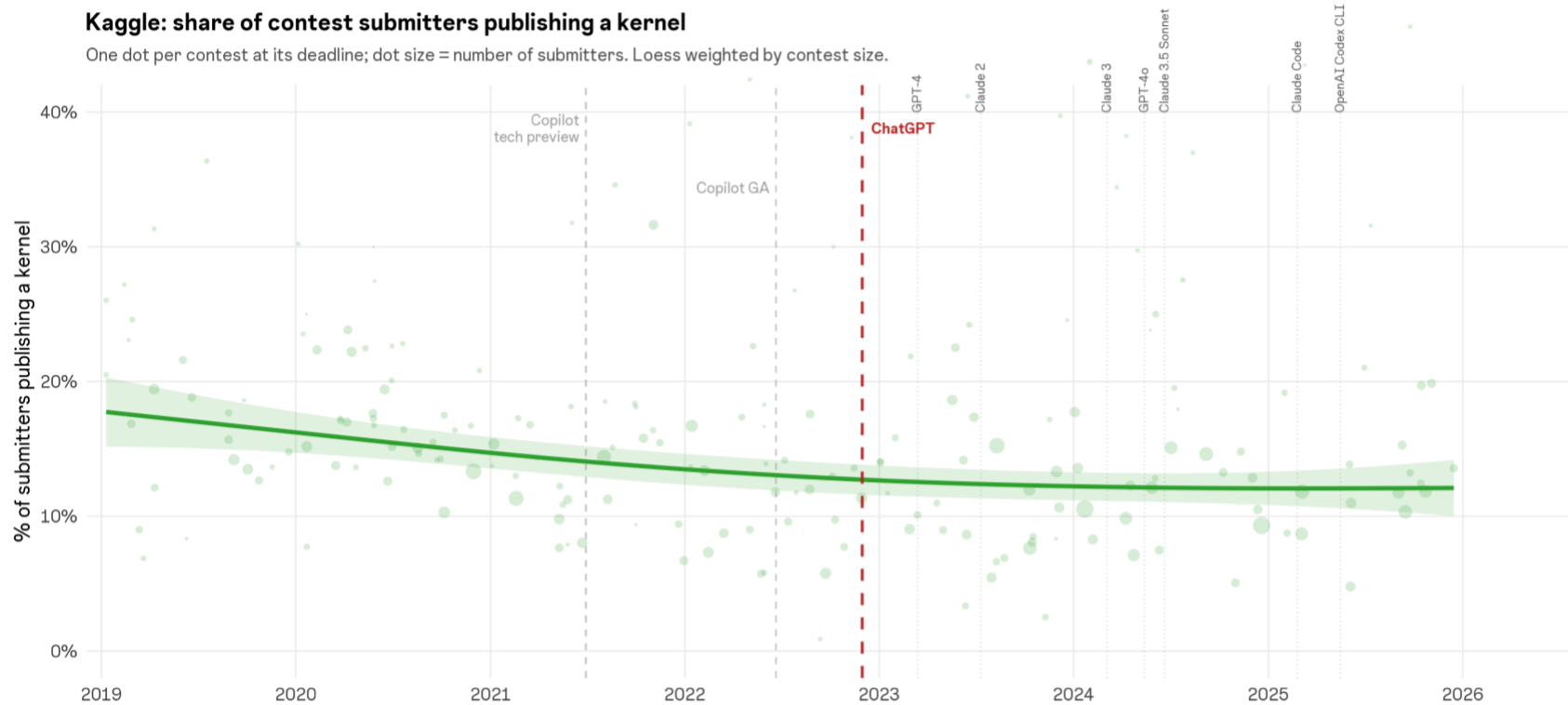


Figure C1. The percentage of Kaggle contest submitters, per contest, who publish at least one kernel. The fraction declined until mid-2022, just prior to the emergence of ChatGPT, after which the rate has remained relatively flat. This suggests that shifts in Kaggle publication activity cannot explain my findings.

Appendix D: Embedding Measure Validation

I will now validate the claim that TF-IDF n-gram embeddings operationalize syntax while Voyage code embeddings operationalize semantics, i.e., intent. I conduct this analysis via two controlled evaluations. In the first, I manipulate the objective or task that the code addresses independent of syntax and then assess the extent to which different embedding approaches yield distance measures that effectively isolate variation in each respect. In the second, I take a similar approach, but hold the code objective fixed, and vary the algorithmic approach. I will demonstrate that TF-IDF embeddings are effective at capturing syntactic variation, but insensitive to the underlying algorithm or objective. Conversely, Voyage code 3 embeddings are uniquely capable of detecting variation in code objective or algorithm, while being insensitive to superficial syntax.

Code Objective vs. Syntax

I first construct a corpus of code snippets, in a controlled manner, reflecting the cross-product of two factors. The first, which I term the code’s objective, reflects what the code seeks to do. The second, syntax, reflects the surface variable-naming style. My corpus includes five code objectives (returning the indices of two values that sum to a target value, detecting duplicate values, finding the most frequent value, identifying the intersection of two lists, and counting element pairs located at a fixed difference from one another within a data structure), where code is implemented employing one of four distinct variable and function naming conventions. The result of this 5-by-4 is 20 distinct code snippets. Variable naming conventions are elaborated upon in Table D1, and code objectives in Table D2.

Crucially, when creating these code snippets, I deliberately employ arbitrary function and variable names that reveal nothing about the code’s objective. Further, the code snippets include no comments. The code’s objective is therefore apparent only from the holistic computation the code performs, whereas shared syntax is reflected strictly by overlap in literal characters and words. The design, therefore, cleanly separates the two levels of abstraction: syntax and semantics.

Table D1. Variable-naming conventions.

Convention	Style	Example variable tokens
Terse	single lowercase letters	a, b, c, d, ...
Greek	spelled-out Greek letters	alpha, beta, gamma, delta, ...
Numeric	v-prefixed indices	v0, v1, v2, v3, ...
Metasyntactic	placeholder words	foo, bar, baz, qux, ...

Grouping snippet pairs that share only a single feature, either an objective (semantics) or naming convention (syntax), I compare the cosine-distance distributions in each embedding space. As Figure D1 shows, TF-IDF n-gram token embeddings treat code as most similar when it shares literal syntax (naming conventions), even when the objective differs, whereas Voyage code-3 embeddings treat code as most similar when it shares an objective, even when syntax differs. Notably, other embedding approaches are less capable of cleanly separating syntax from semantics (aside from Unixcoder, which performs similarly to TF-IDF).

Table D2. Code objectives.

Objective	What the code computes
Two-sum	Indices of the two values that sum to a target
Duplicate detection	Whether any value appears more than once
Mode	The most frequently occurring value
Intersection	The sorted values common to two lists
Pairs at distance k	Count of element pairs with absolute difference k

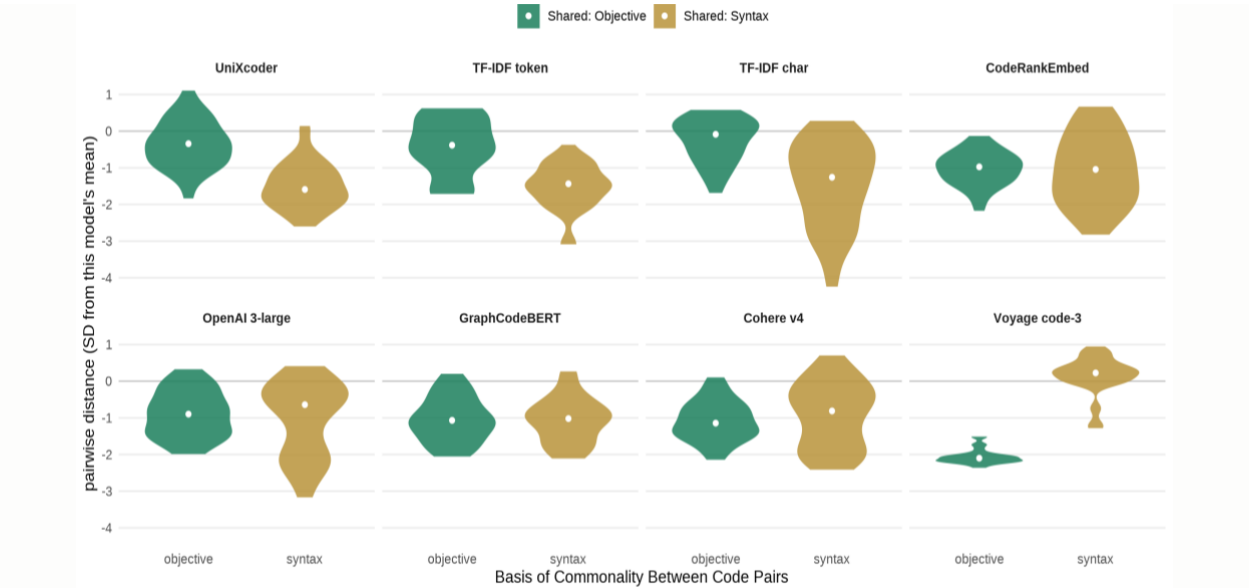


Figure D1. Evaluating which embedding approach best captures the code’s syntax versus the objective.

For each embedding approach, I plot the distribution of pairwise cosine distances between code pairs that share an objective but not a naming convention, versus those that share a naming convention but not an objective. Lower values on the y-axis indicate that an embedding approach treats the group of pairs as systematically more similar. Distances are standardized within each embedding space, so the y-axis is comparable across sub-figures. TF-IDF embeddings better capture code syntax while ignoring its objective, while Voyage code 3 embeddings better capture code objective while ignoring syntax.

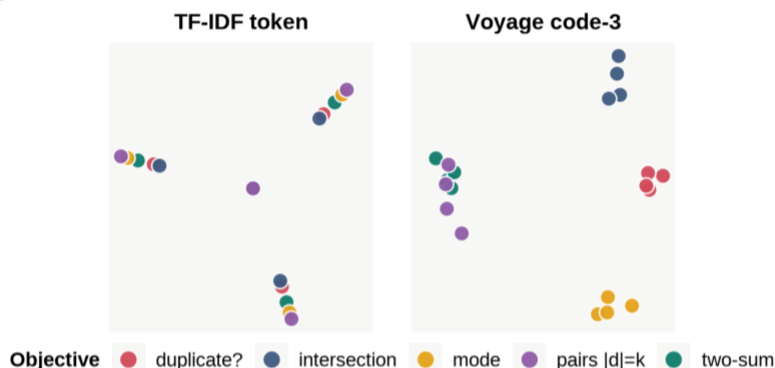
A secondary benefit of this exercise is that it also provides an opportunity to calibrate the reported cosine-distance and spectral-rank measures in the main manuscript against clearly defined instances of shared versus differing syntax and code objectives, enabling a more meaningful interpretation of the reported effects (or lack thereof). In the Voyage code space, when I deliberately shift only the variable naming convention, the impact on cosine distance is generally on the order of 0.10. By contrast, changing the code’s objective increases the Voyage cosine distance to approximately 0.5.

The latter distance closely aligns with the actual distances observed between submissions to different contests (0.54) in the real Kaggle data, confirming that the scale associated with very clear differences in solution approach in the toy example also manifests in real-world contests. Second, and more importantly, the average *within*-contest Voyage distance (0.32) is roughly 60% of the value we observe when I change the coding task entirely. Further, in some Kaggle contests, we observe cosine distances in the Voyage code space that exceed the typical cross-contest cosine

distance. Thus, submissions to a single Kaggle contest are often genuinely different in their approach to the problem at hand, spanning much of the available approach space despite sharing a common objective. This fact suggests that the stable contest-level Voyage results reported in the main text are not a byproduct of contests having overly narrow objectives or opportunities for approach exploration, nor are they the result of the Voyage code embedding measure being insensitive to shifts in code intent.

Each panel is a 2-D projection (MDS on cosine distance) of one embedding's map of the same 20 snippets. Tight same-color groups mean the space organizes code by that factor: Voyage clusters by objective, TF-IDF by naming.

Colored by objective
what the code computes



Colored by syntax
surface variable names only

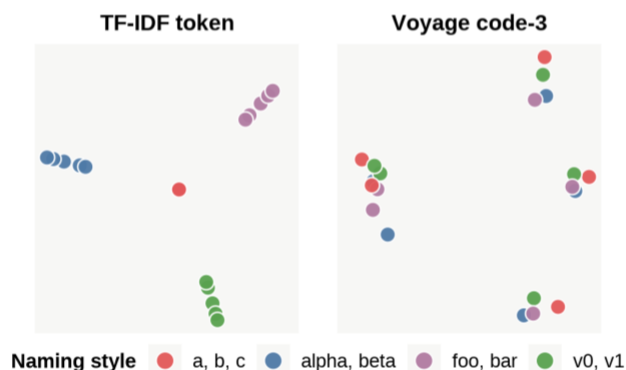


Figure D2. Comparing Embedding Clusters Generated via Each Embedding Approach. Each sub-figure presents a two-dimensional projection (multidimensional scaling on cosine distance) of an embedding's 'map' of the twenty code snippets. Each snippet is colored by objective (top row) and by naming style (bottom row). Once again, I observe that Voyage code 3 cleanly clusters snippets by code intent, whereas TF-IDF n-gram token embeddings cluster cleanly on naming convention.

Algorithm vs. Syntax

In the second evaluation, I repeat the same approach, holding the task fixed. Every generated code snippet implements a list sort. In this task, I again generate a corpus of snippets that reflect the cross-product of two factors: sorting algorithm (Table D3) and naming convention (as detailed earlier in Table D1). This second evaluation examines whether each embedding approach can effectively distinguish the approach taken to a single, fixed objective from the syntax employed. Once again, the code snippets contain no comments, and the naming conventions are ambiguous regarding the algorithmic approach. As expected, I find that Voyage code-3 is particularly effective at grouping code snippets by algorithm (algorithm-vs-syntax AUC of 0.91; the nearest neighbor identified based

on Voyage code embeddings shares the same algorithm 80% of the time). By contrast, other embedding approaches group code snippets primarily based on naming convention, particularly TF-IDF.

Table D3. Sorting algorithms considered in the embedding evaluation.

Sorting algorithm	Structure
Bubble sort	Nested loops with adjacent-element swaps
Insertion sort	Outer loop with an inner while-shift
Selection sort	Outer loop, inner minimum-finder, one swap per pass
Quicksort	Recursive in-place Lomuto partition
Mergesort	Recursive split with a linear merge

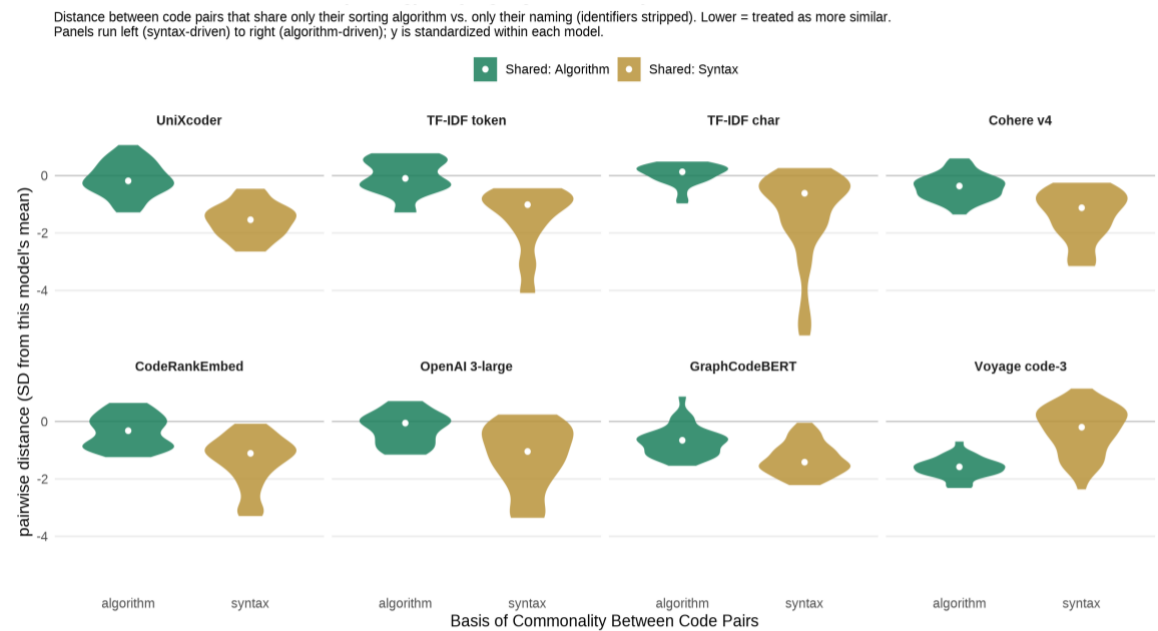
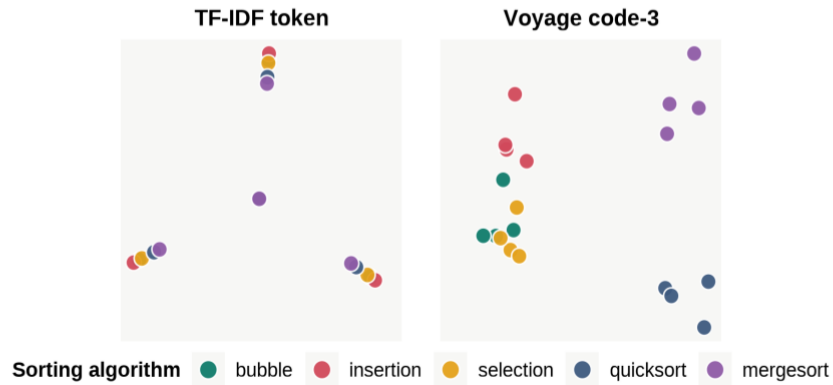


Figure D3. Distribution of Cosine Distances by Code Snippet Pair Group and Embedding Approach. Twenty Python snippets that all sort a list, spanning five sorting algorithms and four naming styles. For each embedding, the distribution of pairwise cosine distances among pairs sharing only their algorithm versus only their naming. Only Voyage code-3 treats a shared algorithm as most similar; the rest treat shared naming as most similar.

Each panel maps (MDS on cosine distance) one embedding's view of 20 sorting snippets (5 algorithms x 4 naming styles). Only Voyage groups by algorithm; TF-IDF groups by naming.

Colored by algorithm
which sorting method



Colored by syntax
surface variable names only

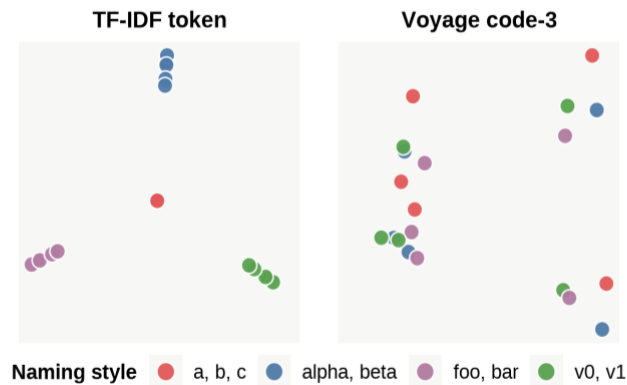


Figure D4. The Features Each Embedding Approach Clusters On: Algorithm vs. Syntax. Two-dimensional projections (Multidimensional Scaling on cosine distance) of the TF-IDF and Voyage embeddings of the same 20 sorting snippets, colored by algorithm (top) and by naming style (bottom). Voyage clusters by algorithm; TF-IDF clusters by naming.

Appendix E: Directly Measuring LLM-Usage

The main analyses do not directly operationalize the role of AI in Kaggle kernels, hence identification is largely based on intertemporal shifts in code characteristics surrounding the introduction of major AI models and coding tools. Accordingly, here, I seek to assess the role of AI coding assistants in these shifts more directly. Note that in all analyses reported elsewhere, code submissions are embedded after stripping comments to ensure that the measures specifically capture convergence in the code as implemented, rather than as described. I now return to code comments and construct a submission-level measure of AI-assisted authorship based on them.

My focus here on code comments is motivated by recent literature showing that LLM-generated code comments exhibit a very distinctive style. Ye et al. (2026) study 5,869 code-generation scenarios drawn from World of Code and LeetCode and find that, although LLM-generated code is broadly comparable to human-written code in aggregate readability, it displays “distinct readability issue patterns,” specifically regarding low-information comments and redundant comments. Ji et al. (2026) analyze active company- and community-maintained repositories from 2021 to 2025 using detector-based proxies for LLM-generated code and comments, finding that comments classified as likely LLM-generated are concentrated in “Explanation” and “Meta” categories, a fact that remains relatively stable over time even as detected LLM-generated code declines. Moreover, those authors find that LLM-generated comments tend to exhibit a relatively lower proportion of grammatically correct sentences. Those authors also motivate the study by noting developers’ anecdotal concerns that LLM-generated comments are often unnatural and fail to match the tone and clarity of human-written comments. Finally, Kang et al. (2024) show that the problem is not limited to style or redundancy: in an evaluation of comments generated by three LLMs, even the best-performing model produced comments containing demonstrably inaccurate statements roughly one-fifth of the time. Taken together, these findings suggest that LLMs may leave a recognizable commenting footprint in generated code, characterized by explanatory or meta-level comments that can be redundant, low-information, unnatural, grammatically weak, or factually inconsistent with the code they describe.

I begin by collecting a random sample of Kaggle kernels submitted prior to ChatGPT’s release in November of 2022, where the code comments are highly unlikely to have been generated by an LLM (only early instantiations of GitHub Copilot were in use toward the end of that period, and only by a minority of coders). This serves as my ‘purely human’ sample. These submissions span 248 contests. Next, taking the same contests, I prompt four different LLMs (Claude Haiku 4.5, Claude Sonnet 4.6, Gemini 2.5 Flash, and GPT-4o) to generate one-shot solutions to each, providing each LLM with the contest title, overview, and dataset description. Finally, I extract all code comments from each human- and LLM-generated code solution, obtaining character-level TF-IDF n-gram representations (3-, 4-, and 5-grams) of each, which I then use to train a logistic regression-based classifier of LLM vs. human authorship. Employing five-fold cross-validation, where folds are sampled at the contest level, i.e., no contest contributes to both training and evaluation folds in any given pass, yields a mean AUROC of 0.996, i.e., for pre-ChatGPT contests, comment text can provide an extremely strong signal of AI comment authorship.

I then validate the classifier on a set of 10 pre-2022 competitions that were withheld from training. Figure E1 plots the score distributions for the human kernels obtained from those competitions and for the matched LLM-generated solutions. The average classifier score for human kernels is

0.14, with only 1.7% exceeding a 0.5 threshold. By contrast, LLM-generated solutions average scores between 0.78 and 0.83, with 87% to 96% (depending on the LLM) exceeding the 0.5 threshold. Detection performance is also comparable across all four LLMs. Hence, again, I have good reason to believe that this LLM-generated comment detector has discriminatory power. That said, it is important to note that, in practice, humans working with LLM coding assistants may not adopt LLM-generated comments verbatim. Moreover, newer model releases may exhibit different commenting patterns, and more recent Kaggle contests may also address topics and provide context that meaningfully shift the distribution of both human and LLM comments. Accordingly, and provided that false positives among genuinely human post-2022 kernels remain rare, the classifier-generated scores are likely to be conservative, a lower-bound proxy for AI usage. Further, I view the LLM-comment detector as likely having high precision but possibly poor recall, implying its output might be viewed as providing a lower bound on AI use.

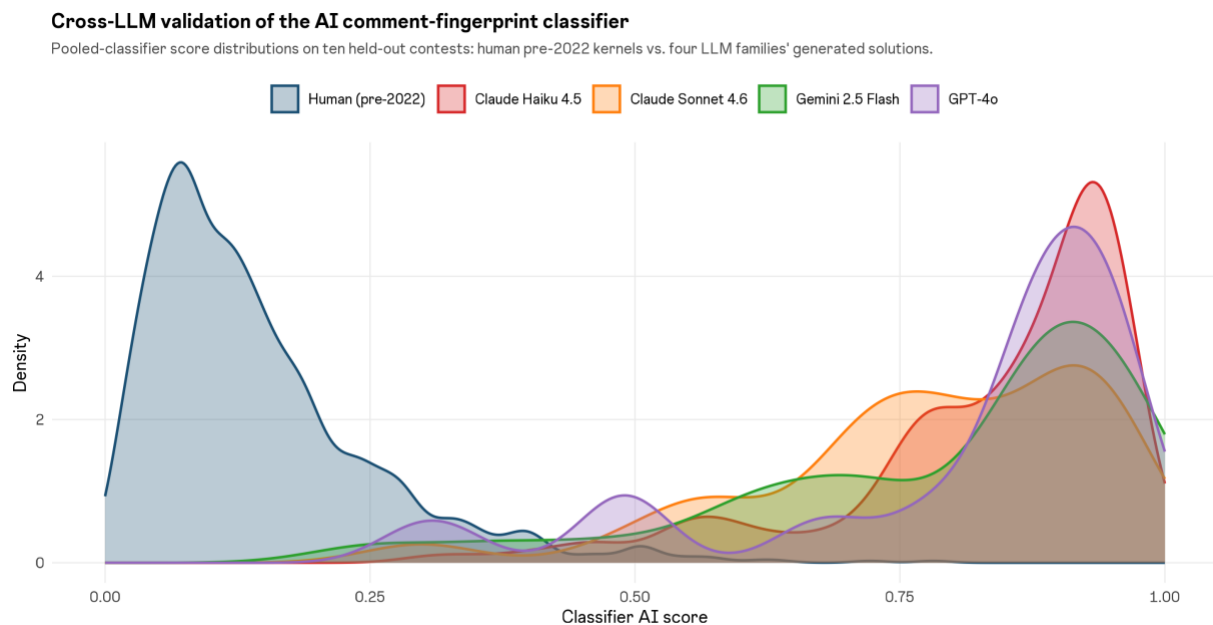


Figure E1. Cross-LLM validation of the AI comment classifier on ten held-out Kaggle competitions. Densities of classifier scores for pre-2022 human kernels versus code solutions generated by four LLMs (Claude Haiku 4.5, Claude Sonnet 4.6, Gemini 2.5 Flash, GPT-4o). Human kernels concentrate near zero, while LLM-generated solutions concentrate near one.

With that context in mind, I apply the comment scorer to my entire Kaggle corpus. Figure E2 depicts the share of contest submissions in my broader Kaggle sample that have an LLM-authorship score of 0.50 or higher, over time. In the pre-AI period, roughly 1% of kernels exceeded the threshold, and the proportion is generally flat through late 2022, when ChatGPT was released. After that point, the share of submissions labeled as including LLM-generated comments rises rapidly, reaching approximately 17% by mid-2026. An interrupted time-series (ITS) regression indicates that the post-release trend change is +0.37 percentage points per month ($p < 0.001$). This descriptive result is notably well aligned with the observed rise in the use of seed 42 and in the timing of shifts in code homogenization measures.

Kaggle: monthly share of contest kernels labeled as containing LLM-generated comments

Share with pooled-classifier AI score ≥ 0.50 (panel 95th percentile).

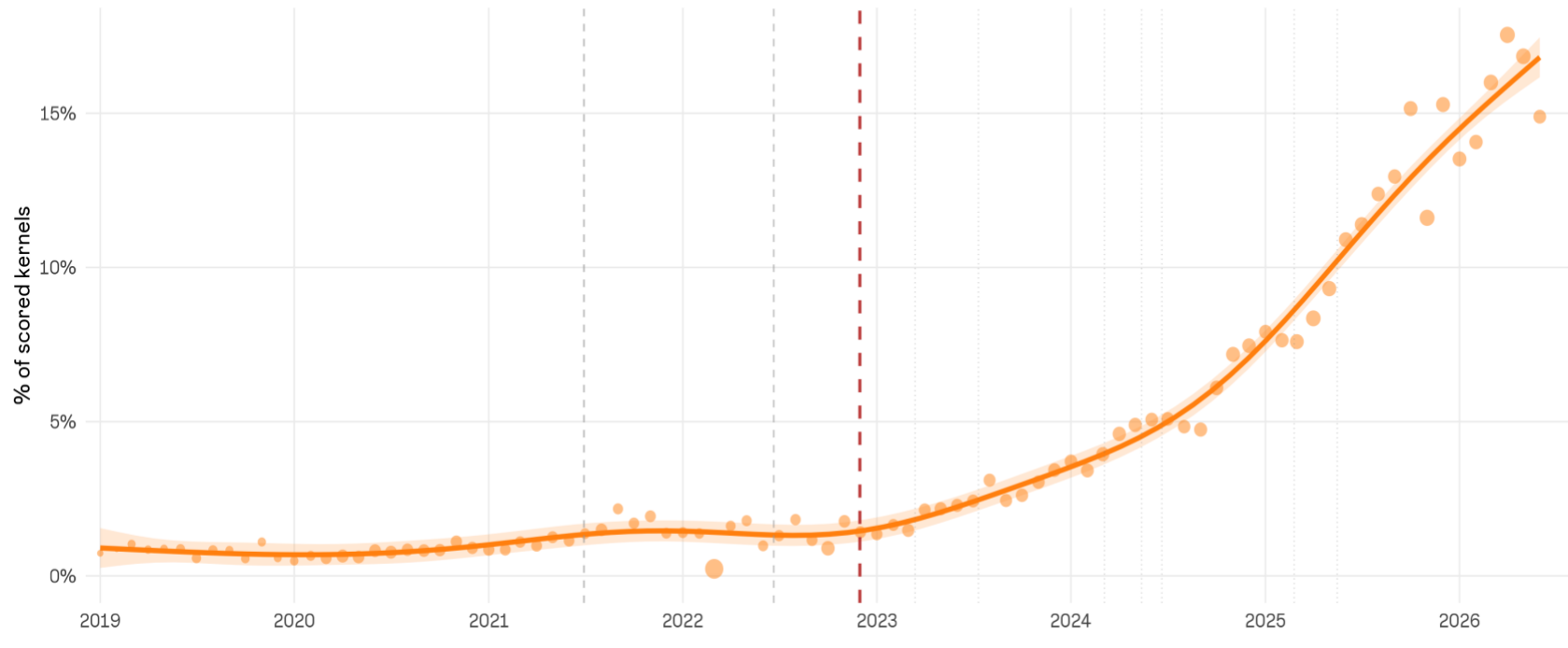


Figure E2. The percentage of scored Kaggle contest kernels labeled as containing LLM-generated comments, over time. I threshold the raw classifier score at 0.50 (the 95th percentile of the raw scores). The share of submissions predicted to be LLM-generated is roughly 1% and flat prior to ChatGPT's release. The value then rises steadily thereafter, reaching approximately 17% by mid-2026.

At the level of individual submissions, heavier AI usage associates positively with seed 42 usage. Table E3 reports a variety of estimates from linear-probability models that regress an indicator for the presence of seed = 42 on the kernel-level LLM-comment score, conditioning on author and contest fixed effects, with standard errors clustered by contest.

Table E3. Kernel-level association between AI comment score and seed-42 usage.

Specification	Fixed effects	Coefficient (SE)	N
(1) All scored kernels	Author	+0.275*** (0.043)	74,742
(2) Seeded kernels	Author	+0.178*** (0.036)	30,954
(3) Seeded kernels	Author + contest	+0.115** (0.036)	30,954
(4) Seeded kernels (robustness)	Author + contest + month	+0.100** (0.035)	30,954

Notes: Linear-probability estimates regressing an indicator for seed = 42 usage on the AI-comment score, with standard errors clustered by contest in parentheses. The estimate in row 1 is based on all scored kernels, so its dependent variable reflects seed-42 use versus some other seed or no seed at all; in rows 2, 3, and 4, I restrict the sample to kernels that set some seed. Fixed effects are as indicated in each column. * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

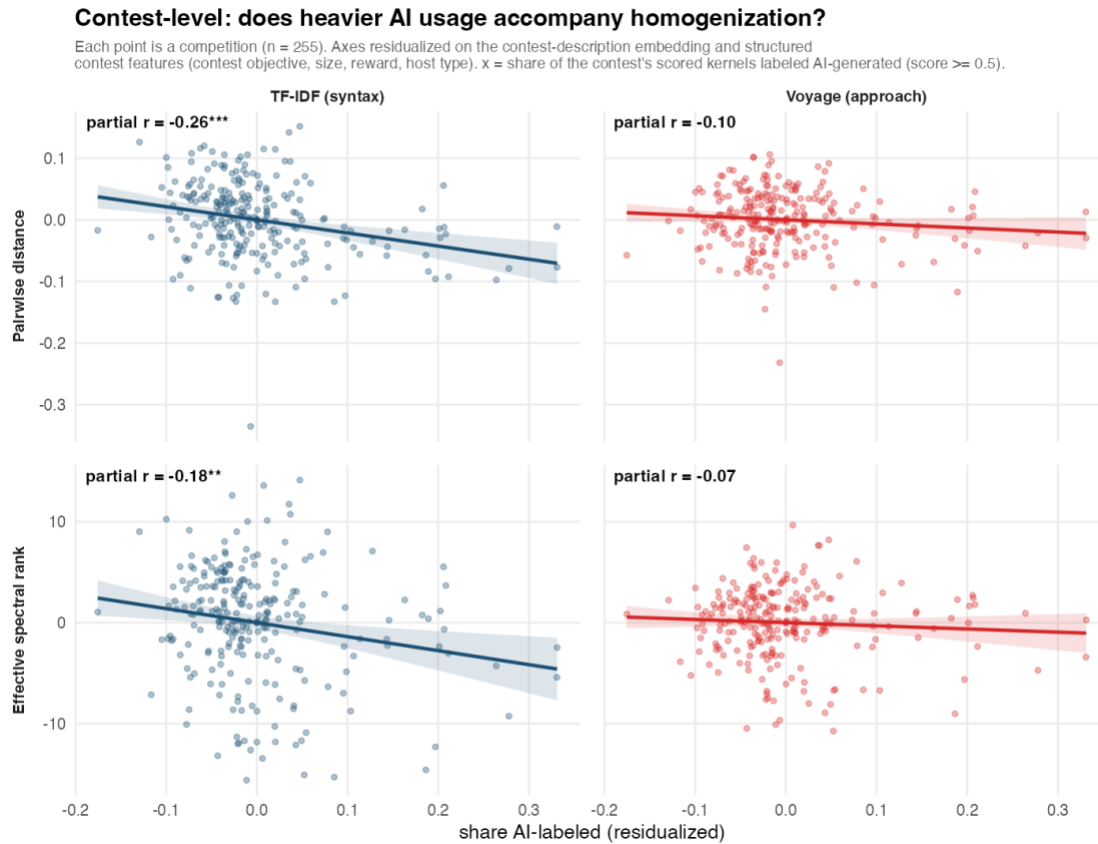


Figure E3. Contest-level association between LLM-comment score and code homogenization. Each point is a competition, residualized with respect to contest-description embedding and structured contest features (objective, size, reward, and host type). The x-axis is the share of a contest's kernels labeled as containing LLM-generated comments (classifier score ≥ 0.50). Heavier LLM-usage is associated with lower within-contest pairwise distance and lower effective spectral rank in the TF-IDF (syntactic) space but has no significant association with either measure in the Voyage (semantic) space.

Across all scored kernels, a one-unit increase in the AI-comment score is associated with a 27.5-percentage-point increase in the probability of seed-42 use (column 1). Among kernels that set some seed, the association remains positive and strongly significant, at 17.8 points with author fixed effects (column 2) and 11.5 points once contest fixed effects are added (column 3). Adding month fixed effects leaves this estimate essentially unchanged (10.0 points, column R), which indicates that the relationship is not merely a shared secular trend.

Finally, I associate the AI comment score at the contest level with my measures of average pairwise code homogenization and contest-level effective spectral rank (where, again, the embedding-based measures are constructed from contest submissions with comments stripped beforehand). The association between AI use and TF-IDF pairwise submission distance is broadly negative, as is the association with TF-IDF effective spectral rank (Figure E3). The corresponding Voyage associations are small and statistically insignificant, consistent with the flat pairwise trend and stable (if anything, marginally expanding) effect rank I document in the main analyses. Thus, I find that heavier AI usage is associated with code homogenization across different levels of abstraction, mirroring the findings reported in the main manuscript and further suggesting that AI is indeed a key mechanism underlying the results.

Appendix F: Sensitivity of Results to Embedding Choice

Two concerns motivate additional checks: that the syntactic convergence is specific to the token-level TF-IDF representation, and that the semantic null reflects an insensitive embedding rather than a real feature of the data. I address both by re-estimating the interrupted-time-series specification of Table 1 on the same competitions using alternative embedding representations, and by probing the dynamic range and anisotropy of the semantic space.

Table F1 reports the post-release trend change (the Time $\times$ Post-ChatGPT coefficient) for each representation. On the syntactic side, I replace the token-level unigram/bigram TF-IDF with character n-gram TF-IDF (character 3–5-grams), an alternative surface-form representation. On the semantic side, I replace Voyage code-3 with an alternative semantic retrieval embedding, Cohere Embed v4. The two syntactic representations indicate similar, statistically significant contraction in both pairwise distance and effective rank, whereas the semantic representations exhibit no significant pairwise contraction or decline in effective rank, and even mild evidence of a rise in rank.

Table F1. Post-release trend change.

Representation	Embedding Type	Pairwise distance	Effective rank
TF-IDF, token 1–2 gram	syntactic (main)	-0.0020*** (0.0006)	-0.157*** (0.040)
TF-IDF, character 3–5 gram	syntactic	-0.0033*** (0.0008)	-0.133*** (0.031)
Cohere Embed v4	semantic	+0.0005 (0.0007)	+0.023 (0.030)
Voyage code-3	semantic (main)	-0.0004 (0.0007)	+0.052† (0.030)

Notes: Estimates reflect the Time $\times$ post-ChatGPT coefficient from the interrupted-time-series specification of Table 1. HC3 standard errors in parentheses. Outcomes are within-contest mean pairwise cosine distance and effective spectral rank, operationalized using four different embedding representations (252 contests with $N \geq 20$ for pairwise distance, and 205 contests with $N \geq 50$ for effective rank). † $p < 0.10$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

Once again, it should be noted that the null result in semantic space does not appear to be an artifact of a compressed embedding. The Voyage space’s cross-contest standard deviation of pairwise distance (0.075) is nearly identical to TF-IDF’s (0.079), and, within a contest, its distances span nearly the full cosine range (1st-to-99th percentile of 0.11 to 1.07). Yet, expressed in units of each space’s own cross-contest variation, the post-release change is 1.09 standard deviations for TF-IDF pairwise distance but only 0.21 for Voyage (0.90 versus 0.44 for effective rank). For measures with equivalent range, the semantic shift is roughly one-fifth of the syntactic shift and is not distinguishable from zero.

The null is also robust to addressing embedding anisotropy. Contextual embeddings often occupy a narrow cone dominated by a few shared directions (Mu and Viswanath 2018; Ethayarajh 2019), thereby compressing cosine distances. Voyage exhibits this common-direction anisotropy in raw form (a mean cosine of 0.46 between random submission pairs), but the anisotropy is removed by the mean-centering I apply to all embedding measures. This is demonstrated by the fact that no single dimension dominates the centered embeddings; the top principal component explains just 6.7% of the variance in the centered embeddings. Projecting out the top one, two, and three global principal components (the all-but-the-top postprocessing suggested by Mu and Viswanath 2018) has no impact on my conclusions about the lack of shifts in Voyage embeddings’ distances or

collective span; the pairwise-distance change remains within ± 0.0006 per month, and the effective-rank change weakens toward zero. Declines in TF-IDF cosine distance and spectral rank also remain when applying the same anisotropy-removal procedure to those embeddings.

These robustness checks indicate that syntactic homogenization is not specific to a particular embedded representation and that the semantic null is not an artifact of embedding insensitivity or anisotropy. Further, the toy validation exercise presented in Appendix D demonstrates that the Voyage embeddings are effective at detecting shifts in solution approach.