Birth, Life, and Death of AI Models*

Aaron P. Kaye† Boston University & MIT
Kazimier Smith MIT
Neil Thompson MIT

August 4, 2026

Please do not circulate, cite, or share with third parties.

Abstract

We characterize the lifecycles of open-weight AI models using a weekly panel of the 108,074 most-downloaded models on Hugging Face, which together account for 99.5 percent of the platform's more than 53 billion downloads. We link these data to token usage from OpenRouter and performance rankings from Arena and Artificial Analysis. We establish key institutional details about the supply of and demand for AI models, and we organize our analysis into three phases: birth, life, and death. In the "birth" phase, we show that major developers often release models in batches, concurrently introducing models that are both vertically differentiated (e.g., by size, capability, and latency) and horizontally differentiated (e.g., by task specialization, hardware compatibility, and alignment). Because weights are open, developers can also build on one another's models. We document cumulative innovation through fine-tuned, quantized, merged and adapted descendants of source models. In the "life" phase, we quantify how demand evolves after release. Even within the sample of top models, demand is concentrated, and the concentration persists: the median model loses ∼68 percent of its release-week downloads within ten weeks, while the most popular models decline gradually; as a result, the top percentile's advantage over the median model is ∼600-fold at release, and ∼900-fold six months later. Further, we estimate spillover effects from new popular model releases on related models and find that the first popular third-party descendant is associated with a 50 to 80 percent increase in weekly downloads of its source model. In the "death" phase, we measure usage obsolescence and technical obsolescence and find that demand often persists long after a model falls behind the technical frontier. Our findings characterize open-weight AI as a market of complementary model portfolios rather than a sequence of vintage replacements: one in which third-party developers building on a source model can raise demand for the original, and in which demand persists long after technical obsolescence.

Keywords: Artificial Intelligence, Open Source, Open Weights, Technology Diffusion, Platform Economics, Cumulative Innovation.
JEL: O33, L17, L86, O31

*First version: August 4, 2026. We thank Isaiah Andrews, seminar participants at MIT FutureTech, and conference participants at CESifo for helpful feedback and suggestions. This paper uses publicly available data and is not subject to a data-sharing agreement or editorial review from Hugging Face, OpenRouter, Arena, Artificial Analysis, or any other platform. We also thank all researchers at Hugging Face who helped make this research possible, including Avijit Ghosh, Yacine Jernite, and Giada Pistilli. All analyses, conclusions, and any remaining errors are our own.
†Contact: apkaye@bu.edu.

1 Introduction

Artificial intelligence is increasingly the subject of debates about productivity, competition, energy use, safety, and industrial policy, yet the market's basic institutional details remain largely undocumented. In mature industries, researchers typically know what the product is, how firms version it, how they introduce new vintages, and when old vintages effectively exit. Although much of the policy discussion treats the AI market as one that consists of a few firms competing to develop ever-larger frontier models, these basic questions about the structure of the AI market remain open. The open-weight market is broad, economically relevant, and the subject of policy discussion.[1] Hugging Face hosts more than three million open-weight models, with over 53 billion cumulative downloads. On OpenRouter, the largest marketplace between inference providers and users, open-weight models account for nearly 30% of token use (Aubakirova et al., 2026).[2]

Training a model from scratch is expensive, but developers can reuse nearly the same underlying training investment to produce a family of related models, which permits cost synergies across a firm's portfolio of models. Such portfolios can also be expanded: once a model's weights are released, they are relatively cheap to copy and modify, allowing both the original developer and third parties to create descendants through fine-tuning, quantizing, distilling, adapting, and merging. Understanding technological change in this market requires evidence on how models enter, how developers use and build on them after release, and when models effectively exit. To fill this gap, we establish basic institutional details about the open-weight AI market and empirically document the lifecycle of open-weight AI models using novel data from Hugging Face, OpenRouter, Arena, and Artificial Analysis. We organize our analysis into three phases: birth (the entry and differentiation of new models), life (the evolution of use and cumulative innovation after release), and death (the process of obsolescence and persistence).

In markets where the relevant products, decisions, and competitive margins are not yet well understood, establishing key institutional details can be critical in shaping research that furthers our understanding of the new market, and better informs policy. Economics research has a tradition that highlights the importance of establishing institutional details. Griliches (1957) documented how hybrid corn spread across U.S. states at different rates, establishing the empirical patterns that shaped decades of further work on technology adoption. Gort and Klepper (1982) traced the entry and exit of firms across product markets, identifying the lifecycle that became a benchmark for theories of industry evolution. Dunne et al. (1988) measured the basic patterns of firm turnover in manufacturing, providing the facts that motivated dynamic models of industry structure. Lerner and Tirole (2002) laid out the institutional logic of open-source software, clarifying the incentive structures that standard industrial organization frameworks had not yet addressed. De Loecker et al. (2020) and Conlon et al. (2023) documented the evolution of firm markups across the U.S. economy, providing empirical evidence to inform policy debates on inflation. In each example, correctly establishing institutional details enabled further research and informed policy. We aim to do something similar for AI by establishing descriptive institutional details and providing empirical analysis to characterize how models enter, compete, diffuse, and persist.

We study these questions in the context of open-weight models. This is an especially appealing setting to understand the AI market, as it offers an unusually visible window into a broader AI market that is otherwise difficult to observe. Open-weight models are models where the weights are publicly available even when

[1] See Letter on "Open Weights and American AI Leadership," organized by Nvidia and Microsoft, July 24, 2026; Anthropic's "Position on open-weight models," July 27, 2026; and Thinking Machines' "A Safe Path to Open Weights," July 31, 2026.

[2] Using open-weight models requires either self-hosting or using an inference provider, for example through a marketplace like OpenRouter.

their full training code or data are not.[3] A few platforms act as intermediaries between model developers, compute providers, and users. These publicly provide detailed information on the models and their demand. Additionally, there is a growing market of platforms dedicated to publicly benchmarking model capabilities. Our primary data come from Hugging Face, the largest platform for hosting and distributing open-weight models. We supplement these data with usage data from OpenRouter, a platform that connects users to compute providers for a variety of open-weight and proprietary models, and with performance rankings from Arena and Artificial Analysis. Together, these sources let us observe supply-side product-market choices and demand-side adoption patterns. On the supply side, developers choose what to release, how many versions to release together, and how to differentiate those versions. Further, on the supply side, we explicitly observe cumulative innovation: a source model can be fine-tuned, quantized, adapted, or merged with other models to create a new, potentially competing, descendant model.[4] On the demand side, we observe attention and adoption over time, which we can use to document the model lifecycles and how adoption changes after the release of related, competing, or more capable models. Our main measures of model demand trade off between coverage and directness. Hugging Face provides broad measurement of an upstream margin of open-weight model demand, while OpenRouter directly measures downstream inference for a narrower part of the market. We provide an econometric framework under which within-model changes in downloads identify proportional, and consequently unitless, changes in latent model demand, without requiring a download to represent the same amount of use across models.

Following our birth-life-death framework, we provide a set of stylized facts and empirical analysis of how open-weight AI models enter, evolve, and exit the market. In the "birth" phase, we find that major developers frequently release models in batches, introducing families of models that are vertically differentiated by size, capability, and latency, and horizontally differentiated by domain specialization, hardware compatibility, and alignment. This implies that the relevant unit of product design is often the model family rather than serial versions of individual models. Further, we find that third-parties play a substantial role in expanding these model families. From our sample of popular models, 30,191 are descendants (27.9%), meaning model entry is shaped by both product-line design and cumulative innovation.[5]

In the "life" phase, we show that hype and demand follow different patterns: Attention peaks quickly, but downloads persist much longer. Across models, demand is concentrated in a subset of ultra popular models. Of the roughly 3 million models on Hugging Face, the top 10% account for 99.8% of downloads, the top 1% account for 98.8%, and the top 0.1%, just 2,922 models, account for 93.3% of downloads. Within a model, demand is concentrated near release, with the median model losing 68% of its release-week downloads within ten weeks.[6] Further, we investigate demand spillovers between source models and their descendants. Theoretically, the effect of a popular descendant is ambiguous: it could cannibalize demand for the source model by offering an upgraded alternative, or it could boost demand by drawing attention to the source in an otherwise crowded market. We find strong evidence of the latter: the release of a first popular third-party descendant is associated with a 50 to 80 percent increase in weekly downloads of its source model. However, for subsequent releases, we find generally negative, but statistically insignificant, spillovers, meaning we cannot definitively rule out either force as the number of descendant models grows.

[3]When a model is open-weight, anyone with the appropriate hardware can deploy it. By contrast, using closed models requires a compute provider with access to the model.

[4]Notably, since the models are open-weight, third party developers can build off of another developer's work.

[5]We consider 27.9% a lower bound, as developers self-report source-descendant relationships, including quantizing, fine-tuning, merging, and adapting, but not model distillation.

[6]The most popular models decline more gradually, so the top percentile's advantage over the median grows from roughly 600-fold at release to 900-fold six months later.

In the "death" phase, we define usage obsolescence as the point at which a model is delisted or no longer attracts meaningful token usage on OpenRouter, and technical obsolescence as the point at which the model loses competitive standing on benchmark or preference-based rankings. Among models that reach each threshold, the median age is 9 weeks at technical obsolescence and 36 weeks at usage obsolescence. We evaluate the impact of reaching technical obsolescence with an event study design, finding that demand often persists long after a model falls behind the technical frontier. By contrast, we find that demand for downloading the model drops significantly after usage fades on Open Router (usage obsolescence). A number of mechanisms might explain these results. For example, open-weight models could be embedded in larger processes like software packages. Users and firms might face switching costs that prevent them from updating their AI workflows at the same frequency as technological improvements. We find that usage and technical obsolescence have different timings and different demand responses, consistent with the switching cost explanation.

Both the institutional details and the empirical results we establish are valuable for AI researchers and policymakers, whose assumptions about market features shape their research conclusions. For example, consider the assumptions underlying these three questions: first, what is the value of the data used to train AI models? Second, how do open weight models affect competition and innovation in the AI market? Third, what does model adoption reveal about demand for improvements in AI capability? On the value of data, assuming firms train economically relevant models one at a time implicitly treats each model as the outcome of a separate training decision. The batch releases and descendants we document show that this model-by-model assumption ignores a training investment's potential synergies across models in the same and subsequent batches, as well as across developers through descendant models. On how open weights affect competition, it is tempting to assume that open-weight models face fierce competition from third-party descendant models, ignoring the possibility of positive spillovers. On technical progress and adoption, evaluating models only on benchmark performance ignores potential lock-in, whereby demand persists long after technical obsolescence. Similar assumptions underpin other important questions, like market definitions in antitrust analysis and safety proposals for older models. The empirical facts we document can inform specific research questions about AI and advance our understanding of the broader AI market.

1.1 Related Literature

This paper relates to three strands of the literature. First, a growing economics and policy literature studies foundation models as a potentially general-purpose technology and asks how openness, complementary assets, and market structure shape diffusion. Second, work in computer science examines open-weight ecosystems as settings of reuse, fine-tuning, and interdependent supply chains. Third, an nascent empirical literature measures pricing and demand in emerging markets for model usage.

The first strand emphasizes that the economic significance of AI depends not only on frontier technical progress, but also on downstream invention, organizational change, and market structure. Bresnahan (2024) argues that AI becomes a general-purpose technology only through complementary co-invention and digital transformation, rather than through frontier model improvements alone. Azoulay et al. (2024) show that openness need not ensure entry when incumbents control complementary assets, while Korinek and Vipra (2024) highlight how scale economies, scope economies, and vertical integration can push foundation-model markets toward concentration. Similarly, Kapoor et al. (2024) argues that open foundation models have properties that create both innovation benefits and governance risks. We contribute to this literature by providing new measurement of how open-weight models actually enter and compete: major releases often

arrive as portfolios of vertically and horizontally differentiated models, suggesting that, for questions related to competition and innovation, the relevant unit of analysis is a firm's portfolio of models, rather than single model releases.

The second strand studies open-weight AI as an ecosystem of reuse and cumulative innovation. Jiang et al. (2023) document that reuse of pretrained models on Hugging Face depends on attributes such as provenance, reproducibility, and portability, but also faces problems of missing metadata, performance discrepancies, and model risk. Dettmers et al. (2023) show that QLoRA sharply lowers the resource cost of adapting large models, making downstream fine-tuning feasible on much more modest hardware. Laufer et al. (2025) adopt an approach from evolutionary biology to document family trees of AI models on Hugging Face. We connect to this literature by adding an economic lifecycle perspective: descendant creation, inherited lineage, and downstream adaptation shape how models create value after release.

The third strand of the literature centers on demand for AI models. Demirer et al. (2025) document rapid growth in models, creators, and providers, along with price declines, persistent price heterogeneity, and both horizontal and vertical differentiation in demand. Fradkin (2025) shows that, on OpenRouter, new model releases are adopted rapidly, but differ substantially in whether they substitute for incumbents or expand the market, and that multihoming across models is common. We build on this literature by making two contributions to the measurement of AI model demand: new data and a complementary econometric framework. On data, we are the first, to our knowledge, to construct a panel of model-week level downloads using publicly available data from Hugging Face. We use publicly reported 30-day rolling and cumulative totals to reconstruct implied weekly downloads for each model.[7] Further, we show that weekly download data are especially useful for answering questions about demand and the impacts of related model releases. Previous papers have relied on 30-day rolling downloads, which mask the precise timing of changes in demand around events of interest and make it especially difficult to understand shorter model lifecycles. This granularity also allows us to demonstrate that "likes," a common proxy for adoption in prior studies, and downloads follow considerably different patterns.[8] On the methods side, we provide an econometric framework and a modest set of assumptions that formally map our constructed weekly download measure to overall changes in market demand.[9]

Taken together, these strands suggest that AI should be analyzed as an evolving ecosystem rather than as a sequence of isolated model launches. Our contribution is to provide a single measurement framework that makes visible family structure, spillover dynamics, and persistence in open-weight models.

1.2 Outline

The remainder of the paper is organized as follows. Section 2 describes the institutional setting, data sources, and sample construction. Section 3 examines model entry and differentiation, including batch releases, model families, and descendant models. Section 4 documents how attention and demand evolve after release and estimates spillovers from subsequent same-author releases and third-party descendants. Section 5 distinguishes usage from technical obsolescence and studies how demand and attention change around each form of obsolescence. Finally, Section 6 summarizes the findings and discusses their implications for AI policy

[7] In other recent work, Longpre et al. (2025) examine changes in model concentration over time. They also use weekly download data. However, they use internal Hugging Face data from an earlier period, June 2020 to August 2025.

[8] Kadasi et al. (2025) notes the limitation of the raw publicly available download data, and instead proposes using likes as a measure of adoption.

[9] The primary assumption is that any event of interest can change demand, but not the relationship between downloads and demand. We call this assumption modest, as it still allows for the relationship between demand and downloads to vary across models, across time, and across model age.

and future research.

2 Data and Setting

Our primary data source is Hugging Face, the largest online platform for hosting machine learning models.[10] Before the rapid rise of large language models, Hugging Face offered an interface through which developers could use popular machine learning methods. As transformer-based models began to dominate the machine learning ecosystem, Hugging Face became a general platform for storing and accessing AI models. Today Hugging Face hosts more than three million models. Because of its popularity and ease of use, it is one of the most complete records of open-weight models available (Laufer et al., 2025). The key qualifier is *open-weight*. Hugging Face makes each model available to any developer that wants to use it by storing the model "weights," a set of numbers learned during training that the model uses to convert input to output. Developers download these weights in a specific format (typically this is done in a straightforward way through the Hugging Face API). Developers can then use the model on their local hardware. Models whose weights are publicly available are called *open-weight*. The term differs from the more common term *open-source*, which refers to software whose *code* is publicly available. Critically, the code used to train and develop an open-weight model is not necessarily made public. Only the core model architecture need be available for developers to use the model. For example, Meta's Llama models are open-weight but not open-source. In contrast, models like OpenAI's GPT-5 are *proprietary*: their weights are not publicly available and users must interface with the model through OpenAI-approved channels. Even the exact parameter count of proprietary models is often a closely guarded secret (Wan et al., 2025). Hugging Face is an excellent environment to study the lifecycle of AI models because of its scale. We have sufficient data to reliably estimate lifecycle patterns that would be difficult to infer from a small sample of proprietary models.

We retrieve data on all models on Hugging Face from the "Model Hub."[11] The model hub provides daily snapshots of every model, along with additional metadata, going back to July 24th, 2024. There are 587 snapshots in total. This is the primary source of our raw data. Each snapshot contains a list of models and many metadata fields. We construct a weekly panel of Hugging Face models by reading one snapshot every seven days starting on July 24th, 2024. While there is a wealth of metadata available, a few key fields are documented in Table 2.1.

Each model has a unique identifier and a creation date recording when the developer first uploaded its weights. Downloads record requests to Hugging Face's servers for specific model files. We observe two download measures: a rolling 30-day count available throughout our sample and a cumulative count available only from February 2025 onward; Section 2.1 describes how we impute cumulative downloads for earlier periods. Likes function similarly to social media engagement: users click a button to express interest, and the platform records a running total. Users are not required to download or use a model before liking it. Finally, Hugging Face assigns each model a trending score that determines its visibility on the platform's main page.[12] Internal conversations with Hugging Face employees indicate that the score incorporates downloads, likes, recency, and other factors, but the exact algorithm is not public. We observe each of these variables on a weekly basis.

[10] huggingface.co
[11] https://huggingface.co/datasets/cfahlgren1/hub-stats
[12] https://huggingface.co/models

Table 2.1: Key variables from Hugging Face Model Hub

Variable	Description
ID	Unique identifier for the model. Consistent over time.
Creation date	Date the developer uploaded model weights to Hugging Face.
Downloads (rolling)	Number of times the model was downloaded in the last 30 days.
Downloads (cumulative)	Total number of downloads since creation. Only available starting February 2025.
Likes	Total number of user "likes" accumulated since creation. Users click a button to express interest; no download or usage is required.
Trending score	Platform-calculated popularity score incorporating downloads, likes, recency, and other factors. Determines model ranking on the Hugging Face models page. Exact algorithm is not publicly available.

Note: Data retrieved from weekly snapshots of the Hugging Face Model Hub, July 2024 to February 2026. See Section 2 for details.

2.1 Model downloads

We use model downloads on Hugging Face as our main outcome of interest. Hugging Face records downloads based on requests to their servers for designated files in a model repository.[13] We think of downloads as an upstream measure of demand because downloading a model generally precedes using the model for inference. Our data include every model on Hugging Face, so we observe all downloads recorded on the largest platform for hosting and distributing open-weight models. This gives us an especially comprehensive view of upstream demand for most of the world.[14] By the end of our sample, models on Hugging Face had been downloaded more than 53 billion times.

The first challenge related to using download data is practical: Hugging Face does not directly report weekly downloads, so we construct a weekly panel from the available information. Hugging Face reports rolling 30-day downloads throughout our sample and cumulative downloads beginning in February 2025. For our analysis, the main problem is that the rolling measure obfuscates the timing of downloads. We use both the rolling and cumulative counts to construct the implied weekly downloads for each model from July 24, 2024 to February 11, 2026. We detail this process in Section A.1. Our approach addresses limitations of the publicly available download measures noted in prior work. [15] We similarly construct weekly likes from the reported cumulative totals.

The second challenge is conceptual: a download is not necessarily a direct measure of downstream use or demand in general. The relationship between model use and downloads may vary across models, users, and inference providers. For example, a user might download a small model one time locally and then use the model for inference repeatedly. Alternatively, a user repeatedly deploying the model with third-party compute might use a container that downloads the same model at the start of each run.[16] We address this measurement issue by focusing much of our analysis on proportional differences in downloads rather than

[13] For details on how Hugging Face counts downloads, see https://huggingface.co/docs/hub/en/models-download-stats.

[14] One important exception is mainland China, where Hugging Face is inaccessible without a VPN and ModelScope is a major domestic alternative.

[15] For example, Osborne et al. (2024) note the limitations of the reported download counts and find that likes are more strongly correlated than downloads with model usage in Hugging Face Spaces.

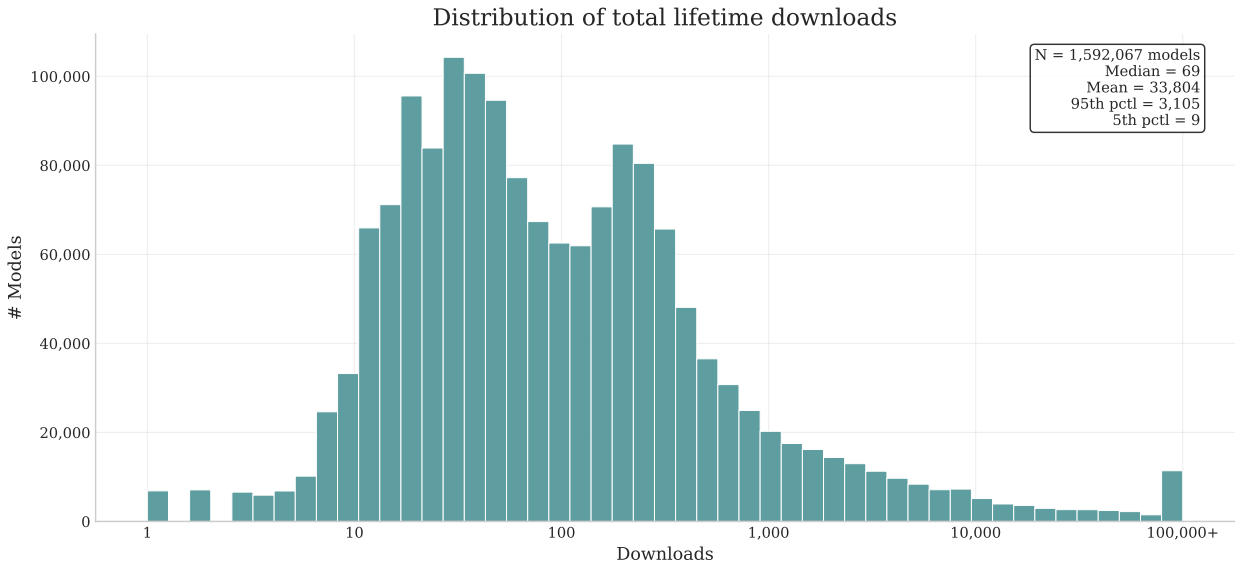
[16] AI models can also be dependencies in software packages, meaning they are downloaded whenever a user installs the package.

their raw levels. In Section 4.2.1, we detail a modest set of conditions under which our estimates can be more broadly interpreted as changes in overall model demand.[17]

2.2 Economically relevant models

Hugging Face acts as a repository for open-weight models much like Youtube acts as a repository for videos. Anyone, including firms, professional developers, academics, and hobbyists, can upload a model to Hugging Face, and the model remains stored on the platform unless its creator removes it. This means that there are many, in fact 3,054,279, models on Hugging Face, yet many of these models see little to no downloads. We aim to restrict our sample to the *economically relevant* models that offer insight into how developers actually use models. Including models that are never used could obscure the true nature of the model lifecycle. We calculate the total downloads each model receives over our sample and drop models that are never downloaded. Of the 3,054,279 models in our data, 1,462,212 are never downloaded, and we exclude these from the sample. Even among models that are downloaded, the vast majority see little activity. Figure 2.1 plots the distribution of cumulative lifetime downloads across all models that receive at least one download. A small fraction of models see huge numbers of downloads. DeepSeek-R1, for example, has over 14 million since its launch in January 2025. Less popular models exhibit a bimodal distribution, but the bulk of models receive fewer than 1,000 downloads in their lifetimes. The median model on Hugging Face, among those that are downloaded,

Figure 2.1: Distribution of total downloads



Note: Plots the distribution of maximum cumulative downloads observed for each model. Excludes 1,462,212 models that are never downloaded.

receives only 69 downloads. We aim to characterize model lifecycles from July 2024 to the present, and the median model receives approximately one download per week during this period. This small number of downloads could obscure more interesting lifecycle patterns among the models that most developers actually use. Of the more than one million models that are ever downloaded, 108,074 models account for 99.5% of all

[17]We also compare downloads with token usage on OpenRouter in Figure A.1. While token usage is a direct measure of downstream inference, OpenRouter data only covers a small share of global inference.

downloads on Hugging Face (Figure A.2). For the remainder of the paper we restrict our analysis to these economically relevant models.[18]

2.3 Inference data

To supplement our data on open-weight models, we collect data from *OpenRouter*, one of the largest intermediary platforms through which users can access AI models.[19] OpenRouter aggregates access to more than 300 models, including both open-weight and proprietary models, through an API. When a user sends a prompt, OpenRouter routes the request to a *provider* that hosts the model and runs inference on its own hardware. Some models have multiple providers; in these cases OpenRouter typically routes requests to the cheapest available option.

OpenRouter is useful for our purposes because it records direct usage in the form of tokens processed.[20] While Hugging Face downloads capture how often developers retrieve model weights, OpenRouter token counts capture how intensively a model is actually used for inference. This distinction matters: a model may be downloaded once and used extensively, or downloaded many times by automated pipelines without meaningful use. Token counts provide a complementary measure of real demand that is positively correlated with downloads (Figure A.1). We observe token counts starting in November 2024, and Figure 2.2 plots total usage over time for all models on OpenRouter (including proprietary models) and for the models we are able to match to Hugging Face. Models in our dataset account for 27.5% of total input and output tokens on the platform, emphasizing the scale and economic significance of the open-weight AI market.

2.4 Model performance data

We also seek a more technologically-grounded measure of how each model's capabilities stack up against its competitors. We focus on two metrics: *benchmarks* and *Elo scores*. Benchmarks offer a clearly defined evaluation rubric that is easy to compare across models, while Elo scores are directly constructed from the preferences of real users.[21] These quality measures are critical because they allow us to separate the roles of intrinsic capability and market dynamics in shaping a model's lifecycle. Two models may exhibit very different download trajectories despite similar quality, or similar trajectories despite different quality levels. Below we detail the data sources for each metric.

2.4.1 Benchmarks

Benchmarks provide a set of questions or tasks with clearly defined answers. Models are evaluated by the number of questions they answer correctly. For example, the Google-Proof Q&A Benchmark contains multiple-choice questions about physics, biology, and chemistry and tries to measure a model's ability to do genuine scientific reasoning, rather than searching the web (Rein et al., 2023). We collect benchmark results from *Artificial Analysis*, an independent firm that has tracked model performance since 2023.[22] For each of more than 700 models, Artificial Analysis collects price, latency, and scores on a variety of standard

[18] Even the 108,074th most popular model has over 2,000 lifetime downloads. To ensure our estimates are robust to the choice of cutoff, we include initial popularity bin controls throughout much of our analysis and repeat the estimation for more popular subsamples.

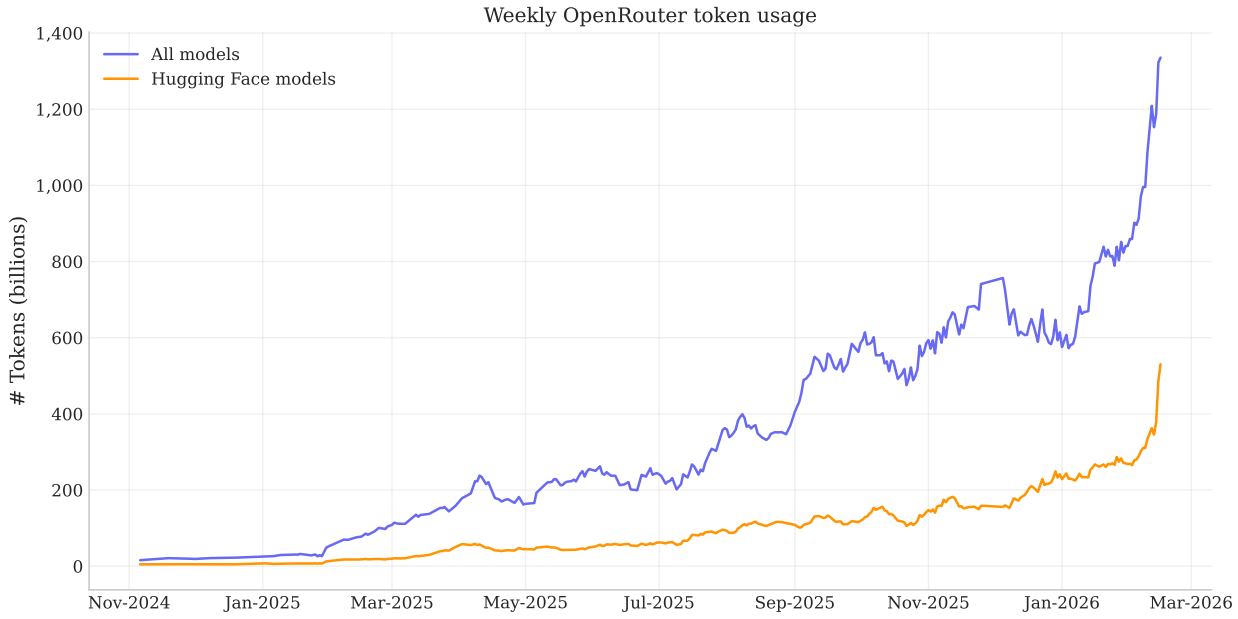
[19] openrouter.ai

[20] Tokens are the fundamental unit of data on which models operate; user input is converted to tokens, and the model outputs tokens which are formatted to produce the final response

[21] Arena uses the Bradley-Terry model for mapping head-to-head preferences into model ranking scores. For details, see "Arena AI FAQ".

[22] artificialanalysis.ai

Figure 2.2: OpenRouter model usage



Note: Plots daily sum of prompt and completion tokens. Values on missing dates are filled in with linear interpolation.

benchmarks that test coding, math, tool use, and related tasks. These data are aggregated into an *Intelligence Index* for each model and more specific Coding, Math, and Agentic indices. We collect these index scores as a measure of technical quality. Table 2.2 shows the top models in each index.

Table 2.2: Artificial Analysis Benchmark Indices

	All models			Hugging Face models		
Category	Models	Highest score	Best model	Models	Highest score	Best model
Intelligence index	594	59.9	Claude Fable 5	338	51.1	GLM-5.2
Coding index	533	78.3	GPT-5.6 Sol	305	68.8	GLM-5.2
Math index	369	99.0	GPT-5.2	220	96.7	DeepSeek V3.2 Speciale
Agentic index	413	54.0	GPT-5.6 Sol	238	43.1	GLM-5.2

Note: Data as of July 9, 2026.

2.4.2 Elo scores

Benchmarks sometimes lack real-world applicability because developers optimize models for high benchmarks scores or because all models score near-perfectly (saturation). We address these issues by collecting model "Elo scores", which measure how well a model performs in pairwise comparisons with competitor models. We collect Elo scores from Artificial Analysis and from *Arena*, one of the most widely used platforms for evaluating large language models through direct human judgment.[23] To calculate the scores, both platforms host "model arenas" in which a user compares the outputs of two models given a single prompt and chooses

[23] arena.ai. Previously named LMArena and Chatbot Arena.

their preferred result.[24] Figure A.4 depicts a typical contest: given a text prompt, two models generate images, and the user does not observe the identity of either model. The user then selects the image they prefer. Thousands of users make these pairwise comparisons, and the platform aggregates the resulting votes into the Elo score for each model. The evaluations come from real users, so they directly capture each model's quality as perceived by a broad user base.

The Elo score data complements the benchmark data by capturing a different dimension of quality. Benchmarks evaluate models on structured tasks with well-defined criteria, while Elo scores reflect subjective human preferences in open-ended interactions. A model may score well on standardized benchmarks but perform less impressively in freeform conversation, or vice versa; indeed, discussion around major model releases often highlights disappointing performance relative to officially published benchmark scores. Including both measures allows us to assess whether the lifecycle patterns we observe are robust to different notions of model quality.

Arena maintains leaderboards across categories including overall performance, coding, mathematics, and instruction following. Artificial Analysis leaderboards focus on media models, including image generation, video generation, and text-to-speech. Table 2.3 shows the number of models in each category and the number we are able to match to our main Hugging Face sample. Note that both platforms include proprietary models in their arenas which are not present on Hugging Face. With detailed data on model usage and quality in hand, we turn to analyzing the model lifecycle.

Table 2.3: Elo scores

	All models			Hugging Face models		
Category	Models	Highest score	Best model	Models	Highest score	Best model
Artificial Analysis						
Text to image	156	1339.0	GPT Image 2	41	1226.2	Cosmos3-Super-Text2Image
Image editing	71	1291.4	Riverflow 2.0	24	1230.0	HunyuanImage 3.0 Instruct
Text to video	84	1218.8	Dreamina Seedance 2.0 720p	12	973.5	LTX-2.3 Fast
Image to video	80	1194.6	Dreamina Seedance 2.0 720p	9	953.7	LTX-2.3 Fast
Arena.ai						
Text	376	1563.6	Claude Fable 5	124	1487.8	GLM 4.7
Vision	136	1366.8	Claude Opus 4.6	24	1211.1	Qwen3 VL Thinking
Search	31	1238.5	Claude Fable 5	0	–	–
Document	29	1500.1	Claude Fable 5	0	–	–
Webdev	92	1653.9	Claude Fable 5	6	1440.1	GLM 4.7
Text to image	73	1362.8	GPT Image 2	8	1167.9	Hunyuan Image 3.0
Image edit	53	1455.8	GPT Image 2	7	1240.6	Qwen Image Edit
Text to video	41	1527.5	Gemini Omni Flash	1	1170.0	Hunyuan Video 1.5
Image to video	42	1473.7	Dreamina Seedance 2.0	1	1196.1	Hunyuan Video 1.5
Video edit	7	1377.1	Dreamina Seedance 2.0	0	–	–

Note: Data as of July 8, 2026.

[24] On Artificial Analysis, the platform pre-selects the prompt and the user only chooses their preferred output. On Arena, the user inputs their own prompt. Both choices could bias the Elo scores, so we combine the data from both platforms.

3 Birth

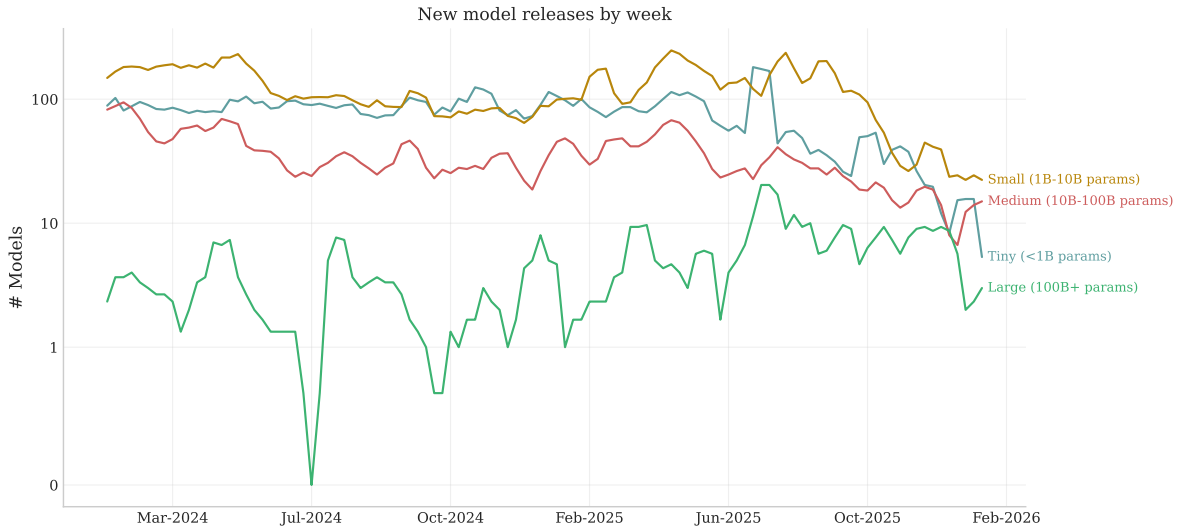
New model releases sometimes disrupt the entire AI market, as with the release of DeepSeek-R1 in January 2025. Further, understanding the context around model releases provides a baseline to which we can compare later sections of the model lifecycle. We therefore begin by studying how and when models are released.

3.1 Model Entry and Model Sizes

We first analyze the entry of new models. Compute costs and other significant fixed costs constrain the supply of AI models. Optimally training a 100 billion parameter model incurs compute costs orders of magnitude higher than a 100 million parameter model (Kaplan et al., 2020; Hoffmann et al., 2022). At the same time, this scaling of compute and size is one of the primary approaches to improving model capability (Brown et al., 2020; Wei et al., 2022; Owen, 2024; Mertens et al., 2026), making model size an important measure of vertical differentiation. Our data show a split in the release patterns by model size:

- **High-volume small models:** Small model releases (<10B parameters) exceed 100 per week, driven by lower entry barriers for smaller developers.
- **Few large models:** Large model releases (>100B parameters) remain rare (1–10 per week), though activity accelerated in late 2025, perhaps due to new capacity and entrants.

Figure 3.1: New model releases by week



Note: new models are models uploaded to Hugging Face for the first time in a given week. Plots 3-week rolling average.

Despite the limitations on model development, the pace of new model releases remained relatively stable over the past year in all size classes. Smaller models with under 100 billion parameters remain dominant, although the numbers of uploads in each category began to converge in late 2025. The size breakdown of model releases reiterates the skewed nature of Hugging Face, where developers create many new small models but only a few large models each week.

3.2 Differentiation Strategies

Large developers often release multiple models simultaneously or in close succession. For example, OpenAI launched gpt-oss-120b and gpt-oss-20b on the same day, and IBM released base, speech-focused, and vision-focused versions of its Granite 3.3 model in the same month. These batches exhibit two dimensions of product differentiation across models:

1. **Vertical Differentiation:** Variation in capability and parameter counts. OpenAI's gpt-oss-120b and gpt-oss-20b are vertically differentiated because the 20 billion parameter model can tackle the same tasks as the 120 billion parameter version, but in general it will perform worse because it has fewer parameters. As with many vertically differentiated goods, some users might prefer the lower quality product due to differences in user-costs.

2. **Horizontal Differentiation:** Variation in alignment, hardware compatibility, and domain-specific capabilities. IBM's Granite 3.3 Speech (2B parameter version) and Granite 3.3 Vision (2B parameter version) are horizontally differentiated. While they derive from the same base model and share the same training data, they are fine-tuned to specialize in different tasks (speech-to-text versus image classification, in this case).

Figure 3.2: Model releases over time



To further illustrate the frequency and types of differentiation we observe, Figure 3.2 plots all open-weight model releases on Hugging Face from four major developers. DeepSeek, Meta, and IBM, frequently release *batches* of five or more models. While OpenAI's frontier models are closed, it maintains the popular "Whisper" text-to-speech models and open-weight GPT models. The typical batch from these developers includes two to four model variants that differ by parameter count and, within each parameter count, two or three variants differentiated by task specialization (e.g. language and vision). Size differentiation ensures the model appeals to users with different preferences for performance versus cost and with varying hardware limitations. Task differentiation provides specialized models for specific needs. The individual model variants within a batch seem almost always to be released in quick succession, and the pace of development is rapid: Llama 3.1 came out less than six months after Llama 3. The patterns we describe suggest major developers invest and strategize at the portfolio level rather than at the individual model level. Training a new model from scratch yields multiple new products that can be released in the same batch, which alters the return to training and may affect the long-term strategies of developers.

One way developers produce batches of models is by building on top of a specific model, which we call a *source* model. This commonly involves starting with a "base" model, the fundamental next-token-predictor version, and fine-tuning it with reinforcement learning to produce an "instruct" version more suitable for answering user prompts. Developers can also use their base model to fine-tune smaller models that are more accessible to users with limited hardware. DeepSeek, for example, used synthetic data from its powerful DeepSeek-R1 to fine-tune several versions of Llama and Qwen. We refer to these variations on the source model as *descendants*. Creating descendants is not trivial, but it is far less costly than training a large model from scratch, so the process allows developers to turn a single training run into multiple products, increasing the return on their initial training investment. There are additional methods available to build one model using another, which we detail in Section 3.4.

We have described two broad patterns in model releases. First, models are often released not as standalone products but in groups we call *batches*. Second, the batch release phenomenon partly derives from developers' ability to build on top of models to create variants called *descendants*. While these are only descriptive examples, the relative uniformity of the patterns across the four major developers suggests they apply more generally to model releases. We now turn to characterizing model releases broadly.

3.3 Batch releases

To more systematically analyze batch releases, we assign all models in our sample to batches using the following procedure:

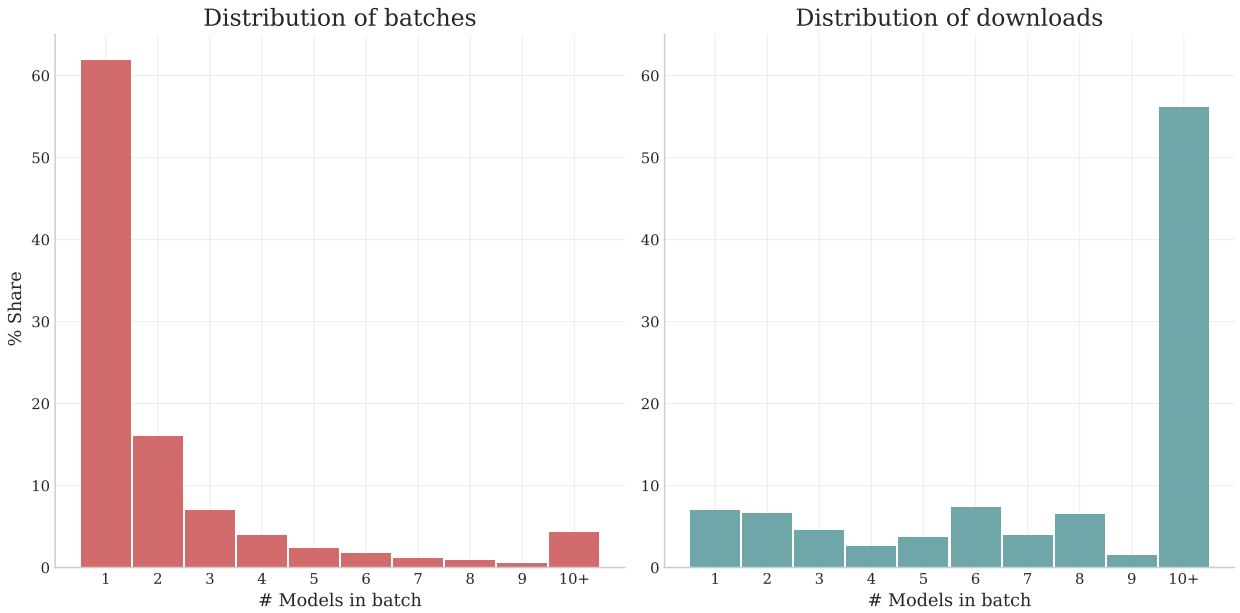
1. Find the model with the earliest release date t_0 and assign it to batch 0
2. Assign all models with a release date in $[t_0, t_0 + 30]$ to batch 0.
3. Among unassigned models, assign the one with the earliest release date t_1 to batch 1.
4. Assign all models with a release date in $[t_1, t_1 + 30]$ to batch 1.
5. Repeat the above process until all models are assigned to batches.

Training new models, as noted above, is time- and compute-intensive. The time needed to train a model from scratch makes it unlikely that two large models released within 30 days are distinct models pre-trained separately. Instead, they are likely different iterations or versions of the same training run, like with the

models fine-tuned using different methods described above. More than 30% of batches have a mean pairwise distance of one day (Appendix Figure B.2), confirming that batches typically include models released close together.

Figure 3.3 plots the distribution of batch size (the number of models in a batch) and of model downloads accruing to models in batches of different sizes. Models on Hugging Face are clearly bifurcated: the typical model is released on its own, i.e. not in a batch, and therefore cannot be vertically or horizontally differentiated. On the other hand, most downloads on the platform go to models released in batches. The contrast between the

Figure 3.3: New model releases by batch size



Note: Batches are groups of models released by the same author within 30 days of each other.
Left: share of models in batches of each size. Right: share of total platform downloads accruing to models in each batch size.

two distributions illustrates a recurring theme of concentrated model use. Even in our sample of economically relevant models, the typical model is released alone, but most downloads go to a small number of models that were released as part of major batches (like the many variants of DeepSeek-R1 released simultaneously). This suggests a release strategy similar to other technological goods like the iPhone: users decide they will buy an iPhone at the next release. On the release date, several variants are announced and users choose which one to buy. It seems plausible that users on Hugging Face choose to use models following a similar two-step process. We now turn to analyzing how these variants are created and how they interact with one another.

3.4 Descendant models

Because the core architecture of an open-weight model is publicly available, developers can build new models on top of the original, improving its performance or adapting it for specific tasks. We refer to the original model as the "source model" and to the adapted version as the "descendant model". Both the original author of the source model and third-party developers can create descendants. For example, OpenAI released their popular text-to-speech model "Whisper Large V3" in late 2023. Other developers subsequently adapted the English-focused source model for other languages, with descendants continuing to appear well into 2025.

Technologically, there are four key ways a developer can build a descendant model:

1. Quantized: reduce the size of the source model by reducing numerical precision in the underlying architecture.
2. Fine-tuned: use additional data to adapt the source model to a specific setting.
3. Merged: combine multiple source models into one descendant.
4. Adapted: adjust the hardware compatibility or add specific capabilities to the source model.

Figure 3.4: Descendant model types

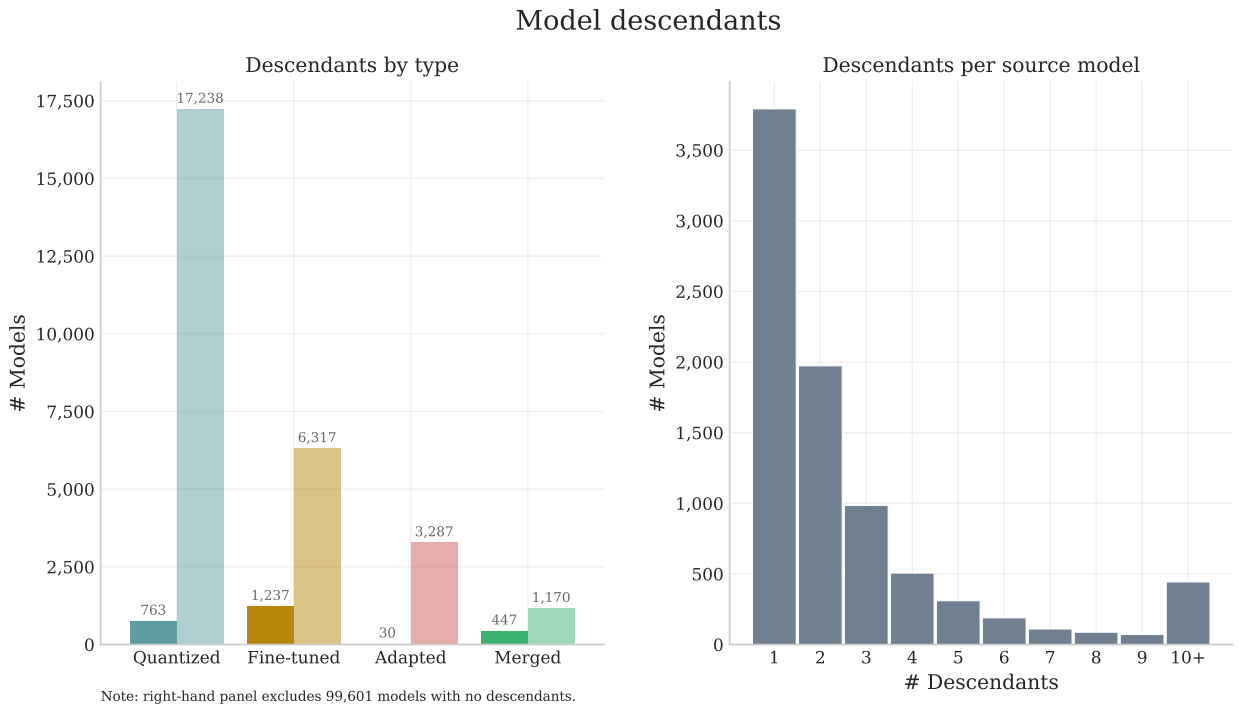
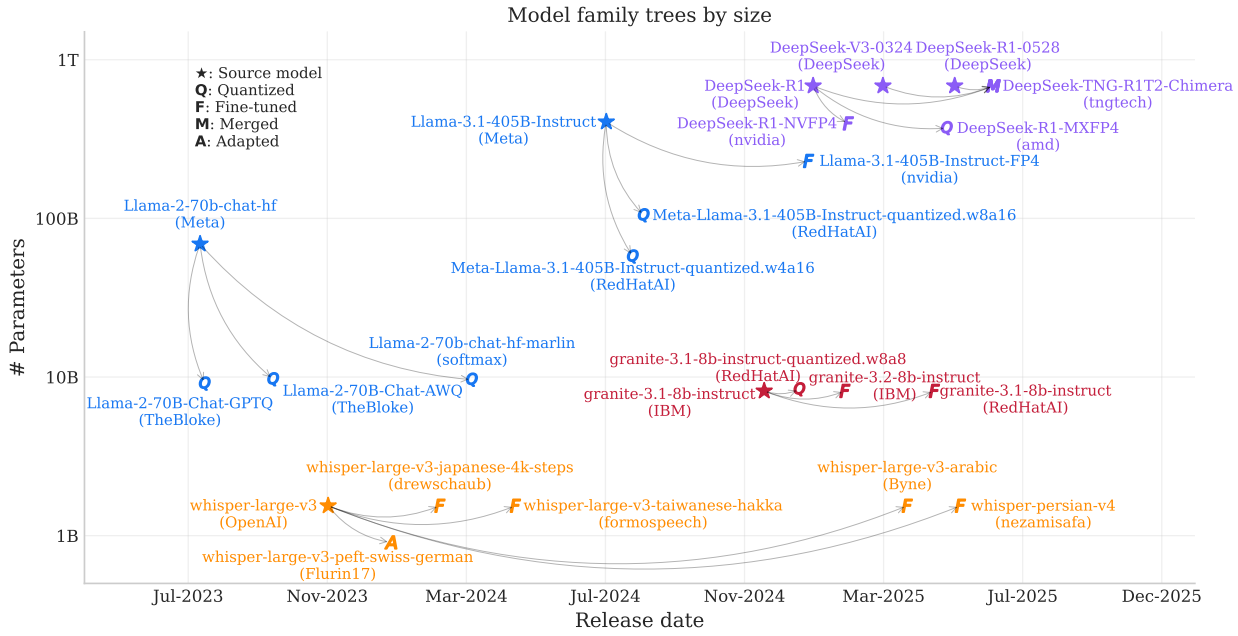


Figure 3.4 characterizes the types and number of descendants on Hugging Face. Third-party descendants are more common than same-author descendants, which reflects the large, active open-weight model ecosystem. Quantization is by far the most common method for creating descendant models. Many source models are large and powerful but require expensive hardware to run, so there is substantial demand for smaller versions that consumers and small businesses could feasibly use. Only 8,473 unique models in our data (7.83%) ever have a descendant, suggesting an ecosystem where developers repeatedly build on a small subset of source models. When models do have descendants, they rarely have more than a few, so descendant model releases are potentially economically significant events, and we analyze their effects in Section 4.2.

Figure 3.5 depicts a selection of popular source models and a subset of their descendants. Meta's early Llama models saw frequent quantization to reduce their large size, while IBM fine-tuned their own Granite 3.1 model to produce version 3.2. Expanding the number and capabilities of models in this way increases the variety of products from which users can choose, reduces their cost (in the case of quantization), and improves their performance on specific tasks (in the case of fine-tuning). The fact that the source models

Figure 3.5: Model family trees



Note: Does not show all descendants.
Quantizing reduces the size of the source models by reducing numerical precision.
Fine-tuning adjusts the source model to perform well on specific tasks.
Merging combines multiple source models into one.
Adapting adds specific functionalities to the source models.

are open-weight, therefore, almost certainly benefits consumers. On the supply side, though, it is less clear how descendant models affect the source model developer. To answer that and other questions, we turn to analyzing the lifecycle of a model after its release.

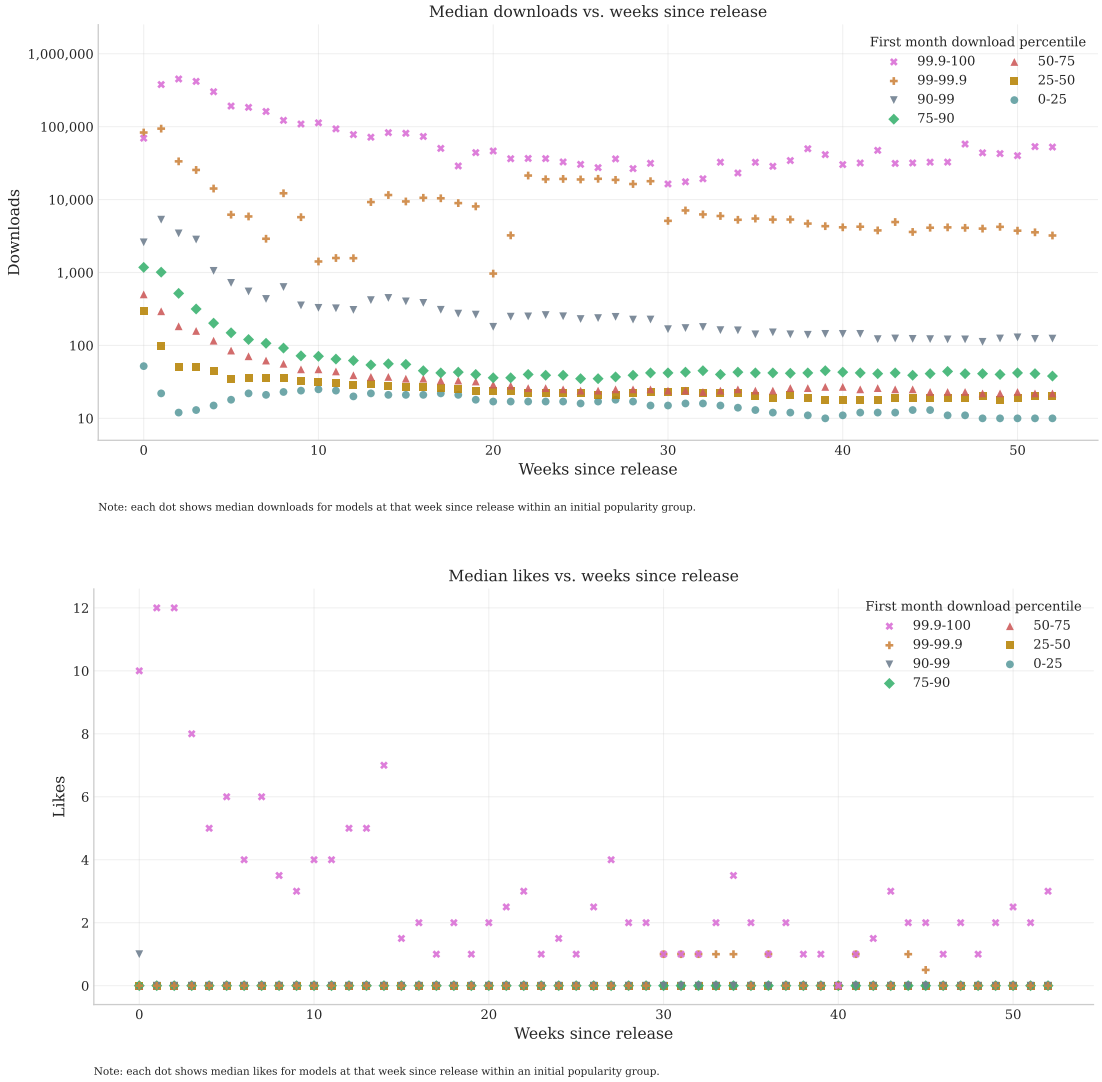
4 Life

Once released, models undergo a distinct usage lifecycle. We distinguish between *attention* (measured by "likes") and *demand* (measured by downloads). Our goal is to quantify, across all models in our sample, how attention and demand vary throughout the model lifecycle.

4.1 Model lifecycle

Although major model releases generate a lot of hype, it remains unclear whether the hype translates into actual usage that lasts beyond the initial release. Figure 4.1 plots download and like counts on Hugging Face for the first year of the model lifecycle. We separate models based on downloads at release to allow for the possibility that the most popular models have distinct lifecycle patterns. Most models receive more likes and downloads in the first week of their existence than in any subsequent week, and downloads remain above zero for the duration of the model's life. While models in all popularity groups exhibit decaying downloads as they age, the rate of decrease varies significantly. Downloads for the most popular 1% of models decline gradually over time. For the bottom 75% of models, downloads drop quickly, and the level of downloads at six months is nearly the same as the level at one year. Download activity does, however, increase and decrease over the

Figure 4.1: Median model lifecycle by initial popularity



lifecycle, potentially reflecting spillovers from newer model releases which we investigate next. New model releases also do not supersede their predecessors immediately. For example, Figure 3.2 shows that Llama 3.1 70B was released about six months after Llama 3 70B and ostensibly should have made Llama 3 obsolete. But our estimates of the lifecycle indicate that many users probably still downloaded Llama 3 70B even several months after Llama 3.1 was released. Exploring why exactly this happens is beyond the scope of our paper, which aims only to characterize the model lifecycle, but possible mechanisms include idiosyncratic features we do not observe (like the tone of a model's responses) and switching costs (like testing and verification requirements for adopting new models in business processes).

Likes show a drastically different pattern from downloads. The median model receives zero likes every week after and including its release week. Even among the top 0.1% of models, likes are rare; the median number of likes in this group in a model's release week is only ten. Likes primarily measure attention rather than demand for the model in software applications. On Hugging Face, attention to models appears scarce,

but models that receive little attention are often still heavily used.

When estimating this "hype-cycle" versus the lifecycle, simply averaging likes or downloads across models may mask important variation and trends. For example, since we do not observe all models for the same number of weeks, composition effects could explain sudden jumps in download counts. We address this by implementing a regression specification that allows model age to more flexibly impact attention and demand. We control for model-specific features since models might have features that affect their use but do not reflect true utility, like better branding which attracts users but does not affect model performance. We also control for calendar time to address broad trends in model use and adoption over time. For example, Hugging Face and large language models in general became more popular over our sample period, and the release of DeepSeek-R1 represented a unique moment in the history of transformer models. We want our picture of the model lifecycle to capture features inherent to the models rather than picking up broader trends in the data. We also control for the day of week on which a model was released to address, for example, the possibility that Hugging Face users download models for work, in which case downloads would be less likely on weekends. Figure C.3 plots the effect of model age on downloads and likes after controlling for these factors and confirms the basic shape of the lifecycle: an immediate spike in downloads at release followed by declining downloads with age.

As an additional check, Figure C.4 plots the median model lifecycle by parameter count rather than initial downloads. While there is a clear relationship between the shape of the lifecycle curve and model popularity, the relationship is less one-to-one with size. For example, mid-sized models with one billion to ten billion parameters see a faster decay in downloads than the smallest models (with under one billion parameters). Attention, though, remains rare. One month after release, the median model in all size classes receives zero likes.

4.2 Spillover effects

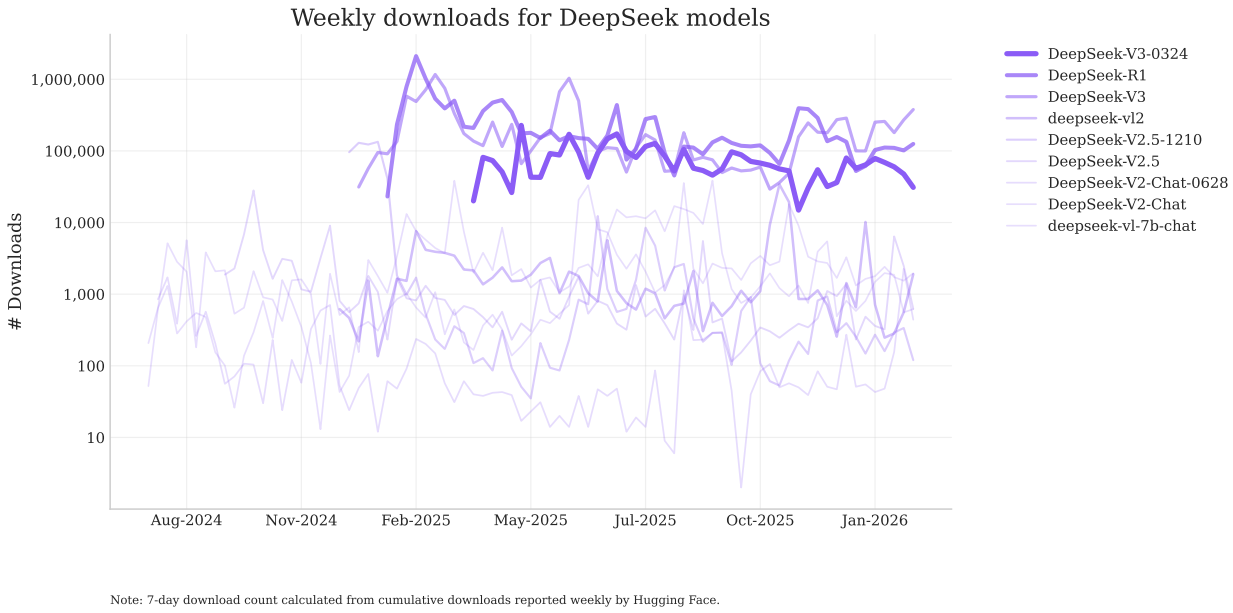
Now that we have documented patterns around model release, we shift to examining what happens during the second phase of the lifecycle: how models are used and how they interact with subsequent releases. In particular, we focus on quantifying *spillover effects*: when a developer releases a new model, does it cannibalize or complement existing, related models?

The answer is not obvious, and different market analogs highlight competing factors. In many product markets, a new version displaces the old one. A new iPhone draws sales away from the previous generation; a new car model year makes last year's version less desirable. If demand for AI models worked this way, older models would see declining use after a new release. Other markets exhibit the opposite pattern. A bestselling novel often lifts sales of the author's backlist; a hit album draws listeners to an artist's earlier music (Hendricks and Sorensen, 2009).[25] In these settings, new releases function as discovery events that expand attention to the creator's full body of work. The open-weight AI ecosystem has features that could push in either direction. A new model might simply replace its predecessor, as with cars. Or a prominent release might draw attention to the developer's brand and catalog, leading users to discover older models suited to different tasks or hardware configurations, as with books and music. If models within a developer's portfolio serve partially complementary roles (a large flagship model for high-stakes tasks, a smaller model for latency-sensitive applications, a fine-tuned variant for a specialized domain), then a new release could expand the set of users engaging with the portfolio rather than simply redirecting existing users to the newest version.

[25] Hendricks and Sorensen (2009) call this "backward spillovers" and attribute it to consumer discovery.

Figure 4.2 offers a suggestive example. When DeepSeek released R1 in January 2025, downloads of its older models, including DeepSeek-V3 and several V2 variants, rose sharply. Rather than displacing these earlier models, the R1 launch appears to have drawn users into DeepSeek's broader catalog. The pattern is consistent with backward spillovers: a prominent new release increases demand for the developer's existing portfolio. Of course, a single time series cannot establish causality. The DeepSeek downloads are themselves quite noisy and downloads of all models on Hugging Face were rising in early 2025, and DeepSeek-R1 was itself a major industry event that may have increased platform-wide activity. To isolate the spillover effect from these confounds, we turn to a more formal analysis.

Figure 4.2: Downloads for DeepSeek models



Note: 7-day download count calculated from cumulative downloads reported weekly by Hugging Face.

4.2.1 Spillover Empirical Strategy

To test for spillovers systematically, we estimate a regression specification that isolates the effect of a new model release on the developer's previous models. We focus on releases that are likely to be salient enough to trigger portfolio-wide discovery: models that rank in the top 30 by Hugging Face's trending score in their release week. Hugging Face calculates this score based on downloads, likes, and other activity; while the exact algorithm is not public, it captures models that receive substantial attention. DeepSeek-R1, for example, was the top trending model for several weeks after its January 2025 release.

We focus on popular releases for two reasons. First, the birth section establishes that developers release models frequently, often in quick succession within a batch. Most of these releases are not individually salient events; they are part of a continuous flow of uploads. Estimating spillovers from every subsequent release introduces noise and makes it difficult to identify meaningful effects. Second, popular releases are the cases where spillover effects, if they exist, are most likely to be economically meaningful. A trending model generates the kind of attention that could plausibly redirect user behavior across a developer's portfolio.

Let j index models and t index calendar weeks. We estimate the following specification:

$$y_{jt} = \sum_{m \in \mathcal{M}} \left(\sum_{k=1}^{K_m} \tau_k^m \mathbb{1}(r_{jt}^m \geq k) + \mathbb{1}(r_{jt} > K_m) \left(\kappa_1^m + \kappa_2^m \log\left(r_{jt} - K_m \right) \right) \right) + \xi_j + \delta_t + \alpha_{b(j),t-t_0^j} + \varepsilon_{jt} \quad (4.1)$$

where y_{jt}, the outcome, is the number of downloads model j received in week t. Our parameters of interest, τ_k^m, denote the impact of subsequent releases on the outcome. We index τ_k^m based on the relationship between the focal model and related models and the within-relationship relative order of release: we use m to denote the type of relationship between models, and we use k to denote the relative release order of the related models (within group m). We estimate up to K_m separate τ_k^m terms for each m, and use κ_1^m and κ_2^m to control for any additional spillovers for releases beyond K_m. The parameter ξ_j is a model fixed effect. δ_t is a calendar week fixed effect. We control for the heterogeneous model lifecycle patterns, as documented in Figure 4.1, with $\alpha_{b(j),t-t_0^j}$, which are initial popularity bin ($b(j)$) and age ($t - t_0^j$) specific fixed effects.

We analyze the following model relationships, as captured by $m \in M$:

1. Same-author non-descendant: a model released by the same author after the focal model.
2. Same-author descendant: a model descended from the focal model, created by the same developer.
3. Third-party descendant: a model descended from the focal model, created by a different developer.

For each relationship type, we divide models into those whose release-week trending score was in the top 10 and those in the top 11-30. We separate the effects of a recent release (up to four weeks old) and a mature release (more than four weeks old). In total, $\mathcal{M}$ includes twelve release types and we estimate the effect of all types jointly. For each type, m, we choose a maximum number of subsequent releases, K_m based on the number of releases of that type. For same-author non-descendants, we set $K_m = 9$. For same-author descendants, we set $K_m = 2$, and we exclude top-coding since we never observe more than two subsequent releases of this type. For third-party descendants, we set $K_m = 2$. Figure C.6 motivates these choices. Essentially, we top-code further releases once the number of observations with at least that number of releases has dropped ten-fold compared to the number of observations with at least one release.

Even popular models have weeks with zero downloads or likes. We estimate the coefficients using pseudo-Poisson maximum likelihood (PPML), which is well suited for our setting: PPML is consistent for the conditional mean when the count outcome has zeros (Silva and Tenreyro, 2006; Cohn et al., 2021), and allows for high-dimensional fixed effects without an incidental parameters problem (Wooldridge, 1999; Correia et al., 2020).[26] We include model fixed effects to absorb time-invariant differences across models, calendar week fixed effects to control for platform-wide trends in usage, and model age fixed effects to ensure we are not attributing the natural decline in downloads with age to the effect of a new release. Since the model lifecycle varies starkly by popularity, we interact the age fixed effects with initial model popularity. We restrict the subsequent release to models at least seven days ahead of the focal model. Without this restriction, a model released one day after the focal model would count as a source of potential spillovers, but the focal model would still see substantial activity due to its own novelty, making it difficult to separate the spillover from standard lifecycle dynamics.

Mapping differences in downloads to differences in demand Notably, the PPML specification with model fixed effects lets us read the estimates as effects on demand and not only on downloads. Downloads

[26] For count outcomes with zeros, regressions of a $\log(1+x)$ or inverse hyperbolic sine transformation introduces a variety of issues (Chen and Roth, 2024; Mullahy and Norton, 2024)

are our primary outcome and are of interest in their own right. The object of economic interest, however, is demand for the model, and downloads measure demand only up to a scaling factor that may differ across models. Suppose $y_{jt} = \lambda_j \mu_t q_{jt}$, where q_{jt} is demand for model j in week t, λ_j is a model-specific ratio of downloads to demand, and μ_t is a ratio common to all models in week t. Since the conditional mean in Equation (4.1) is multiplicative, λ_j is absorbed by the model fixed effect and μ_t by the calendar week fixed effect, leaving the treatment coefficients unchanged. The assumption we need is that λ_j does not respond to the release events we study: a new release must not change how many downloads a given quantity of demand generates.[27]

4.2.2 Spillover Results

Figure 4.3 plots the estimated coefficients $\tau_1^m, \tau_2^m, \kappa_1^m$ for third-party descendant models. The results reveal a clear spillover pattern: popular new releases increase downloads of the developer's older models, and the effect is persistent. We also see a short-lived increase in likes.

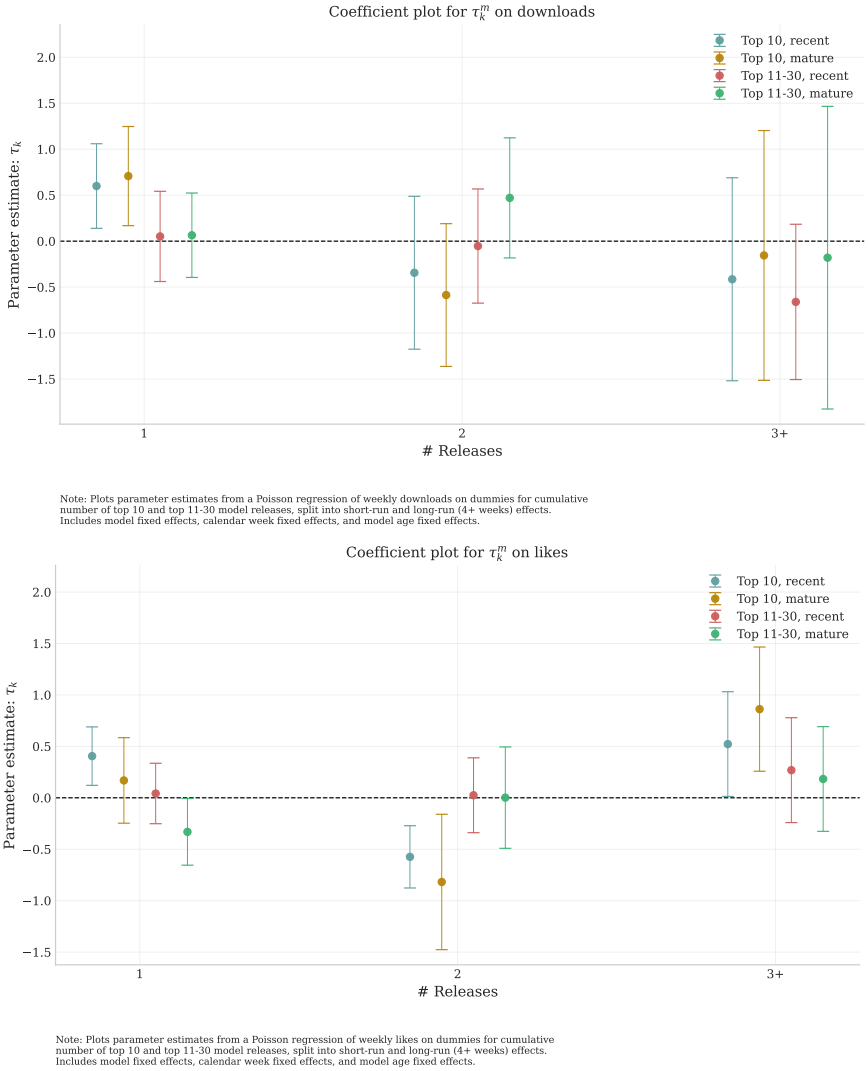
Downloads. The first third-party descendant release raises downloads of the focal model. The short-run coefficient, capturing the four weeks following the release, is positive and statistically significant, and the long-run coefficient, capturing the period beyond four weeks, is of comparable magnitude, so the effect does not fade once the initial attention subsides. Descendants ranked in the top 10 by trending score drive the result, while those ranked 11 to 30 have no statistically distinguishable effect, consistent with more prominent releases generating stronger discovery. We find only noisy effects from the second, third, and later releases; the first salient descendant accounts for most of the spillovers. The positive long-run coefficient for the first release is consistent with users discovering genuine use cases for older models rather than merely downloading them out of curiosity. If DeepSeek-R1 leads a user to explore DeepSeek's catalog and discover that DeepSeek-V3 performs well for their particular application, that user may continue relying on DeepSeek-V3 after the initial activity around R1's release subsides.

Likes. The response of likes is more varied. The first prominent release generates a short-run increase in likes for the developer's older models, consistent with users exploring the developer's existing catalog of models. The coefficient on the second subsequent release is negative, indicating a reversal of the initial bump, while the coefficient on the third release and beyond is again positive, possibly because source models with three or more descendants are particularly selected. The variation might occur because likes measure the "hype" around a model, which can both build up and dissipate rapidly, while actually downloading a model takes more time and may be stickier. This divergence between likes and downloads underscores a distinction that runs throughout our analysis: attention and utility are not the same thing, and metrics that proxy for hype can tell a different story than metrics that proxy for use.

Interpretation. Several mechanisms could drive the positive spillovers we observe. First, a prominent release might generate brand recognition, raising awareness of a developer's full catalog of models among users who were previously unfamiliar with the developer. Second, users evaluating a new model may benchmark it against the developer's older models, generating downloads in the process. Third, if models within a portfolio serve complementary roles (different sizes for different hardware, different specializations for different tasks),

[27] This assumption would fail if, for example, a successor release prompts users to re-download the focal model only to benchmark the two against each other.

Figure 4.3: Effects of subsequent model releases: third-party descendant models



a new release may expand the portfolio's coverage and draw users who then adopt multiple models. Fourth, media coverage and community discussion of a trending model often reference the developer's earlier work, creating organic discovery channels.

We cannot definitively distinguish among these mechanisms with our data, but the persistence of the download effect favors explanations rooted in genuine complementarity and discovery over simple curiosity. If users were merely trying out old models and then abandoning them, we would expect the effect to dissipate quickly. Instead, the sustained increase in downloads suggests that at least some users find lasting value in older models after encountering the developer through a new release.

Robustness across release types and samples. We estimate Equation 4.1 jointly across all three release types and on three samples of focal models: the full sample and the top 10% and top 1% of focal models by initial popularity. Table 4.1 reports the effects of third-party descendant models, and Section C.3 reports the estimates for same-author descendant and non-descendant models along with the corresponding coefficient

plots.[28] The third-party descendant effect is the most robust, raising focal-model downloads in all three samples; same-author descendant and non-descendant releases produce noisier, less consistent effects. In no case do we find strong evidence of cannibalization.

These results have implications for how we think about competition in the AI model market. The standard intuition that new products cannibalize old ones does not hold here. Instead, a developer's models appear to function more like a portfolio of complementary products than a sequence of substitutes. This matters for market definition in antitrust analysis: aggregating a developer's models into a single product would obscure the within-portfolio dynamics that our estimates reveal. It also matters for developers' release strategies: the evidence suggests that investing in a high-profile release can generate returns across the full portfolio, not just for the new model itself. We now turn to the final phase of the lifecycle: when and how models exit the market.

[28] While we report coefficients in separate tables for readability, the effects are all estimated jointly using Equation (4.1).

Table 4.1: Spillover effect estimates for third-party descendant models

	Downloads			Likes		
	Full	Top 10%	Top 1%	Full	Top 10%	Top 1%
Top 10						
1 release (recent)	0.5996**	0.5612**	0.4251**	0.4051***	0.4096***	0.3404***
	(0.2347)	(0.2334)	(0.1734)	(0.1450)	(0.1338)	(0.1137)
2 releases (recent)	-0.3437	-0.3254	-0.3162	-0.5740***	-0.5644***	-0.7774***
	(0.4241)	(0.4231)	(0.3921)	(0.1544)	(0.1503)	(0.1795)
3+ releases (recent)	-0.4145	-0.4813	-0.4858	0.5221**	0.4741*	0.6962**
	(0.5632)	(0.5599)	(0.5501)	(0.2597)	(0.2527)	(0.2929)
Log(3+ releases) (recent)	-0.0257	0.0040	0.1768	0.1104	0.1738	0.2164
	(0.5006)	(0.5227)	(0.5614)	(0.2879)	(0.3055)	(0.3964)
1 release (mature)	0.1080	0.1241	0.5181**	-0.2363	-0.1999	0.0179
	(0.1775)	(0.1781)	(0.2018)	(0.1523)	(0.1572)	(0.1720)
2 releases (mature)	-0.2423	-0.2313	-0.4673	-0.2444	-0.2287	-0.6874***
	(0.4921)	(0.4657)	(0.5291)	(0.2459)	(0.2456)	(0.2322)
3+ releases (mature)	0.2591	0.2865	0.0142	0.3399***	0.3002**	0.3897***
	(0.2785)	(0.2803)	(0.3065)	(0.1280)	(0.1350)	(0.1403)
Log(3+ releases) (mature)	0.0650	-0.0497	-0.0823	-0.6038**	-0.4516	-0.4111
	(0.5144)	(0.5285)	(0.5255)	(0.2752)	(0.2884)	(0.3596)
Top 11–30						
1 release (recent)	0.0516	0.1083	0.5262**	0.0420	0.0228	0.0984
	(0.2503)	(0.2563)	(0.2627)	(0.1499)	(0.1485)	(0.1394)
2 releases (recent)	-0.0532	-0.0886	-0.2831	0.0247	0.0481	0.4428***
	(0.3168)	(0.3045)	(0.2912)	(0.1857)	(0.1806)	(0.1633)
3+ releases (recent)	-0.6607	-0.6107	-0.9200*	0.2687	0.2496	0.8833***
	(0.4309)	(0.4314)	(0.4930)	(0.2601)	(0.2594)	(0.3072)
Log(3+ releases) (recent)	1.776*	1.874*		1.012	1.063	
	(0.9439)	(0.9721)		(0.8355)	(0.8154)	
1 release (mature)	0.0129	0.0264	0.2119	-0.3730***	-0.3970***	-0.5293***
	(0.1940)	(0.1962)	(0.2517)	(0.1073)	(0.1103)	(0.0995)
2 releases (mature)	0.5239**	0.4981**	0.4315	-0.0230	-0.0400	0.0499
	(0.2130)	(0.2189)	(0.2950)	(0.1784)	(0.1876)	(0.1467)
3+ releases (mature)	0.4810	0.5699	0.7116	-0.0858	-0.0715	-0.9770***
	(0.6387)	(0.6296)	(0.6971)	(0.1995)	(0.2017)	(0.3347)
Log(3+ releases) (mature)	-1.575*	-1.630*		-1.023**	-1.080**	
	(0.9119)	(0.8808)		(0.4845)	(0.4780)	
Model	Yes	Yes	Yes	Yes	Yes	Yes
Date (week)	Yes	Yes	Yes	Yes	Yes	Yes
Age*initial popularity	Yes	Yes	Yes	Yes	Yes	Yes
Observations	6,749,333	634,679	57,536	2,178,219	410,241	43,400
Pseudo R^2	0.91381	0.89629	0.90198	0.76996	0.82161	0.86925

Standard errors in parentheses clustered at the model level
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

5 Death

We document above that most downloads occur around a model's release, yet models often remain in use long afterward. What, then, marks the end of a model's lifecycle? In this section, we distinguish between two concepts. We use *usage obsolescence* to describe the point at which a model no longer attracts meaningful usage on a major platform. We use *technical obsolescence* to describe the point at which a model has been displaced by stronger alternatives on benchmark-based or preference-based rankings. Since downloads on Hugging Face are our primary outcome of interest, we do not use them to define obsolescence; doing so would mechanically induce a decline in downloads near the end of the lifecycle. Instead, we combine several external data sources to measure when models become obsolete.

5.1 Usage obsolescence

Because OpenRouter is one of the largest platforms for accessing AI models, it offers a useful window into their lifecycles. OpenRouter reports total tokens used by each model on a daily basis and removes outdated models. We treat removal from the platform as evidence that a model has reached obsolescence. We also classify a model as usage-obsolete when its share of total platform tokens becomes persistently negligible. Concretely, for each model we observe on OpenRouter, we record the first date on which *either* of the following conditions holds:

1. We no longer observe the model on OpenRouter.
2. The share of tokens accruing to model j in week t,

$$s_{jt} = \frac{\text{tokens}_{jt}}{\sum_{k \in \mathcal{K}} \text{tokens}_{kt}},$$

is in the bottom 10% of the distribution of $\{s_{jt}\}_j$ in that week and in every week thereafter, where $\mathcal{K}$ is the set of all models on OpenRouter in week t.

Table 5.1: OpenRouter usage obsolescence summary

	All models (n=717)		Hugging Face models (n=321)	
	Number	Share	Number	Share
Dropout	72	10.0%	16	5.0%
Token share	153	21.3%	79	24.6%
Both	20	2.8%	6	1.9%

Note: Data as of February 11, 2026.

Table 5.1 summarizes how models become usage-obsolete on OpenRouter. The token share condition is more than twice as common as the dropout condition. Table D.1 lists the subset of models that eventually meet both criteria; for these, we use the earlier of the two dates as the usage obsolescence date. All twenty drop below the token share threshold first, and most of those are then removed from OpenRouter within two weeks. To maintain consistency throughout the paper, we restrict subsequent usage obsolescence analysis to the 321 economically relevant models in our main sample that can be matched to OpenRouter. When calculating the distribution of token shares, we use *all* models on OpenRouter to better capture each model's relevance compared to its full set of competitors, including proprietary models.

5.2 Technical obsolescence

To extend our ability to measure the end of a model's useful life, we analyze the data from Artificial Analysis and Arena. These data give us complementary measures of technical obsolescence: benchmark-based index scores and pairwise user preferences from the model arenas. Researchers, developers, and practitioners routinely consult Artificial Analysis and the Arena leaderboards when selecting models, so continued presence on the platforms is a meaningful signal of ongoing technical relevance. We define technical obsolescence by combining the three data sources described in Section 2.4:

1. Benchmark-based index scores from Artificial Analysis
2. Elo scores from pairwise model comparisons on Artificial Analysis
3. Elo scores from pairwise model comparisons on Arena

For each model in each data source, we identify the category in which it ranked highest in its release week; Table D.2 reports the number of models in each category. We then track the model's rank in that category against its competitors over time. The set of competitors changes as new models enter and platforms remove some old models.[29] Once the model's rank (in its best category) falls below 30 in *any* of the three data sources, we classify it as technically obsolete. Table 5.2 summarizes the combined obsolescence classifications.

Table 5.2: Technical obsolescence summary

	All models (n=773)		Hugging Face models (n=322)	
Platform	Number	Share	Number	Share
Any	616	79.7%	309	96.0%
Artificial Analysis Benchmarks	314	40.6%	194	60.2%
Arena.ai	301	38.9%	150	46.6%
Artificial Analysis Arena	121	15.7%	37	11.5%
Multiple (2+)	118	15.3%	71	22.0%

Note: Data as of February 11, 2026.

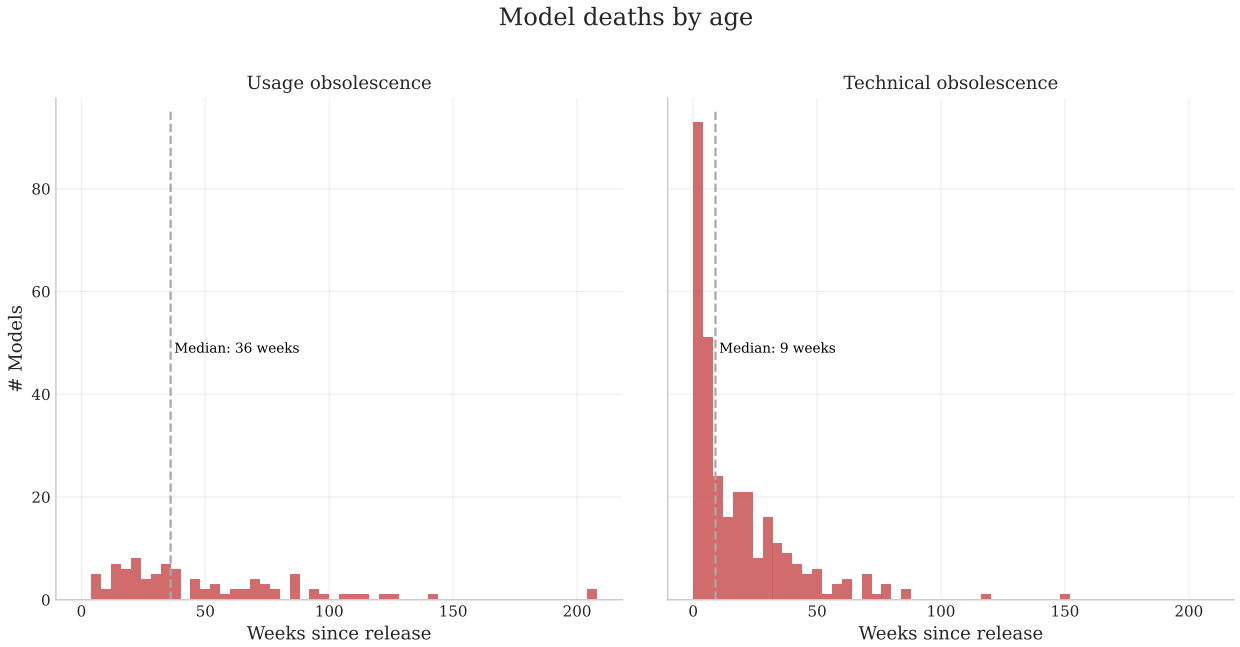
Each of the data sources above captures a different dimension of technological relevance. The Artificial Analysis index scores measure technical competitiveness on standardized benchmarks. The arena-based measures from both Artificial Analysis and Arena measure technical competitiveness through direct user preferences. Because benchmark results and model arenas capture different dimensions of model quality, combining them gives a more complete picture of the model lifecycle. The fact that a majority of models do reach technical obsolescence is consistent with the rapid pace of technological progress: a model that ranked near the frontier on a task two years ago has almost certainly been surpassed by newer models since then. Our survival analysis in the sections that follow analyzes both usage and technical obsolescence. We thereby capture models that lose relevance through declining use, weaker benchmark performance relative to newer models, or worsening user preference.

[29] When a model is removed from Artificial Analysis or Arena, we fill its most recently observed benchmark or Elo score forward to the present. This captures the fact that the scores are meant to be a time-invariant measure of model quality.

5.3 Age at obsolescence

Figure 5.1 plots the distribution of model age at obsolescence. The median model reaches usage obsolescence 36 weeks after release, although some models remain relevant much longer. Llama 2 (13B), for example, remains in use well after newer alternatives appear. Although Llama 3 superseded Llama 2 roughly a year later, Llama 2 does not reach usage obsolescence until substantially later. Technical obsolescence is more

Figure 5.1: Model age at obsolescence



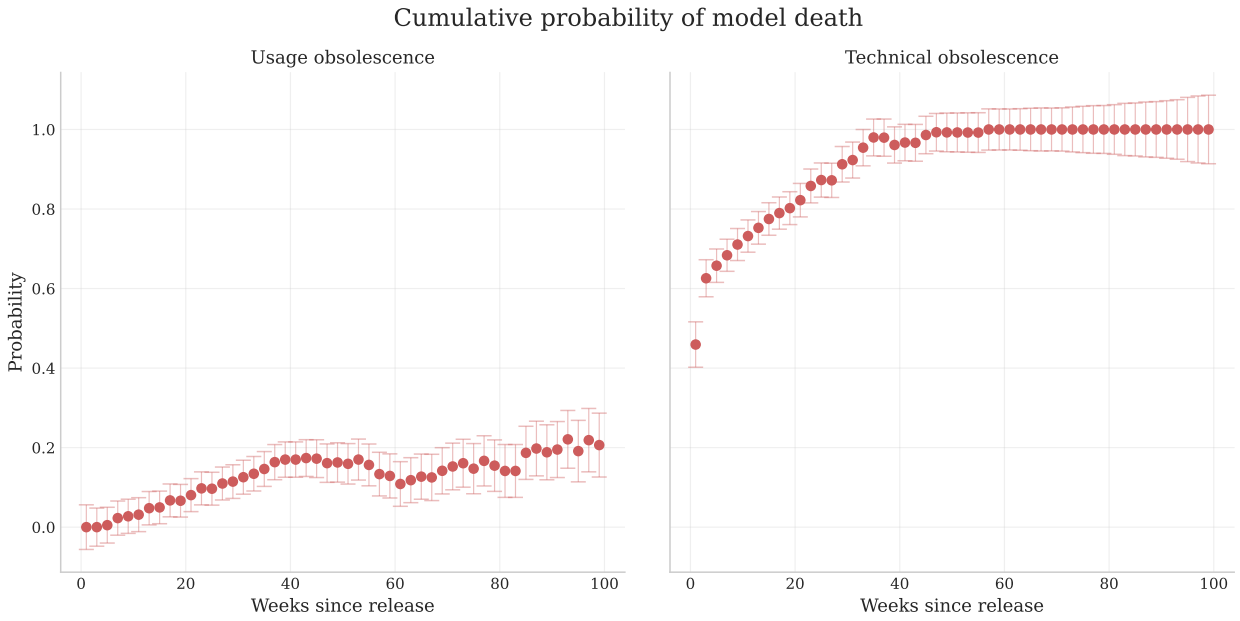
Note: Usage obsolescence indicates token share below 10th percentile on OpenRouter or removal from OpenRouter.
Technical obsolescence indicates dropping below rank 30 in highest-ranked category at release on Artificial Analysis or Arena.
Restricted to models matched to Hugging Face.

common and occurs much earlier than usage obsolescence: the median model is technically obsolete just nine weeks after release, and 28% of models in our sample reach technical obsolescence in their first month. Since hundreds of new open-weight models are released each month (Figure 3.1), it is not entirely surprising that the technological frontier advances quickly. The contrast between usage and technical obsolescence, though, is striking, and the relative rarity of usage obsolescence suggests that falling behind the frontier by no means leads users to abandon a model altogether. To reinforce these conclusions, we turn to a more precise description of how obsolescence risk evolves over the lifecycle. We begin with an ordinary least squares regression that estimates the cumulative probability that a model has reached obsolescence by a given age. Specifically, we implement the following specification separately for usage and technical obsolescence:

$$\texttt{obsolete}_{jt} = \beta_{t-t_0^j} + \varepsilon_{jt}, \tag{5.1}$$

where $\texttt{obsolete}_{jt} \in \{0, 1\}$ is an indicator equal to 1 if model j has reached obsolescence by calendar week t and 0 otherwise, and t_0^j indexes the calendar week in which model j was released. The $\beta_{t-t_0^j}$ terms are model-age dummies for each observed value of $t - t_0^j$. The regression is estimated by OLS with no intercept, so each estimated coefficient $\widehat{\beta}_{t-t_0^j}$ is the sample mean of $\texttt{obsolete}_{jt}$ at age $t - t_0^j$. Figure 5.2 plots the estimated coefficients $\hat{\beta}_0, \ldots, \hat{\beta}_{99}$; we do not observe enough technical-obsolescence events to estimate probabilities

Figure 5.2: Share of models reaching usage obsolescence by weeks since release.



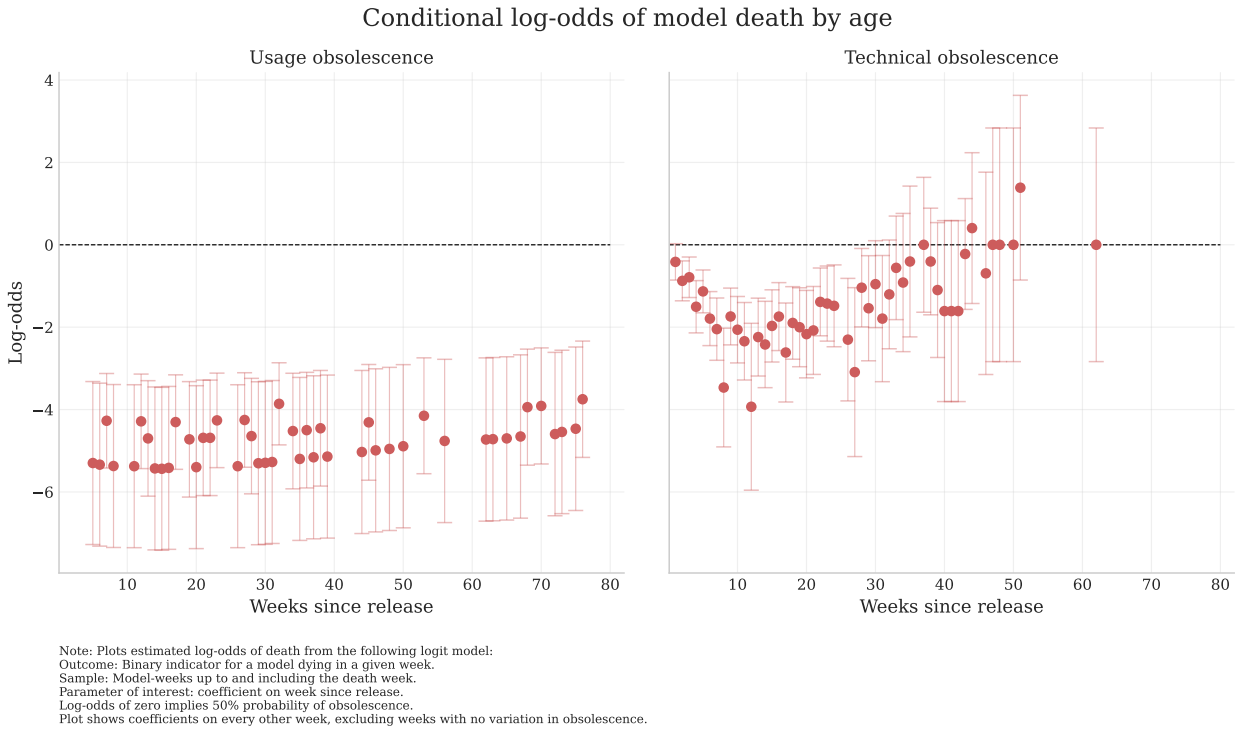
Note: Plots coefficients from the following ordinary least squares regression:
Outcome: Binary indicator for whether a model is obsolete in a given week.
Parameter of interest: Coefficient on week since release dummy (week 52 excluded).
Plot shows coefficients on every other week.

beyond 100 weeks of age. The probability of obsolescence increases with age. Usage obsolescence peaks around 20%, while technical obsolescence is practically guaranteed by one year after release. If we observed all models over their full lifecycle, these probabilities would increase monotonically. In practice, however, some models were only recently added to OpenRouter, Artificial Analysis, or Arena, so we do not observe their eventual obsolescence. As a result, we observe fewer models at 60 weeks since release than at 40 weeks since release, for example. To address this censoring problem, and because ordinary least squares regression is not ideal for estimating probabilities bounded between 0 and 1, we also estimate a hazard model. The hazard model answers questions of the following form: conditional on a model still being in use 40 weeks after release, what is the probability that it reaches usage obsolescence in week 41? To do so, we estimate the following specification:

$$\Pr(o_{jt} = 1 \mid o_{j,t-1} = 0) = \Lambda\left(\alpha_{t-t_0^j}\right) + \varepsilon_{jt}, \tag{5.2}$$

where $o_{jt} \in \{0, 1\}$ is an indicator equal to 1 if model j reaches obsolescence in calendar week t and 0 otherwise, t_0^j indexes the calendar week in which model j was released, and $\Lambda(x) = \frac{1}{1+e^{-x}}$ is the logistic function. The $\alpha_{t-t_0^j}$ terms are model-age fixed effects, with α_{52} as the excluded baseline. The estimation sample is restricted to model-weeks in which model j has not yet reached obsolescence at the start of week t, so the final observation for a given model is the week in which it becomes usage-obsolete. Standard errors are clustered at the model level. Figure 5.3 plots the estimated coefficients $\hat{\alpha}_0, \ldots, \hat{\alpha}_{76}$. We exclude coefficients beyond age 76 due to insufficient obsolescence data. We also exclude coefficients for model-ages at which we observe no variation in obsolescence, since we cannot reliably estimate them. Even so, the log-odds plot reinforces the pattern in the cumulative-probability plot above. The log-odds increase (that is, move closer to zero) at later model ages, which implies that conditional on surviving to a given age, an older model is more

Figure 5.3: Log-odds of obsolescence by age.



likely to reach obsolescence in the following week. The technical obsolescence plot exhibits an additional and more nuanced U-shape in log odds. Conditional on not being immediately technically obsolete, models become *less* likely to reach technical obsolescence up to about six months after release. This suggests that some models are technologically far superior to their competitors and therefore remain at the frontier for some time. As a final check, we compute the implied cumulative probability of obsolescence at each age using the hazard-model estimates. The implied survival curve as a function of model age is:

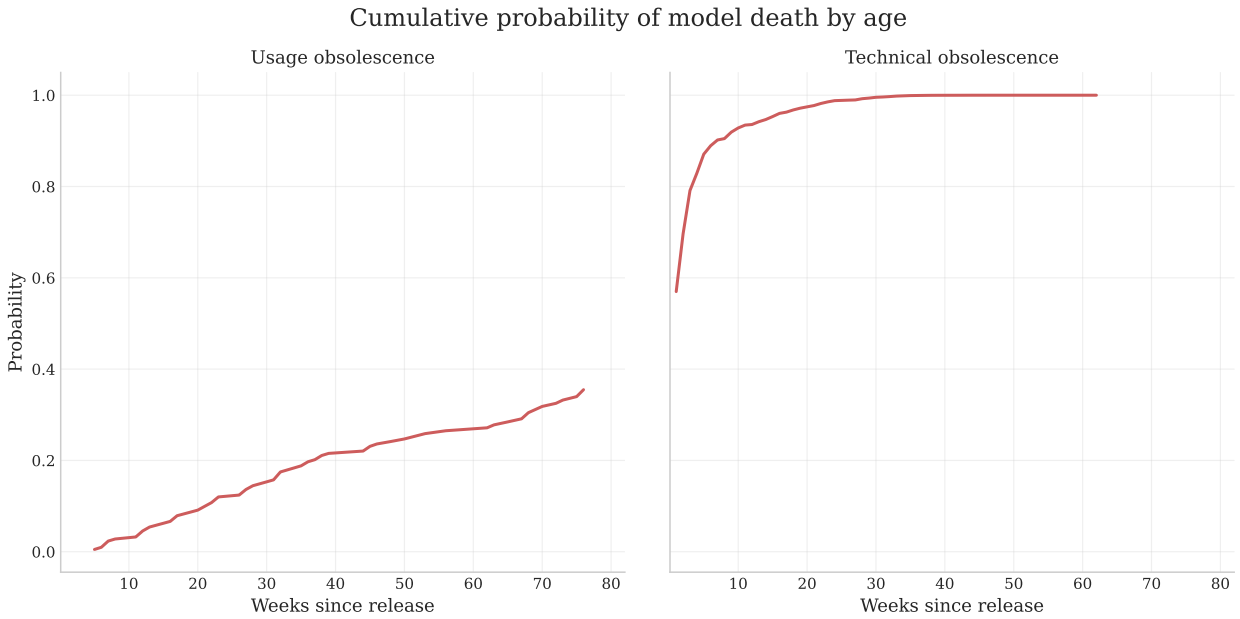
$$\widehat{S}(a) = \prod_{s=0}^{a} \left(1 - \Pr\Big(o_{jt} = 1 \mid o_{j,t-1} = 0,\ t - t_0^j = s\Big)\right), \tag{5.3}$$

and the corresponding cumulative probability of obsolescence by age a is

$$\widehat{F}(a) = 1 - \widehat{S}(a). \tag{5.4}$$

Figure 5.4 plots these calculated cumulative probabilities. Reassuringly, the estimates are similar to those in Figure 5.2. Usage obsolescence is less common than technical obsolescence and reaches about 20% probability at 40 weeks since release. Technical obsolescence is far more likely and approaches 100% probability by 40 weeks after release. Notably, the probability of usage obsolescence is less than 40% even 1.5 years after release. Understanding why users continue to rely on older models remains an interesting direction for future research.

Figure 5.4: Cumulative probability of usage obsolescence by weeks since release.



Note: Cumulative probability implied by estimated conditional probabilities.

5.4 Effects of obsolescence

Because our obsolescence measures are defined separately from our Hugging Face dataset, we can use them to study Hugging Face outcomes with less concern that the patterns are mechanical or endogenous. We are particularly interested in what happens to attention and demand on Hugging Face after a model reaches usage or technical obsolescence. To study Hugging Face activity around that event, we estimate event studies centered on both usage and technical obsolescence dates. We implement a standard panel event study specification described in Miller (2023) and implemented frequently in prior literature, such as Autor (2003). Specifically, we plot the estimated values of τ_k from the following specification:

$$y_{jt} = \sum_{k=-15}^{15} \tau_k \mathbb{1}\left(t - t_d^j = k\right) + \xi_j + \delta_t + \alpha_{b(j),t-t_0^j} + \varepsilon_{jt} \tag{5.5}$$

where y_{jt} is the number of downloads (or likes) model j received in week t, t_0^j is the calendar week of model j's release, and t_d^j indexes the calendar week in which model j reaches obsolescence. Our parameters of interest, τ_k, capture the effect on the focal model's Hugging Face outcome k weeks before or after it reaches obsolescence. The specification includes model fixed effects ξ_j, calendar-week fixed effects δ_t, and model-age fixed effects $\alpha_{t-t_0^j}$. We cluster standard errors at the model level. We estimate the coefficients using pseudo-Poisson maximum likelihood because downloads and likes include many zeroes and remain highly skewed even among economically relevant models. As in Equation 4.1, we include fixed effects for model, calendar week, and model age interacted with initial popularity to isolate how Hugging Face activity changes around obsolescence rather than around secular trends or the ordinary lifecycle of model usage.

Figure 5.5 plots the estimated coefficients $\hat{\tau}_{-15}, \ldots, \hat{\tau}_{15}$. Because we observe relatively few models on Artificial Analysis and Arena, the estimates are imprecise. Although the estimates are somewhat noisy, they

Figure 5.5: Demand and attention around obsolescence



suggest some decline in downloads and likes after a model becomes usage-obsolete. Since usage obsolescence is defined using OpenRouter data while downloads and likes are measured in the Hugging Face data, these effects are not purely mechanical. One plausible interpretation is that once a model becomes usage-obsolete on OpenRouter, it receives less attention from model users more broadly, and that decline spills over to Hugging Face. At the same time, the absence of a large negative discontinuity in downloads provides further evidence of the surprising longevity of open-weight models. Even after a model reaches usage obsolescence on a major platform such as OpenRouter, its activity on Hugging Face does not fall to zero. If OpenRouter usage obsolescence is a credible signal that a model has largely lost commercial relevance, the persistence of Hugging Face activity suggests that open-weight models continue to provide value well beyond that point.

Technical obsolescence has no statistically significant effect on Hugging Face downloads, while it does lead to a persistent decline in likes. Downloads proxy for use, and likes measure attention. The pattern therefore suggests that as the technological frontier moves forward, users shift their attention to the most cutting-edge models but continue to find use-cases for older models.

These results underscore the continuing relevance of older models and have implications about the

structure of the AI market. Despite frequent new model releases, older models can retain economic relevance long after release. The market for open-weight models is one where products have long lifespans and new products compete with a variety of existing products rather than immediately superseding them.

6 Conclusion

This paper documents the lifecycle of open-weight AI models in three phases: birth, life, and death. We show that the open-weight ecosystem is best understood not as a sequence of isolated model launches, but as a market organized around model families, descendant models, and persistent legacy use. In the birth phase, we find that economically important releases often arrive as portfolios of vertically and horizontally differentiated variants. In the life phase, we find that models continue to create value after release through cumulative innovation and through backward spillovers from later launches. In the death phase, we find that technical supersession does not imply economic exit: older models frequently persist through downstream reuse and embedded workflows. Across all three phases, attention and adoption follow different patterns. Likes and trending status capture a short hype-cycle; downloads and token usage reveal a much longer economic lifecycles.

These facts matter beyond their immediate empirical context. As we argue in the introduction, research on the value of training data, the returns to frontier compute, market definition in competition analysis, and the design of model deprecation policies all benefit from getting the institutional details right. A researcher who treats each model as an independent product will mismeasure the return to data investments. An antitrust analysis that aggregates a developer's portfolio into a single product will miss the complementarities our spillover estimates reveal. A policy that mandates sunsetting older models will need to account for the downstream dependencies and embedded workflows that sustain their use. We hope the institutional details here provide a foundation that enables subsequent work to better advance our understanding of the AI industry and its broader impacts.

References

Aubakirova, M., A. Atallah, C. Clark, J. Summerville, and A. Midha (2026, January). State of AI: An Empirical 100 Trillion Token Study with OpenRouter. arXiv:2601.10088 [cs.AI].

Autor, D. (2003, January). Outsourcing at Will: The Contribution of Unjust Dismissal Doctrine to the Growth of Employment Outsourcing. *Journal of Labor Economics 21*(1), 1–42.

Azoulay, P., J. Krieger, and A. Nagaraj (2024, May). Old Moats for New Models: Openness, Control, and Competition in Generative AI. Technical Report w32474, National Bureau of Economic Research, Cambridge, MA.

Bresnahan, T. (2024, March). What innovation paths for AI to become a GPT? *Journal of Economics & Management Strategy 33*(2), 305–316.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020, July). Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL].

Chen, J. and J. Roth (2024, March). Logs with Zeros? Some Problems and Solutions. *The Quarterly Journal of Economics 139*(2), 891–936.

Cohn, J. B., Z. Liu, and M. Wardlaw (2021). Count Data in Finance. *SSRN Electronic Journal*.

Conlon, C., N. H. Miller, T. Otgon, and Y. Yao (2023, May). Rising Markups, Rising Prices? *AEA Papers and Proceedings 113*, 279–283.

Correia, S., P. Guimarães, and T. Zylkin (2020, March). Fast Poisson estimation with high-dimensional fixed effects. *The Stata Journal: Promoting communications on statistics and Stata 20*(1), 95–115.

De Loecker, J., J. Eeckhout, and G. Unger (2020, May). The Rise of Market Power and the Macroeconomic Implications*. *The Quarterly Journal of Economics 135*(2), 561–644.

Demirer, M., A. Fradkin, N. Tadelis, and S. Peng (2025). The Emerging Market for Intelligence: Pricing, Supply, and Demand for LLMs.

Dettmers, T., A. Pagnoni, A. Holtzman, and L. Zettlemoyer (2023). QLoRA: Efficient Finetuning of Quantized LLMs. Version Number: 1.

Dunne, T., M. J. Roberts, and L. Samuelson (1988). Patterns of Firm Entry and Exit in U.S. Manufacturing Industries. *The RAND Journal of Economics 19*(4), 495.

Fradkin, A. (2025). Demand for LLMs: Descriptive Evidence on Substitution, Market Expansion, and Multihoming. Version Number: 1.

Gort, M. and S. Klepper (1982, September). Time Paths in the Diffusion of Product Innovations. *The Economic Journal 92*(367), 630.

Griliches, Z. (1957, October). Hybrid Corn: An Exploration in the Economics of Technological Change. *Econometrica 25*(4), 501.

Hendricks, K. and A. Sorensen (2009, April). Information and the Skewness of Music Sales. *Journal of Political Economy 117*(2), 324–369.

Hoffmann, J., S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. v. d. Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre (2022). Training Compute-Optimal Large Language Models. Version Number: 1.

Jiang, W., N. Synovic, M. Hyatt, T. R. Schorlemmer, R. Sethi, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis (2023). An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry. Version Number: 1.

Kadasi, P., S. R. Kondam, S. V. Chaturvedula, R. Sen, A. Saha, S. Sikdar, S. Sarkar, S. Mittal, R. Jindal, and M. Singh (2025, April). Model Hubs and Beyond: Analyzing Model Popularity, Performance, and Documentation. arXiv:2503.15222 [cs.CL].

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). Scaling Laws for Neural Language Models. Version Number: 1.

Kapoor, S., R. Bommasani, K. Klyman, S. Longpre, A. Ramaswami, P. Cihon, A. Hopkins, K. Bankston, S. Biderman, M. Bogen, R. Chowdhury, A. Engler, P. Henderson, Y. Jernite, S. Lazar, S. Maffulli, A. Nelson, J. Pineau, A. Skowron, D. Song, V. Storchan, D. Zhang, D. E. Ho, P. Liang, and A. Narayanan (2024). On the Societal Impact of Open Foundation Models. Version Number: 1.

Korinek, A. and J. Vipra (2024, November). Concentrating Intelligence: Scaling and Market Structure in Artificial Intelligence. Technical Report w33139, National Bureau of Economic Research, Cambridge, MA.

Laufer, B., H. Oderinwale, and J. Kleinberg (2025). Anatomy of a Machine Learning Ecosystem: 2 Million Models on Hugging Face. Version Number: 1.

Lerner, J. and J. Tirole (2002, June). Some Simple Economics of Open Source. *The Journal of Industrial Economics 50*(2), 197–234.

Longpre, S., C. Akiki, C. Lund, A. Kulkarni, E. Chen, I. Solaiman, A. Ghosh, Y. Jernite, and L.-A. Kaffee (2025, November). Economies of Open Intelligence: Tracing Power & Participation in the Model Ecosystem. arXiv:2512.03073 [cs.CY].

Mertens, M., N. Fischl-Lanzoni, and N. Thompson (2026). Is there "Secret Sauce" in Large Language Model Development? Version Number: 2.

Miller, D. L. (2023, May). An Introductory Guide to Event Study Models. *Journal of Economic Perspectives 37*(2), 203–230.

Mullahy, J. and E. C. Norton (2024, April). Why Transform Y ? The Pitfalls of Transformed Regressions with a Mass at Zero*. *Oxford Bulletin of Economics and Statistics 86*(2), 417–447.

Osborne, C., J. Ding, and H. R. Kirk (2024, October). The AI Community Building the Future? A Quantitative Analysis of Development Activity on Hugging Face Hub. *Journal of Computational Social Science 7*(2), 2067–2105. arXiv:2405.13058 [cs.SE].

Owen, D. (2024, January). How predictable is language model benchmark performance? arXiv:2401.04757 [cs.LG].

Rein, D., B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman (2023, November). GPQA: A Graduate-Level Google-Proof Q&A Benchmark. arXiv:2311.12022 [cs.AI].

Silva, J. M. C. S. and S. Tenreyro (2006, November). The Log of Gravity. *The Review of Economics and Statistics 88*(4), 641–658.

Wan, A., K. Klyman, S. Kapoor, N. Maslej, S. Longpre, B. Xiong, P. Liang, and R. Bommasani (2025, December). The 2025 Foundation Model Transparency Index. arXiv:2512.10169 [cs].

Wei, J., Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus (2022, October). Emergent Abilities of Large Language Models. arXiv:2206.07682 [cs.CL].

Wooldridge, J. M. (1999, May). Distribution-free estimation of some nonlinear panel data models. *Journal of Econometrics 90*(1), 77–97.

Appendix

A Data

A.1 Constructing Weekly Downloads

If we observed every model in the data starting on its creation date, we could use the rolling 30 day download count to calculate cumulative downloads. From that we could calculate the number of times the model was downloaded each week. However, our data is truncated at July 2024, so we do not observe initial values of rolling 30 day downloads for all models. We observe cumulative downloads directly starting in February 2025, which creates a six month gap during which we do not observe cumulative downloads. We remedy this with the following observation. Let D_t represent cumulative model downloads at time t and let R_t be the number of downloads the model received in the 30 days leading up to and including t. Then

$$D_{t-30} = D_t - R_t \tag{A.1}$$

Since we observe D_t and R_t at the daily level, we can use Equation A.1 to calculate D_t for each day between July 2024 and February 2025. Then we can calculate $d_t = D_t - D_{t-1}$, the number of downloads the model received on day t. Specifically, we use the following algorithm:

1. Initialize $\tilde{D}_{jt} = D_{jt}$ wherever D_{jt} is observed.
2. Repeatedly enforce monotonicity on the current series. If the series decreases between some date and the next non-missing date, find the largest such drop, set both endpoints of that drop to missing, and repeat until the remaining non-missing series is weakly increasing.
3. Interpolate the values removed in step 2 using shape-preserving cubic interpolation between observed points and linear extrapolation at the ends. If only one non-missing value remains, fill the series by carrying that value forward and backward. Replace only the values removed in step 2 with these interpolated values.
4. Use the 30-day identity $\tilde{D}_{j,t-30} = \tilde{D}_{jt} - R_{jt}$ to backfill missing cumulative downloads. To allow for small timing mismatches across snapshots, form

   $$B_{jt} = \text{mean}\left\{\tilde{D}_{jt} - R_{j,t-1}, \tilde{D}_{jt} - R_{jt}, \tilde{D}_{jt} - R_{j,t+1}\right\},$$

   where the mean is taken over available terms, and use B_{jt} to fill $\tilde{D}_{j,t-30}$ whenever that earlier value is missing.
5. Repeat steps 2–4 until no new values are filled. Then fill any remaining gaps by the interpolation rule in step 3.
6. After processing all models, any negative imputed values are set to missing, the code checks that each model's cumulative download series is weakly increasing, and the final series is rounded to integers before it is merged into the panel as cumulative downloads.

After completing the backfilling process, we observe a monotonically increasing series of cumulative downloads for each model j and each day t in the data. Note that we observe cumulative likes directly for the duration

of our sample, so the likes variable does not suffer from this missing data problem. All analysis proceeds with these data aggregated to the weekly level.

A.2 Matching OpenRouter and Hugging Face

We match OpenRouter models to Hugging Face models using Hugging Face model IDs reported in the OpenRouter data.

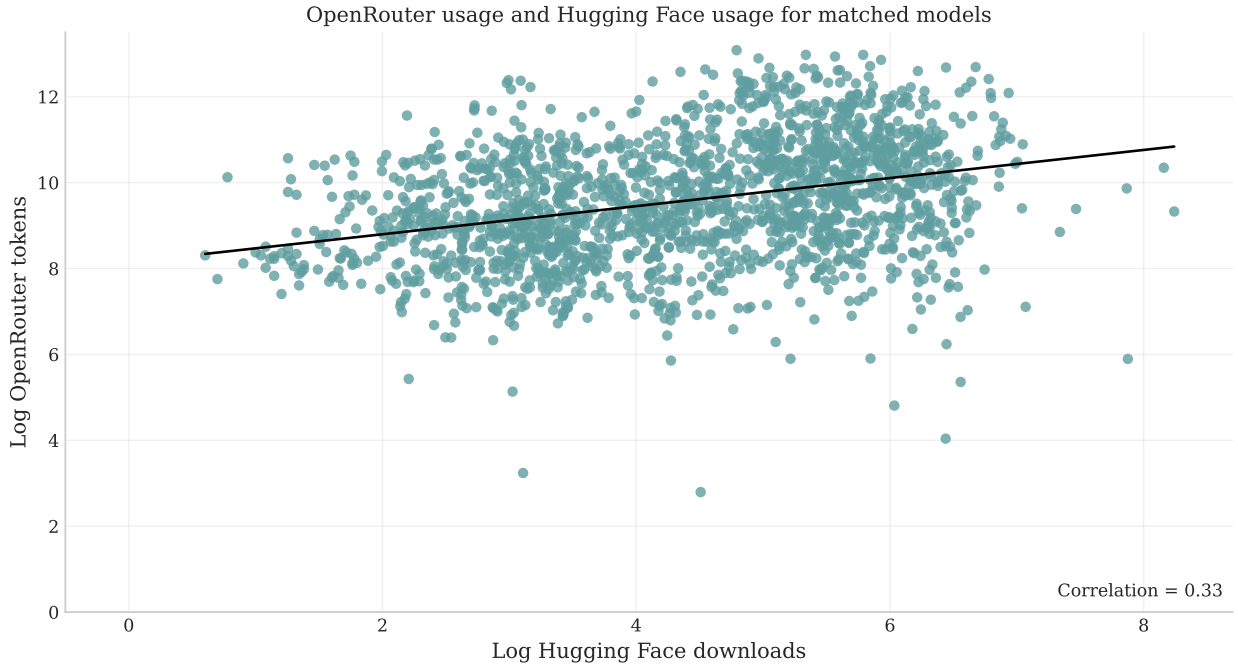
1. For each OpenRouter snapshot, we extract the Hugging Face model ID when OpenRouter reports one. In the raw data this appears in fields such as `hf_slug` or `hugging_face_id`.

2. We standardize OpenRouter model IDs before propagating matches. In particular, we drop variant suffixes that appear after a colon and strip trailing date stamps of the form "-YYYYMMDD." We assign the same Hugging Face model ID to all OpenRouter observations that share this cleaned OpenRouter ID.

3. After constructing the daily OpenRouter panel, we keep observations with a non-missing Hugging Face ID and aggregate the OpenRouter data to one observation per Hugging Face model ID and date.

4. Finally, we split the Hugging Face model ID into author and model name and merge the OpenRouter panel to the weekly Hugging Face panel by author, model name, and date. Once a Hugging Face model is matched in one week, we carry that match across all weeks for that model in the Hugging Face panel.

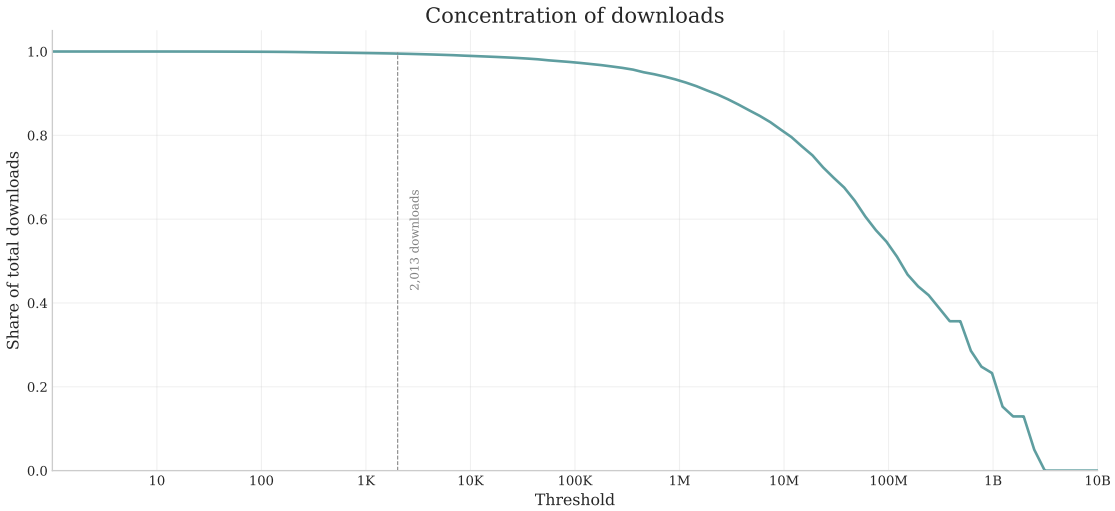
Figure A.1: Matched-model cumulative Hugging Face downloads and cumulative OpenRouter token use



Note: Plots 1,916 model-month observations for 251 models with non-zero download and token counts on Hugging Face and OpenRouter.

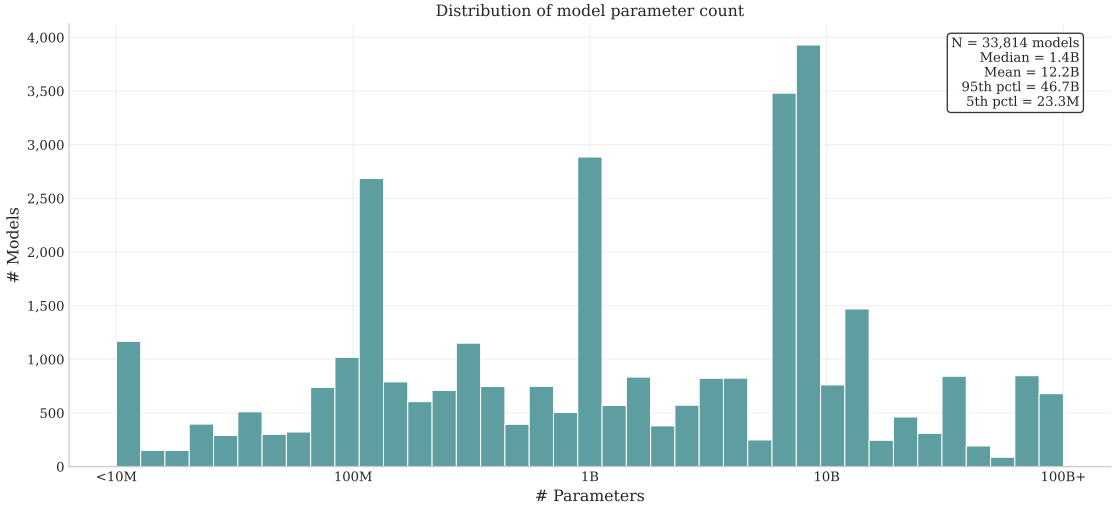
A.3 Additional figures

Figure A.2: Downloads concentration



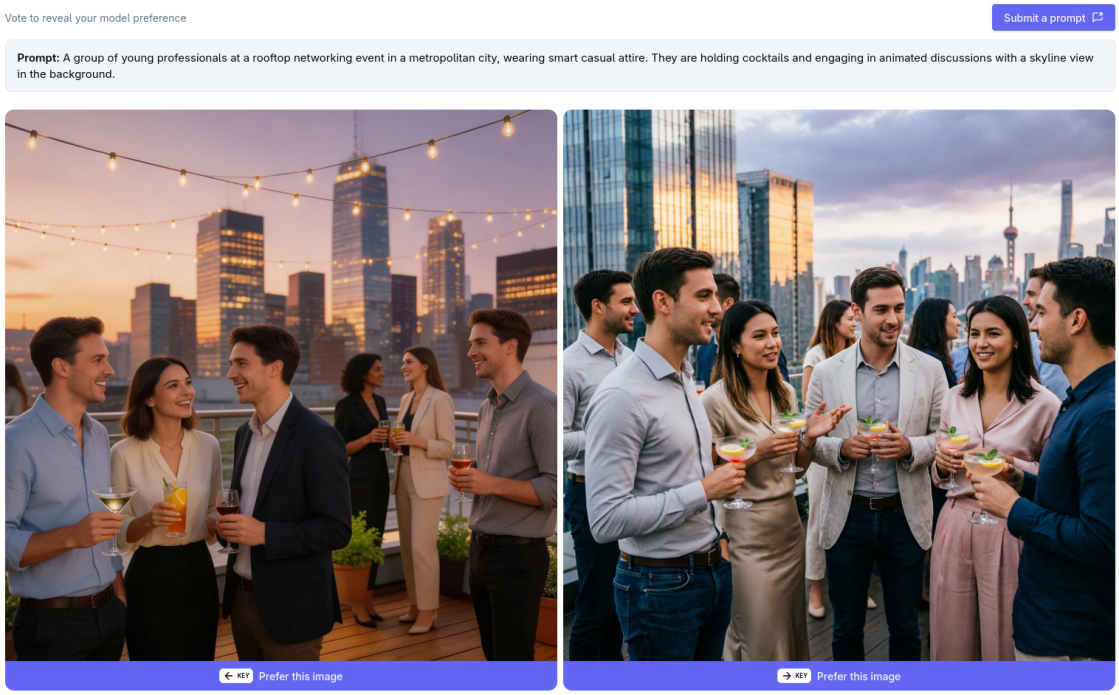
Note: plots the share of total downloads accounted for by models with more than M downloads for increasing values of M.
Models with more than 2,013 downloads account for 99.5% of all downloads on Hugging Face.

Figure A.3: Distribution of model size



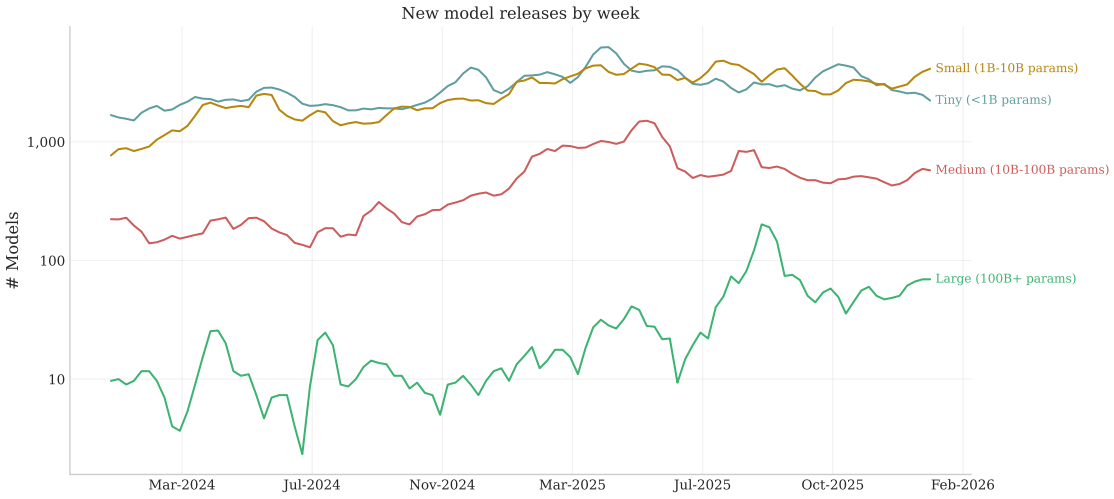
Note: Excludes 74,260 models with no reported parameter count.

Figure A.4: Artificial Analysis Image Arena



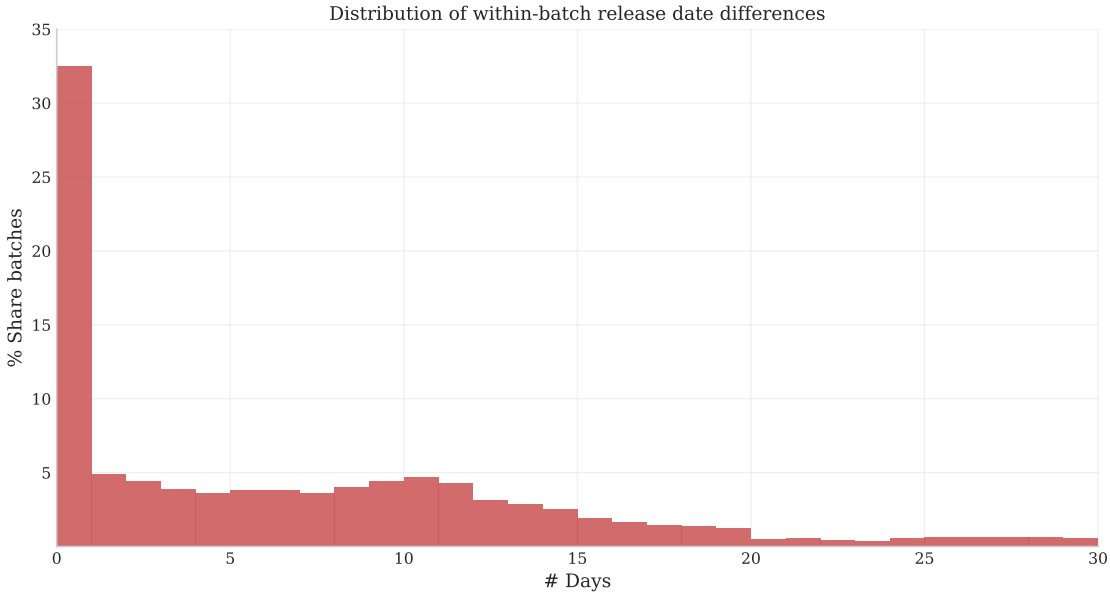
B Birth

Figure B.1: New model releases by week: all Hugging Face models



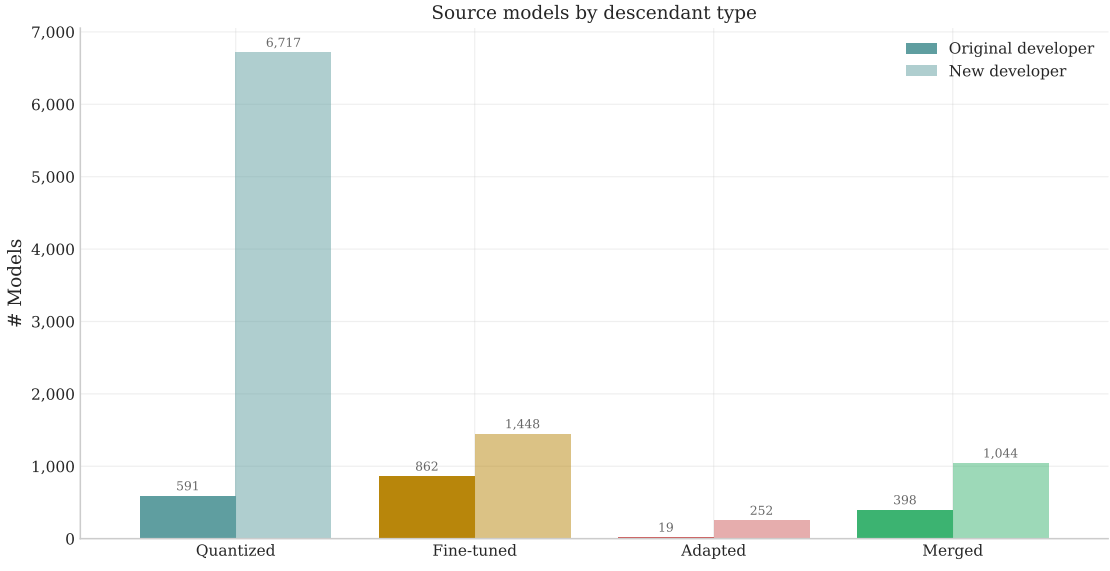
Note: new models are models uploaded to Hugging Face for the first time in a given week. Plots 3-week rolling average.

Figure B.2: Within-batch release date differences



Note: Plots the distribution of the mean pairwise difference between model release dates in each batch.
Batches are groups of models released by the same author within 30 days of each other.
Includes batches with at least two models.

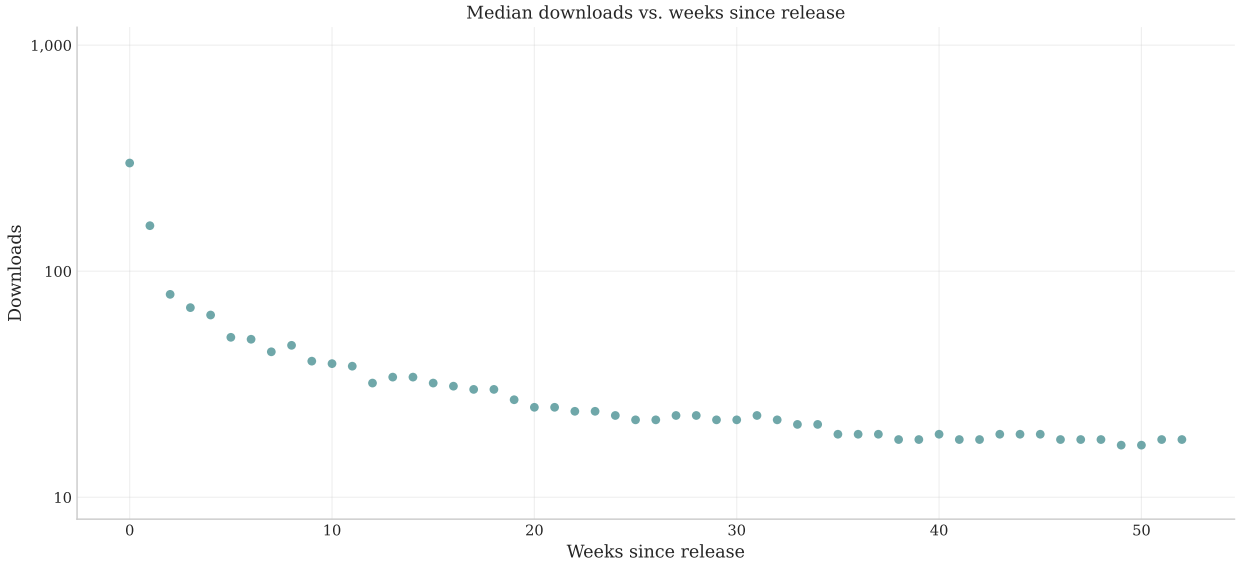
Figure B.3: Source models by descendant type



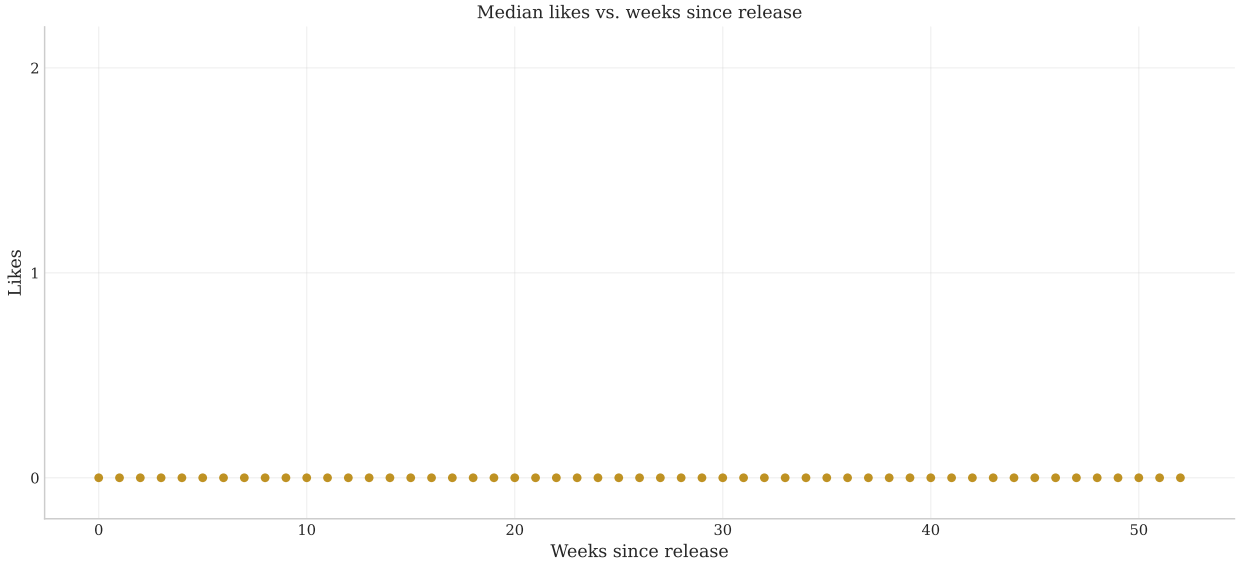
Note: excludes 99,601 models with no descendants.

C Life

Figure C.1: Demand and attention at model release



Note: each dot shows median downloads for models at that week since release.



Note: each dot shows median likes for models at that week since release.

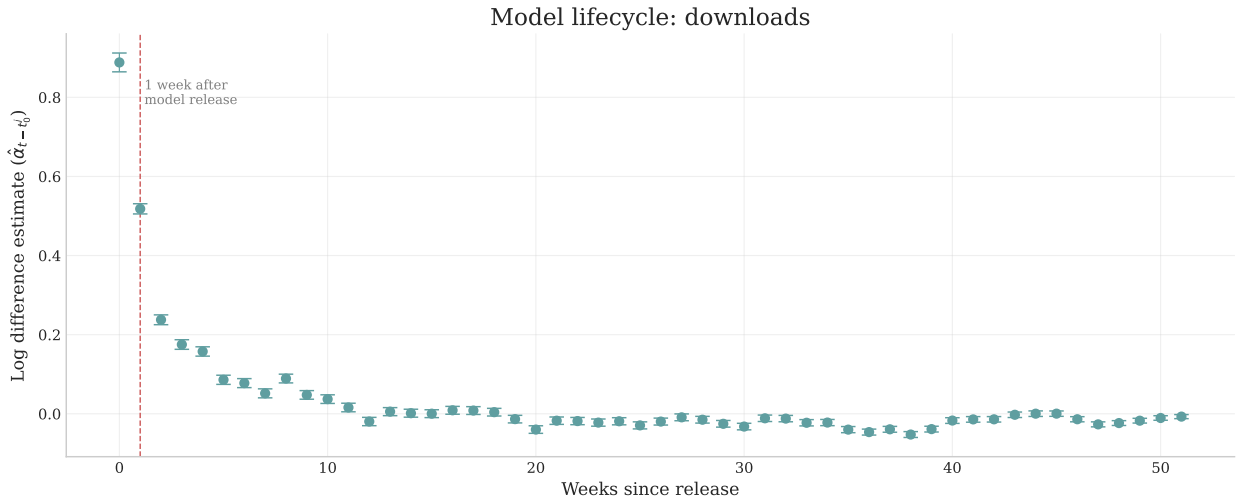
Although major model releases generate a lot of hype, it remains unclear whether the hype translates into actual usage that lasts beyond the initial release. When estimating this "hype-cycle" versus the lifecycle, simply averaging likes or downloads across models would mask important variation and trends. Instead, we allow for model age to more flexibly impact attention and demand. We control for model-specific features since models might have features that affect their use but do not reflect true utility, like better branding which attracts users but does not affect model performance. We also control for calendar time to address

broad trends in model use and adoption over time. For example, Hugging Face and large language models in general became more popular over our sample period, and the release of DeepSeek-R1 represented a unique moment in the history of transformer models. We want our picture of the model lifecycle to capture features inherent to the models rather than picking up broader trends in the data. We finally control for the day of week on which a model was released to address, for example, the possibility that Hugging Face users download models for work, in which case downloads would be less likely on weekends. Our full specification is the following:

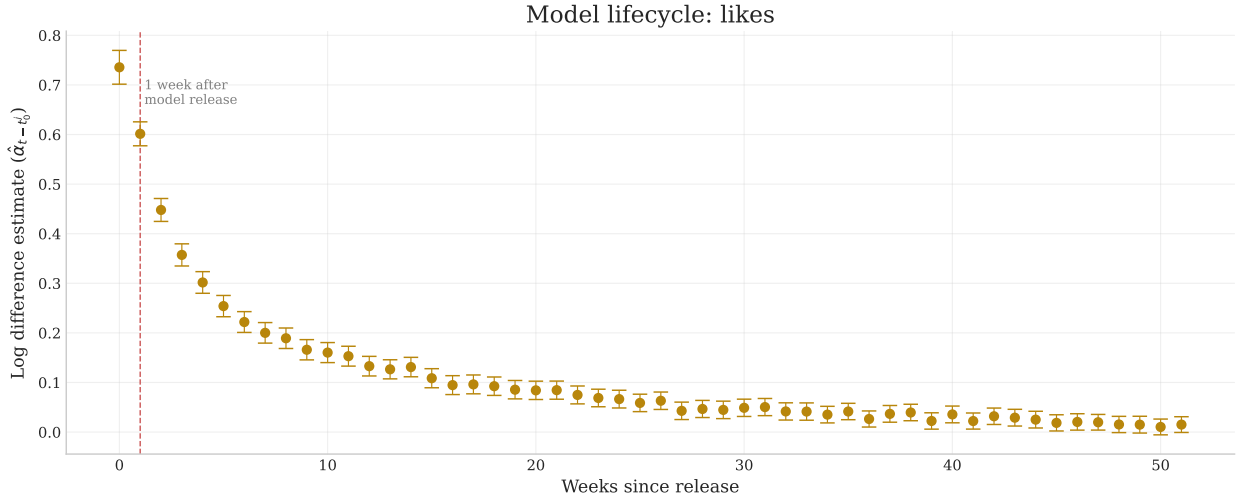
$$y_{jt} = \alpha_{t-t_0^j} + \alpha_{53}\mathbb{1}(t - t_0^j > 52) + \zeta_{dow}\mathbb{1}(t = t_0^j) + \xi_j + \delta_t + \varepsilon_{jt} \tag{C.1}$$

where the outcome is $y_{jt} = \log_{10} p_{jt}$ and p_{jt} is the number of downloads model j received in week t, and t_0^j indexes the calendar week in which model j was released. Our parameters of interest are the $\alpha_{t-t_0^j}$ terms, which are model age fixed effects for each $t - t_0^j \in \{0, 1, \ldots, 51\}$. The specification also includes model fixed effects ξ_j, calendar week fixed effects δ_t, and a day-of-week-of-release fixed effect ζ_{dow}. We cluster standard errors at the model level, and the excluded (baseline) model age is 52 ($\alpha_{52} = 0$). Because of the length of our sample, estimates of the model lifecycle beyond one year of release become increasingly noisy. We therefore focus our analysis on the first year of a model's life; this explains our choice to pool the model age variables beyond week 52.

Figure C.2: Demand and attention at model release



Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly downloads.
Parameter of interest: Coefficient on week since release.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log downloads relative to excluded week (52).



Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly likes.
Parameter of interest: Coefficient on week since release.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log likes relative to excluded week (52).

C.1 Lifecycle analysis by popularity

We estimate the following specification:

$$y_{jt} = \sum_{k=1}^{K} \left(\alpha_{t-t_0^j} + \alpha_{53} \mathbb{1}(t - t_0^j > 52) \right) \mathbb{1}(j \in Q_k) + \zeta_{dow} \mathbb{1}(t = t_0^j) + \xi_j + \delta_t + \varepsilon_{jt} \tag{C.2}$$

where

- $y_{jt} = \log_{10} p_{jt}$ where p_{jt} is the number of downloads model j received in week t.
- t_0^j is the calendar week in which model j was released.
- $\alpha_{t-t_0^j}$ is a model age fixed effect for each $t - t_0^j \in \{0, 1, \ldots, 51\}$.
- Q_k is the set of models that fall in a particular percentile of week-1 downloads when compared to other models at the same calendar time.
- ζ_{dow} is a day of week of release fixed effect.
- ξ_j is a model fixed effect.
- δ_t is a calendar week fixed effect.
- Standard errors are clustered at the model level.
- The excluded model age is α_{52}.

Note that there are actually multiple excluded fixed effects in Equation C.2, namely $\alpha_{52}\mathbb{1}(j \in Q_k)$ for each k. To understand this, first write a more explicit version of the regression equation without any excluded variables

$$y_{jt} = \sum_{k=1}^{K} \left(\sum_{\tau=0}^{52} \alpha_\tau \mathbb{1}(t - t_0^j = \tau) + \alpha_{53} \mathbb{1}(t - t_0^j > 52) \right) \mathbb{1}(j \in Q_k) + \zeta_{dow} \mathbb{1}(t = t_0^j) + \xi_j + \delta_t + \varepsilon_{jt} \tag{C.3}$$

which, without any excluded variables, collapses to

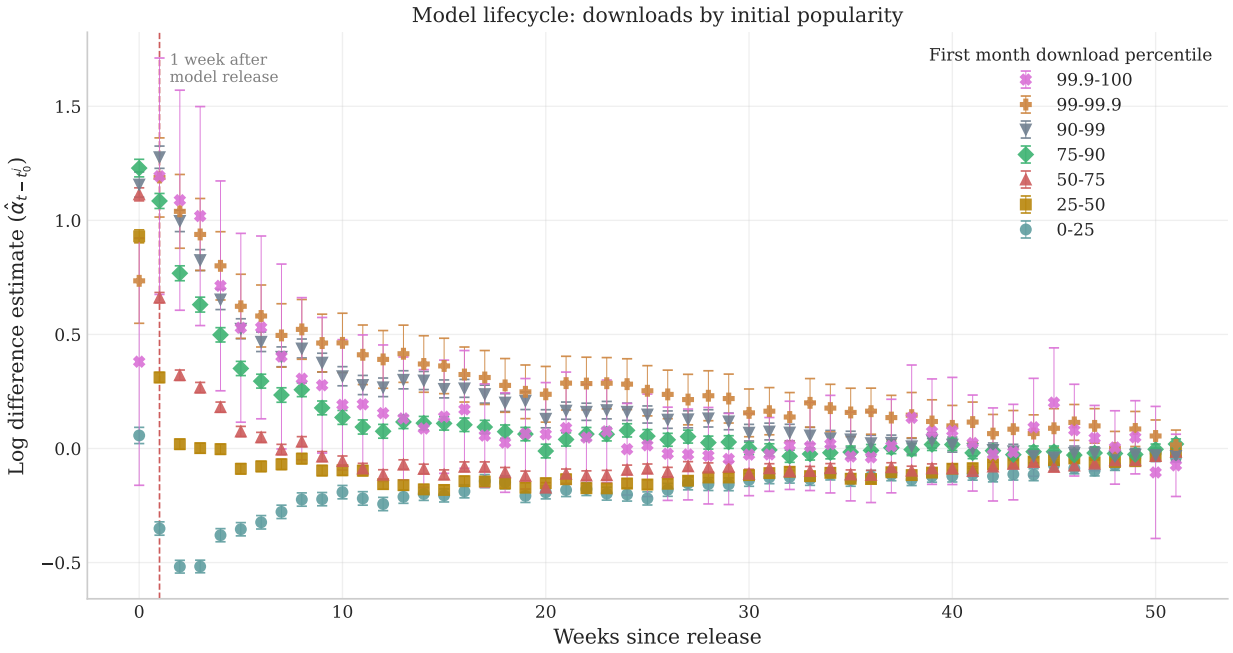
$$y_{jt} = \sum_{k=1}^{K} \alpha_k \mathbb{1}(j \in Q_k) + \zeta_{dow} \mathbb{1}(t = t_0^j) + \xi_j + \delta_t + \varepsilon_{jt}$$

This equation cannot be estimated because the indicator $\mathbb{1}(j \in Q_k)$ is equal to the sum of the model fixed effects ξ_j over all $j \in Q_k$. Now suppose we exclude a single variable, say $\alpha_{52}\mathbb{1}(t - t_0^j = 52)\mathbb{1}(j \in Q_K)$. Then Equation C.3 collapses to

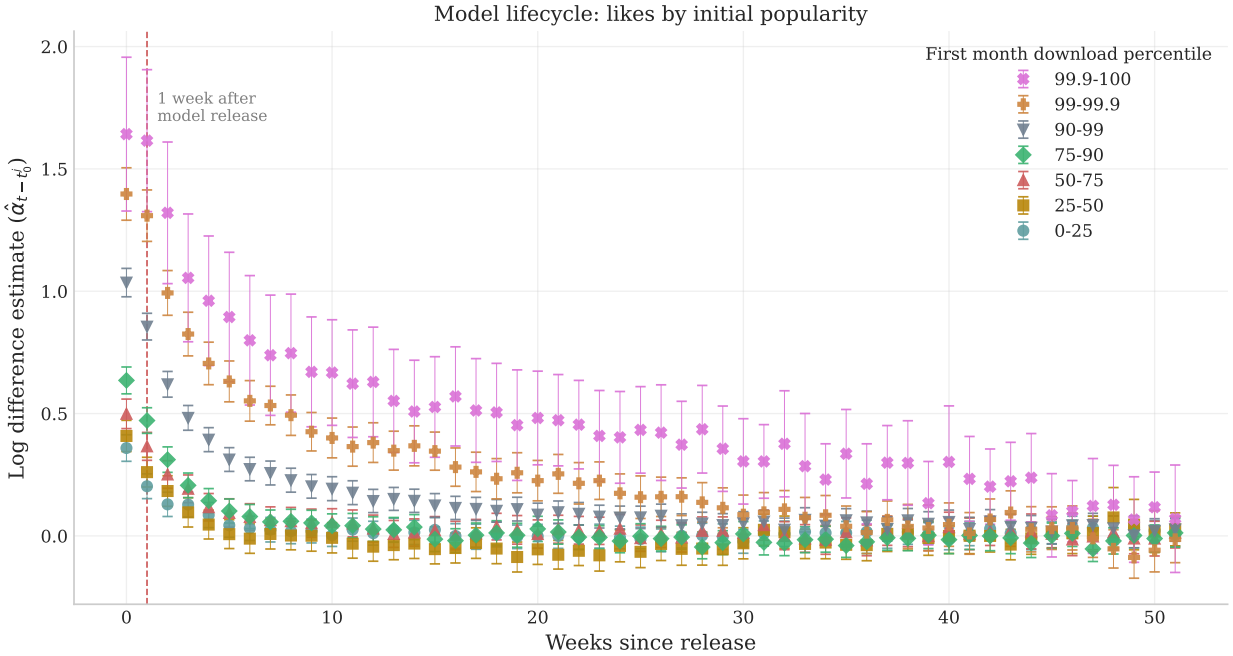
$$y_{jt} = \sum_{k=1}^{K-1} \alpha_k \mathbb{1}(j \in Q_k) + \left(\sum_{\tau=0}^{51} \alpha_\tau \mathbb{1}(t - t_0^j = \tau) + \alpha_{53} \mathbb{1}(t - t_0^j > 52) \right) \mathbb{1}(j \in Q_K) + \zeta_{dow} \mathbb{1}(t = t_0^j) + \xi_j + \delta_t + \varepsilon_{jt}$$

This eliminates the collinearity problem for models $j \in Q_K$, but not for $j \in Q_1, \ldots, Q_{K-1}$. Therefore we need to exclude $\alpha_{52}\mathbb{1}(t - t_0^j = 52)\mathbb{1}(j \in Q_k)$ for *all* $k \in \{1, \ldots, K\}$.

Figure C.3: Demand and attention at model release



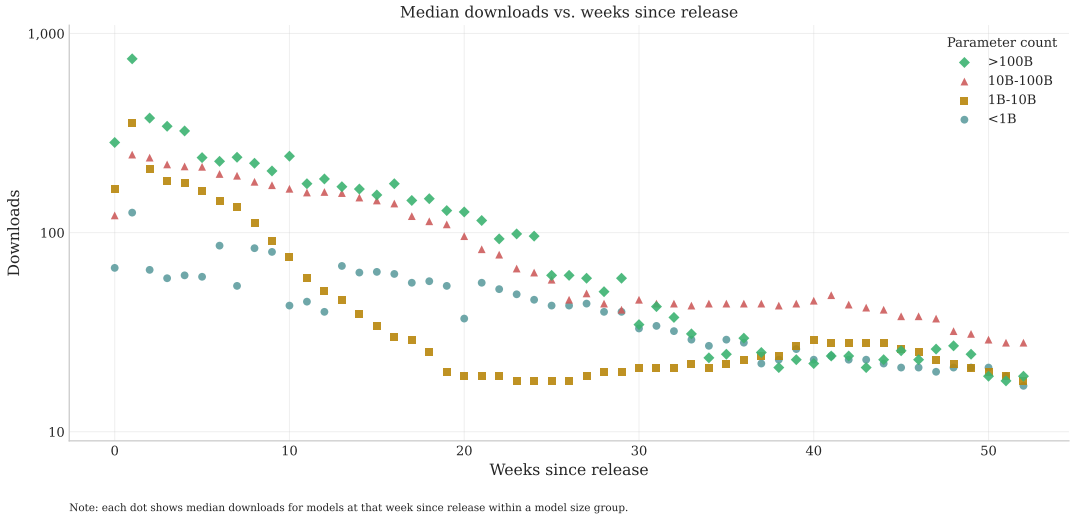
Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly downloads.
Parameter of interest: Coefficient on week since release interacted with first month download percentile.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log downloads relative to excluded week (52).
Models with zero downloads in the first 4 weeks are included as a separate group in the regression but omitted from the plot.



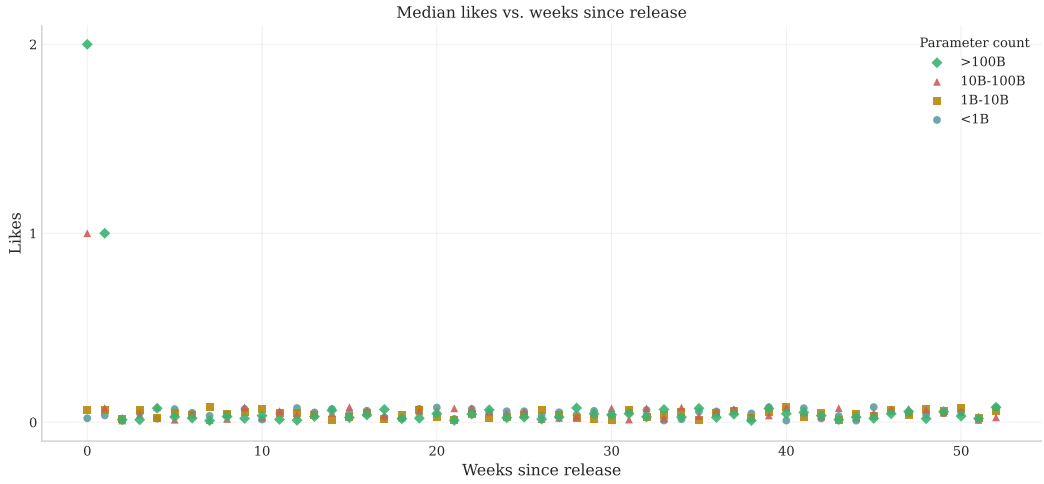
Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly likes.
Parameter of interest: Coefficient on week since release interacted with first month download percentile.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log likes relative to excluded week (52).
Models with zero downloads in the first 4 weeks are included as a separate group in the regression but omitted from the plot.

C.2 Lifecycle analysis by model size

Figure C.4: Median model lifecycle by size

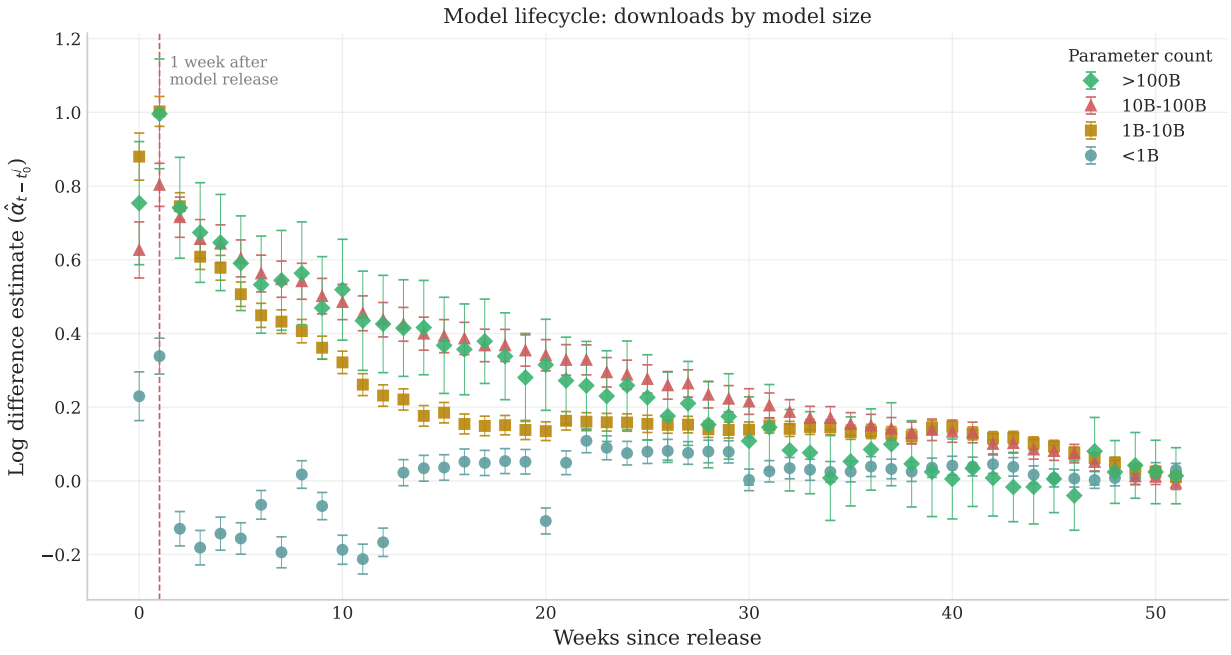


Note: each dot shows median downloads for models at that week since release within a model size group.

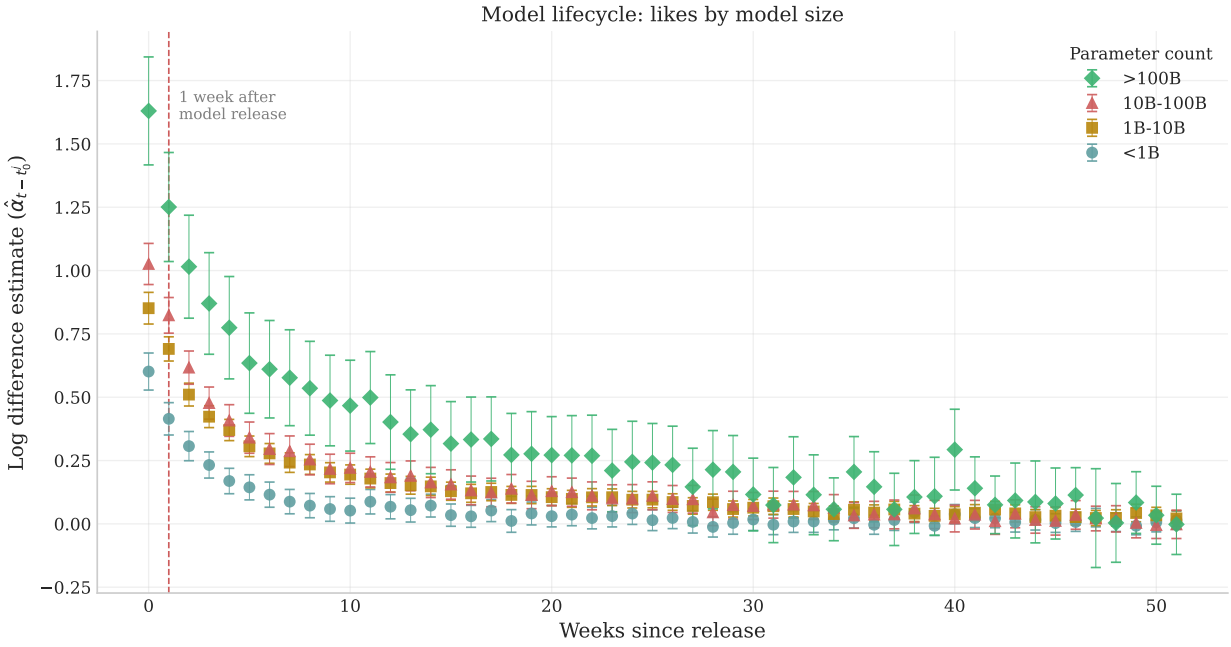


Note: dots show median likes by week and model size group. Zero medians are jittered upward.

Figure C.5: Demand and attention at model release



Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly downloads.
Parameter of interest: Coefficient on week since release interacted with model size group.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log downloads relative to excluded week (52).

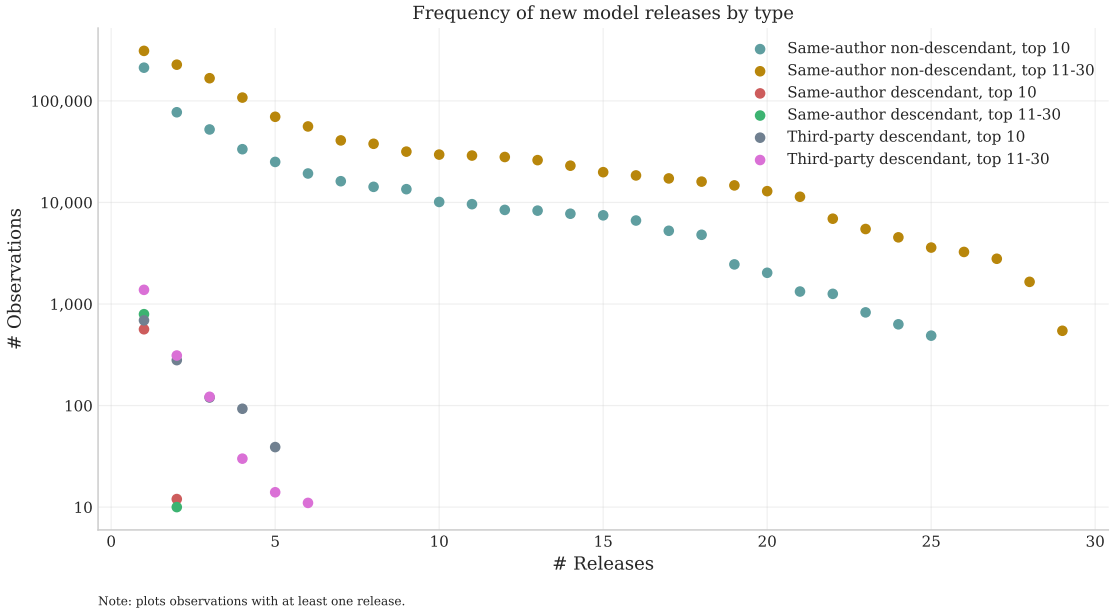


Note: Plots parameter estimates from the following linear regression:
Outcome: Base 10 log of weekly likes.
Parameter of interest: Coefficient on week since release interacted with model size group.
Additional controls: Day of week of release, calendar week fixed effects, model fixed effects.
Vertical axis shows change in log likes relative to excluded week (52).

C.3 Spillover effects

Here we present additional analysis of spillover effects from Equation 4.1.

Figure C.6: Subsequent model release counts



Note: plots observations with at least one release.

Table C.1: Spillover effect estimates for same-author descendant models

	Downloads			Likes		
	Full	Top 10%	Top 1%	Full	Top 10%	Top 1%
Top 10						
1 release (recent)	0.4036	0.3662	0.2153	-0.1706	-0.1971	0.2741
	(0.2757)	(0.2921)	(0.3339)	(0.2519)	(0.2723)	(0.2081)
2 releases (recent)	-0.9902***			1.017***		
	(0.3372)			(0.3206)		
1 release (mature)	0.0198	0.0828	-0.0451	-0.3688**	-0.3842**	-0.4854***
	(0.1998)	(0.2135)	(0.2843)	(0.1453)	(0.1578)	(0.1654)
2 releases (mature)	1.711***			-1.368***		
	(0.2487)			(0.2329)		
Top 11–30						
1 release (recent)	-0.1694	-0.1266	0.2402	0.8127***	0.8173***	0.3106
	(0.4354)	(0.4359)	(0.6210)	(0.1615)	(0.1504)	(0.2166)
2 releases (recent)	-2.722***	-2.707***		-8.936***	-8.814***	
	(0.5161)	(0.5038)		(0.2366)	(0.2319)	
1 release (mature)	0.3735	0.3431	-0.5710	-0.4841**	-0.5681***	-0.3635
	(0.4426)	(0.4448)	(0.7872)	(0.1994)	(0.1606)	(0.3636)
2 releases (mature)	0.7938	0.7706		8.211***	8.346***	
	(0.4991)	(0.4879)		(0.1290)	(0.1326)	
Model	Yes	Yes	Yes	Yes	Yes	Yes
Date (week)	Yes	Yes	Yes	Yes	Yes	Yes
Age*initial popularity	Yes	Yes	Yes	Yes	Yes	Yes
Observations	6,749,333	634,679	57,536	2,178,219	410,241	43,400
Pseudo R^2	0.91381	0.89629	0.90198	0.76996	0.82161	0.86925

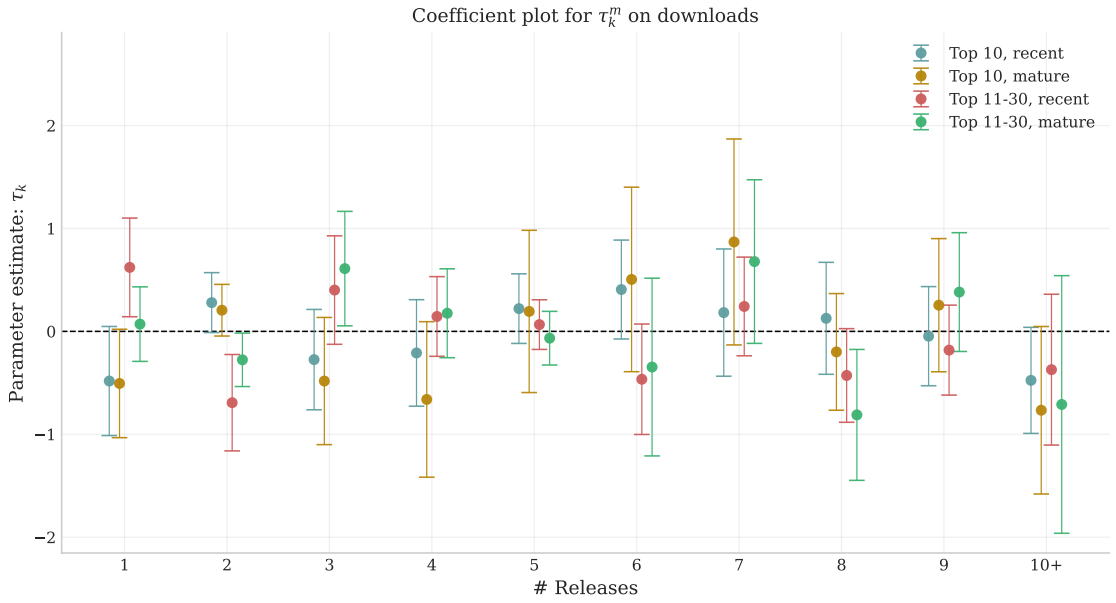
Standard errors in parentheses clustered at the model level
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Table C.2: Spillover effect estimates for same-author non-descendant models

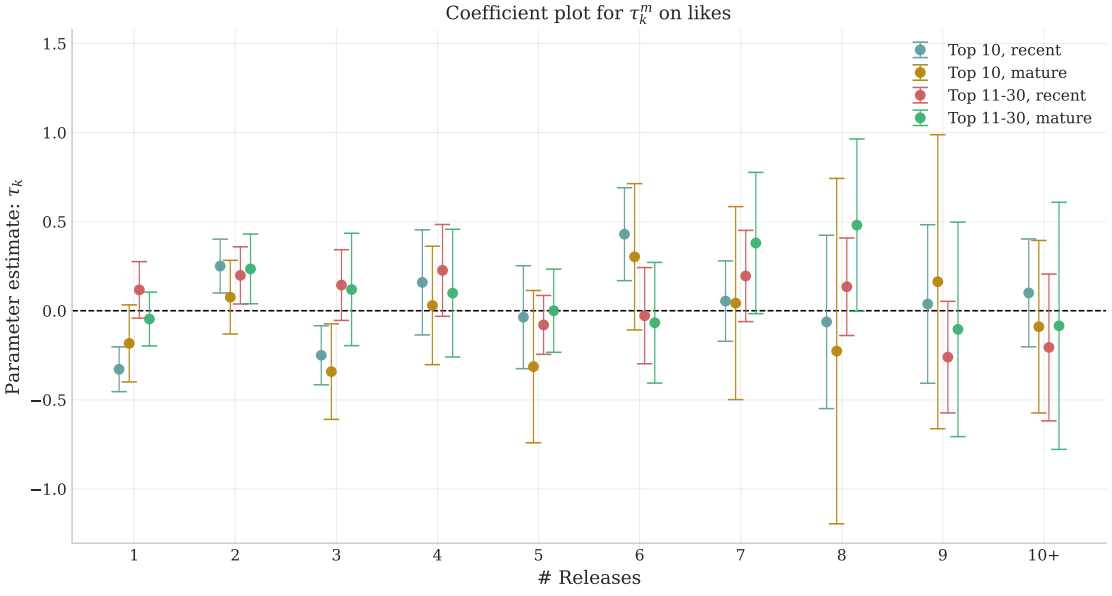
	Downloads			Likes		
	Full	Top 10%	Top 1%	Full	Top 10%	Top 1%
Top 10						
1 release (recent)	-0.2126	-0.2672	-0.4823*	-0.1697***	-0.1672***	-0.3287***
	(0.1903)	(0.2028)	(0.2704)	(0.0467)	(0.0511)	(0.0641)
2 releases (recent)	0.1380	0.1866*	0.2797*	0.1516**	0.2199***	0.2506***
	(0.1036)	(0.1083)	(0.1485)	(0.0615)	(0.0640)	(0.0771)
3 releases (recent)	-0.1274	-0.1715	-0.2743	-0.1370**	-0.2167***	-0.2501***
	(0.1652)	(0.1726)	(0.2487)	(0.0678)	(0.0702)	(0.0846)
4 releases (recent)	-0.2590	-0.3751*	-0.2096	0.0488	0.0468	0.1589
	(0.1815)	(0.2096)	(0.2641)	(0.0725)	(0.0839)	(0.1504)
5 releases (recent)	0.2107	0.3982**	0.2211	0.0145	0.0144	-0.0363
	(0.1412)	(0.1561)	(0.1726)	(0.0796)	(0.0864)	(0.1474)
6 releases (recent)	0.2884**	0.2972**	0.4067*	0.3630***	0.3391***	0.4296***
	(0.1307)	(0.1462)	(0.2452)	(0.1279)	(0.1308)	(0.1331)
7 releases (recent)	0.4093*	0.4419*	0.1826	0.0970	0.0380	0.0540
	(0.2423)	(0.2592)	(0.3154)	(0.1046)	(0.1018)	(0.1152)
8 releases (recent)	-0.2889	-0.2895	0.1269	0.3181*	-0.0833	-0.0628
	(0.1997)	(0.2082)	(0.2775)	(0.1915)	(0.1394)	(0.2482)
9 releases (recent)	0.3922**	0.4016**	-0.0466	-0.1329	0.1962	0.0378
	(0.1676)	(0.1746)	(0.2458)	(0.1771)	(0.1317)	(0.2270)
10+ releases (recent)	-0.2800	-0.2889	-0.4761*	0.0966	0.1256	0.1001
	(0.2195)	(0.2263)	(0.2630)	(0.1183)	(0.1207)	(0.1545)
Log(10+ releases) (recent)	1.287***	1.322***	0.9268*	0.2834	0.2489	0.0820
	(0.4730)	(0.4854)	(0.5227)	(0.2107)	(0.2203)	(0.3555)
1 release (mature)	0.0405	0.0804	-0.0238	-0.1047	-0.0163	0.1452
	(0.1528)	(0.1635)	(0.2135)	(0.0665)	(0.0736)	(0.0933)
2 releases (mature)	-0.1136	-0.1241	-0.0739	-0.1014	-0.1273*	-0.1746**
	(0.0824)	(0.0881)	(0.1006)	(0.0655)	(0.0683)	(0.0759)
3 releases (mature)	-0.2683**	-0.2780**	-0.2082	0.0381	0.0120	-0.0916
	(0.1238)	(0.1296)	(0.1427)	(0.0650)	(0.0701)	(0.1016)
4 releases (mature)	-0.0406	-0.2670	-0.4518**	-0.2476***	-0.1991*	-0.1293
	(0.1599)	(0.1650)	(0.2161)	(0.0908)	(0.1054)	(0.1216)
5 releases (mature)	-0.2813	-0.1552	-0.0272	-0.2480**	-0.1738	-0.2776**
	(0.1773)	(0.1853)	(0.2908)	(0.1057)	(0.1097)	(0.1287)
6 releases (mature)	-0.1511	-0.1492	0.0982	-0.1660*	-0.1847*	-0.1268
	(0.2000)	(0.2109)	(0.2857)	(0.1003)	(0.1074)	(0.1190)
7 releases (mature)	0.5674**	0.5709**	0.6864**	-0.1795	-0.0640	-0.0113
	(0.2394)	(0.2467)	(0.2978)	(0.1420)	(0.1482)	(0.2064)
8 releases (mature)	-0.5150**	-0.6499***	-0.3263	-0.1148	-0.1976	-0.1639
	(0.2278)	(0.2402)	(0.2140)	(0.1706)	(0.1880)	(0.2997)
9 releases (mature)	0.3169	0.4446*	0.3011	-0.0251	0.0430	0.1250
	(0.2103)	(0.2280)	(0.2780)	(0.1183)	(0.1416)	(0.2627)
10+ releases (mature)	-0.1132	-0.0814	-0.2904	-0.1691	-0.1914*	-0.1899
	(0.2027)	(0.2096)	(0.2919)	(0.1074)	(0.1103)	(0.1694)
Log(10+ releases) (mature)	-0.3567	-0.4491	0.4752	0.1499	0.1104	0.2727
	(0.3122)	(0.3416)	(0.4367)	(0.1780)	(0.1910)	(0.3984)
Top 11–30						
1 release (recent)	0.8282***	0.8643***	0.6218**	0.0357	0.0277	0.1169
	(0.2427)	(0.2400)	(0.2446)	(0.0464)	(0.0538)	(0.0809)
2 releases (recent)	-0.2558	-0.2997	-0.6931***	0.1037**	0.1238**	0.1983**
	(0.1767)	(0.1888)	(0.2389)	(0.0517)	(0.0540)	(0.0819)
3 releases (recent)	0.1749	0.2266	0.4015	0.0627	0.0103	0.1440
	(0.1649)	(0.1835)	(0.2689)	(0.0694)	(0.0783)	(0.1013)
4 releases (recent)	0.0604	0.0828	0.1449	0.1812**	0.1805*	0.2263*
	(0.1292)	(0.1487)	(0.1976)	(0.0809)	(0.0997)	(0.1315)
5 releases (recent)	0.0467	0.0205	0.0660	0.0037	0.0169	-0.0797
	(0.0923)	(0.1011)	(0.1231)	(0.0577)	(0.0622)	(0.0843)
6 releases (recent)	-0.2519*	-0.2587	-0.4653*	0.0496	0.0416	-0.0278
	(0.1459)	(0.1670)	(0.2737)	(0.0776)	(0.0807)	(0.1377)
7 releases (recent)	0.3017**	0.3019**	0.2421	0.0102	-0.0152	0.1949
	(0.1256)	(0.1339)	(0.2446)	(0.0971)	(0.1061)	(0.1307)
8 releases (recent)	-0.3824**	-0.3823*	-0.4283*	0.2836***	0.2952***	0.1346
	(0.1944)	(0.2089)	(0.2321)	(0.0974)	(0.1108)	(0.1397)
9 releases (recent)	-0.1711	-0.1660	-0.1821	0.0302	0.0725	-0.2605
	(0.2540)	(0.2713)	(0.2230)	(0.0923)	(0.1041)	(0.1598)
10+ releases (recent)	-0.1098	-0.0948	-0.3719	-0.6418***	-0.6248***	-0.2062
	(0.2096)	(0.2192)	(0.3739)	(0.1252)	(0.1356)	(0.2102)
Log(10+ releases) (recent)	-0.1970	-0.3566	0.2138	0.5281***	0.6461***	0.2652
	(0.4257)	(0.4509)	(0.4814)	(0.1892)	(0.2128)	(0.3653)
1 release (mature)	-1.069**	-1.073***	-0.5511	-0.0969**	-0.0420	-0.1635**
	(0.4216)	(0.4136)	(0.3356)	(0.0421)	(0.0488)	(0.0768)
2 releases (mature)	0.5591**	0.5112*	0.4167*	0.0675	0.0547	0.0364
	(0.2748)	(0.2747)	(0.2460)	(0.0427)	(0.0479)	(0.0723)
3 releases (mature)	0.1515	0.1763	0.2086	-0.0792	-0.0444	-0.0247
	(0.1379)	(0.1540)	(0.2289)	(0.0563)	(0.0712)	(0.1180)
4 releases (mature)	-0.0558	-0.1206	0.0312	0.0622	-0.0309	-0.1276
	(0.1489)	(0.1646)	(0.1970)	(0.0739)	(0.0900)	(0.1384)
5 releases (mature)	0.0272	-0.0308	-0.1320	0.0860	0.0964	0.0797
	(0.0874)	(0.0910)	(0.1274)	(0.0757)	(0.0877)	(0.1024)
6 releases (mature)	0.2684	0.4115	0.1187	0.0103	-0.0104	-0.0398
	(0.3103)	(0.3218)	(0.2813)	(0.0698)	(0.0777)	(0.1081)
7 releases (mature)	0.2719	0.2369	0.4363**	-0.0988	-0.0240	0.1851
	(0.2008)	(0.2069)	(0.2220)	(0.0809)	(0.0868)	(0.1263)
8 releases (mature)	-0.2194**	-0.1926*	-0.3832**	0.2323**	0.2156**	0.3459**
	(0.0927)	(0.0990)	(0.1906)	(0.0983)	(0.1074)	(0.1732)
9 releases (mature)	0.1664	0.1683	0.5640***	0.2963***	0.2447**	0.1558
	(0.1402)	(0.1446)	(0.1958)	(0.1107)	(0.1180)	(0.1769)
10+ releases (mature)	-0.3382	-0.3101	-0.3383	-0.2157*	-0.2315	0.1213
	(0.2202)	(0.2317)	(0.3035)	(0.1268)	(0.1415)	(0.1922)
Log(10+ releases) (mature)	0.3659	0.4275	-1.059***	-0.1513	-0.2200	-1.209***
	(0.3479)	(0.3738)	(0.3049)	(0.1758)	(0.2014)	(0.3211)
Model	Yes	Yes	Yes	Yes	Yes	Yes
Date (week)	Yes	Yes	Yes	Yes	Yes	Yes
Age*initial popularity	Yes	Yes	Yes	Yes	Yes	Yes
Observations	6,749,333	634,679	57,536	2,178,219	410,241	43,400
Pseudo R^2	0.91381	0.89629	0.90198	0.76996	0.82161	0.86925

Standard errors in parentheses clustered at the model level
*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Figure C.7: Spillover event study: same-author non-descendant releases, top 1% of focal models

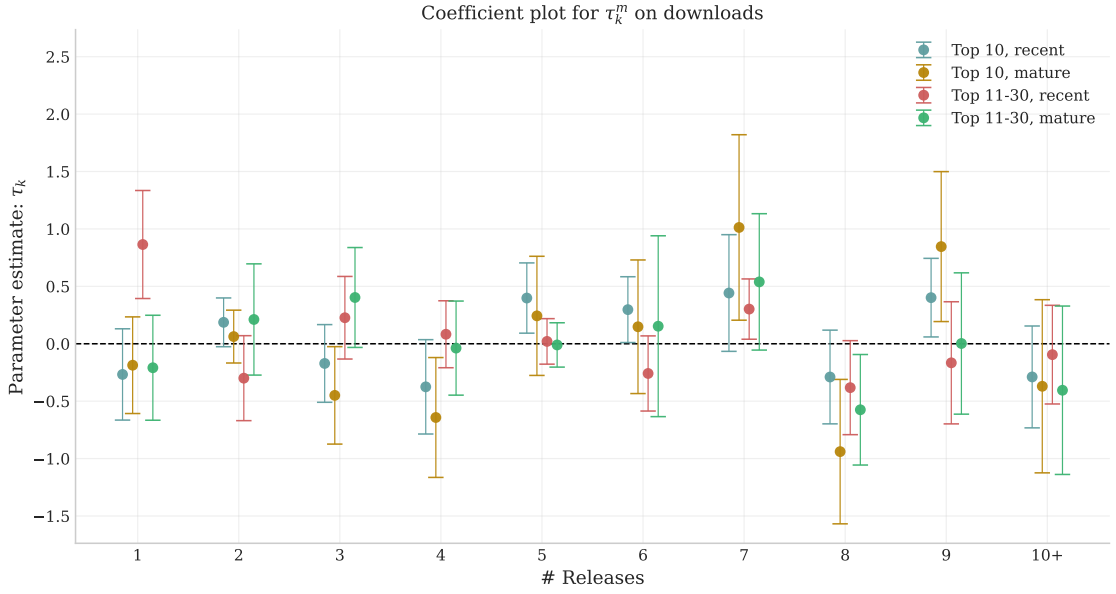


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

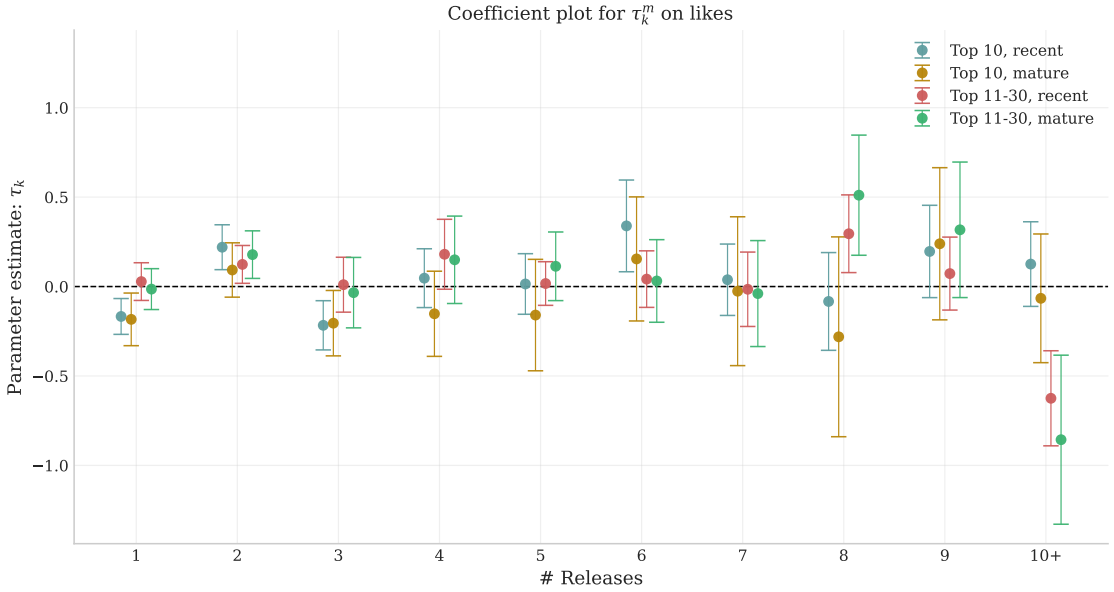


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

Figure C.8: Spillover event study: same-author non-descendant releases, top 10% of focal models

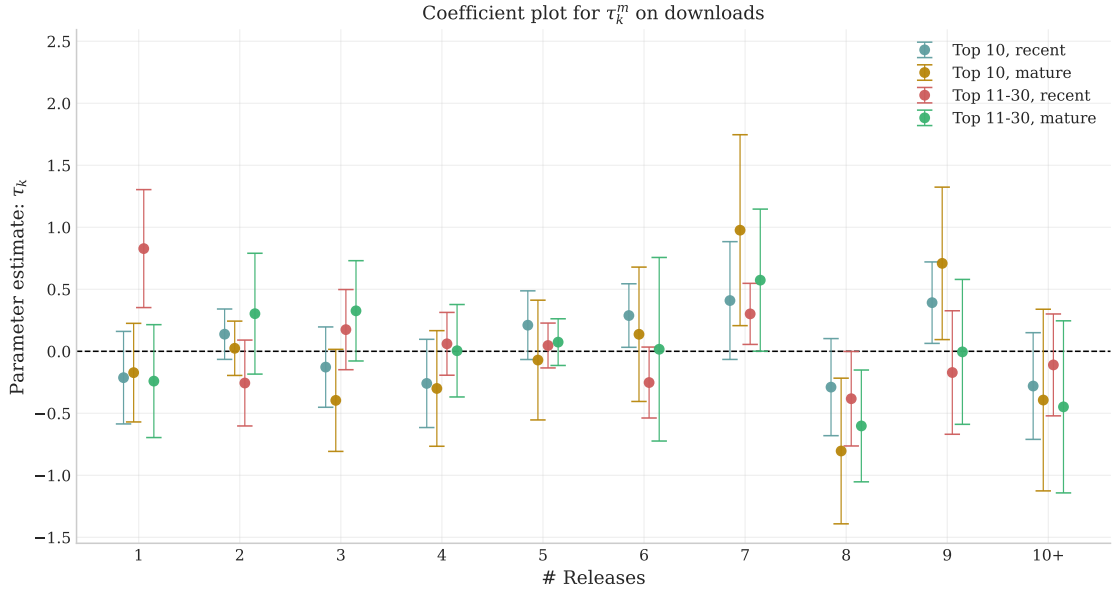


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.

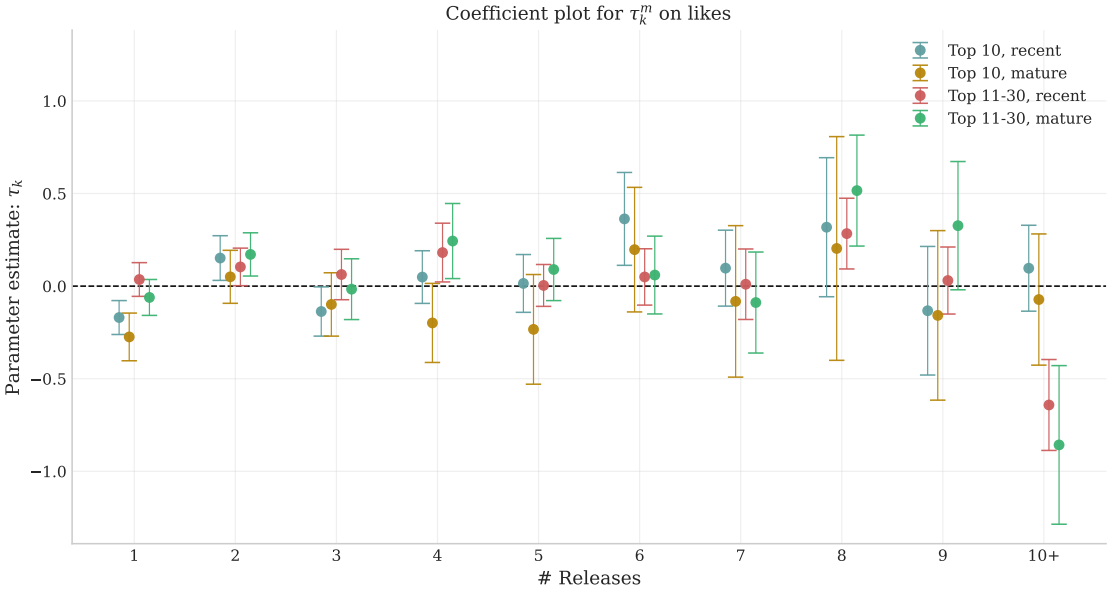


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.

Figure C.9: Spillover event study: same-author non-descendant releases, full sample

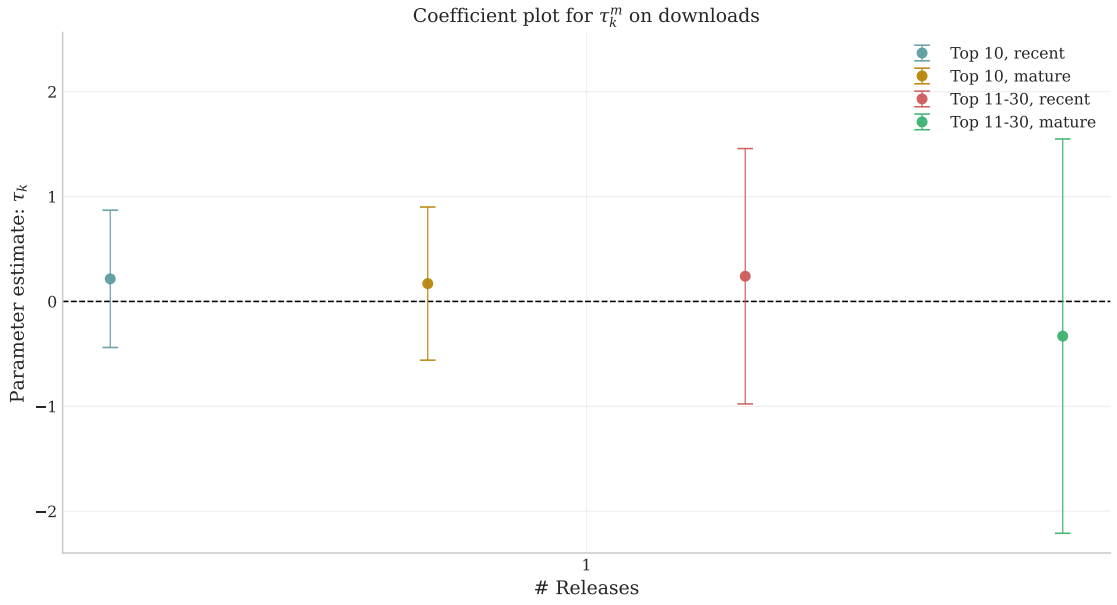


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects.

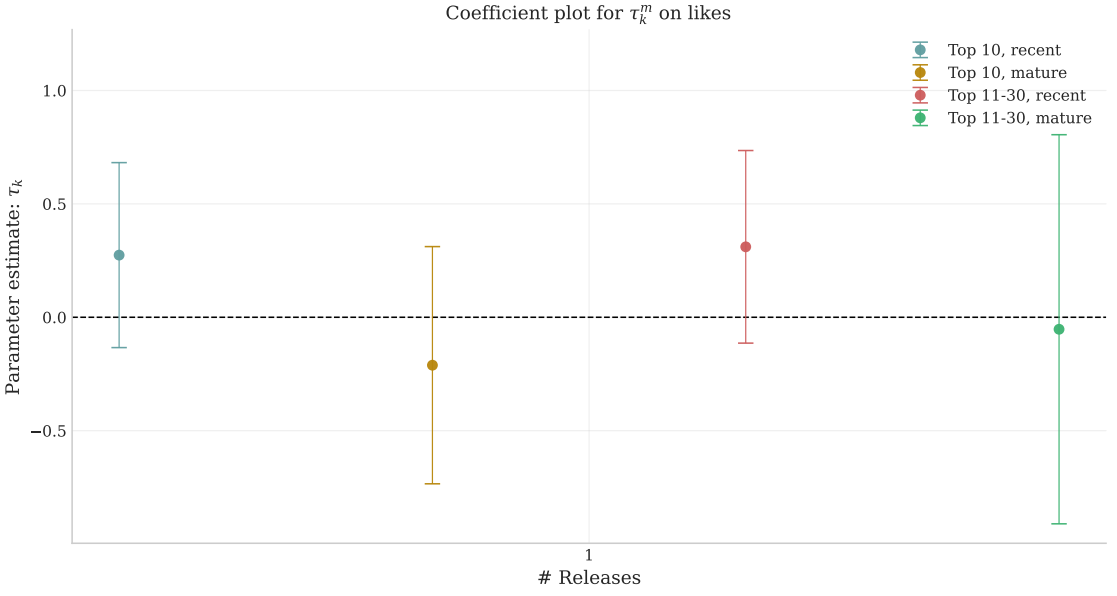


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects.

Figure C.10: Spillover event study: same-author descendant releases, top 1% of focal models

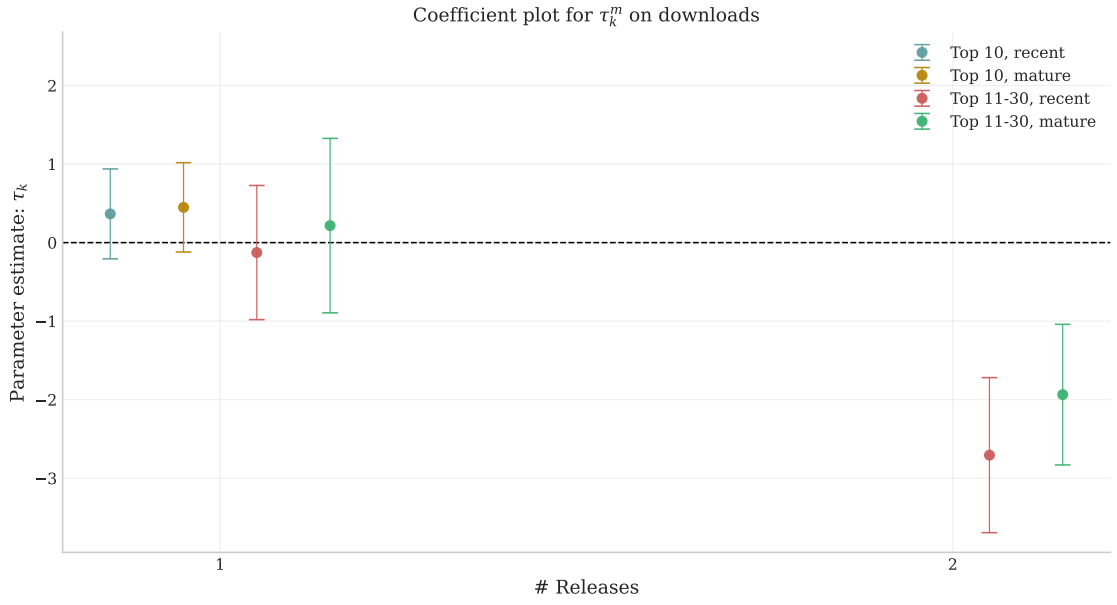


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

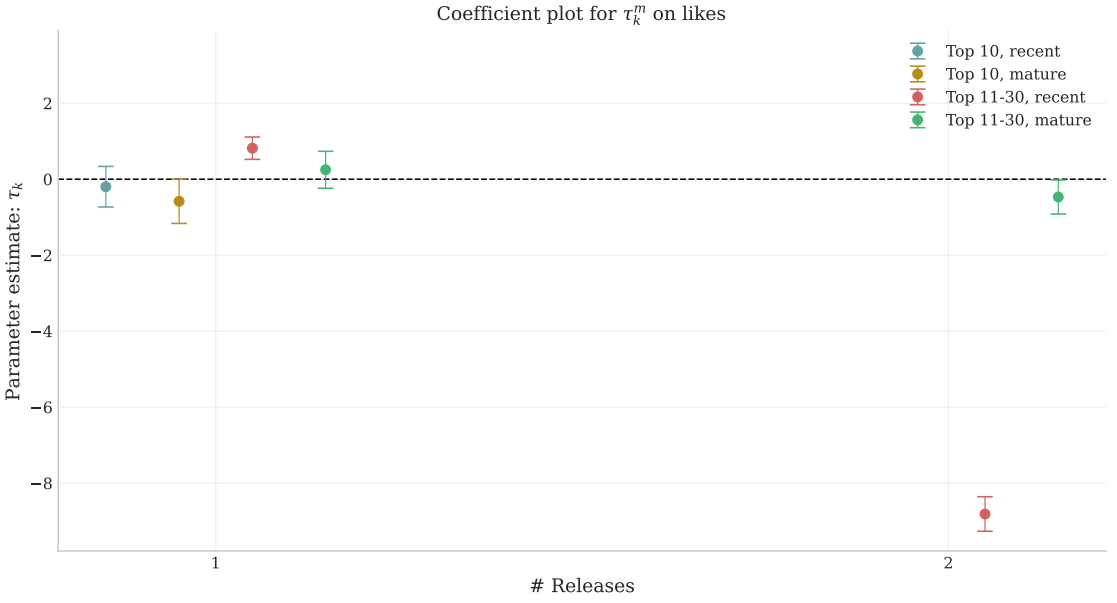


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

Figure C.11: Spillover event study: same-author descendant releases, top 10% of focal models

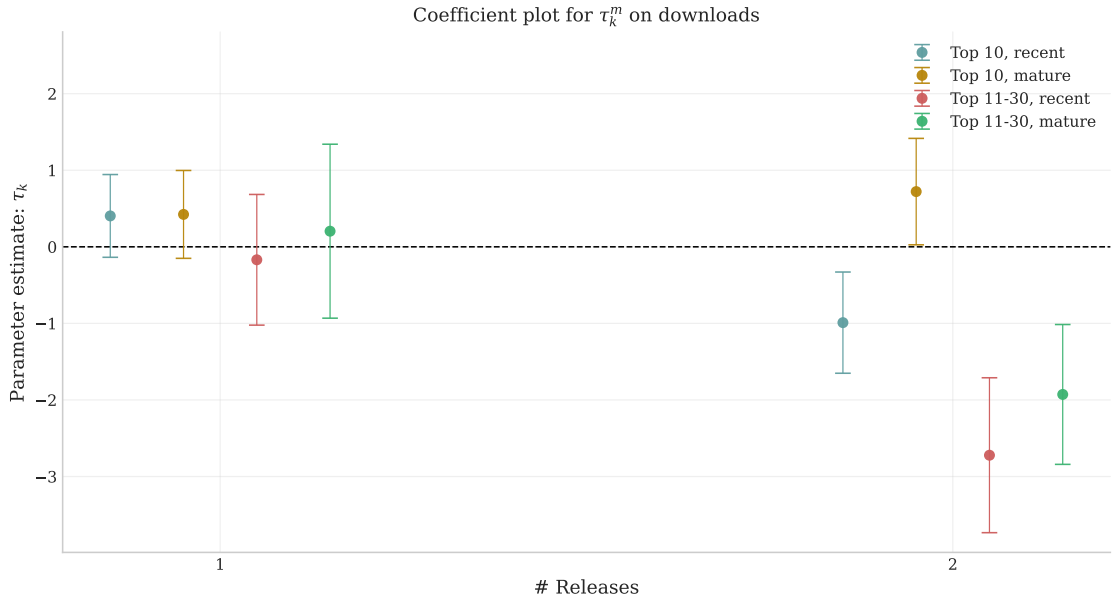


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.

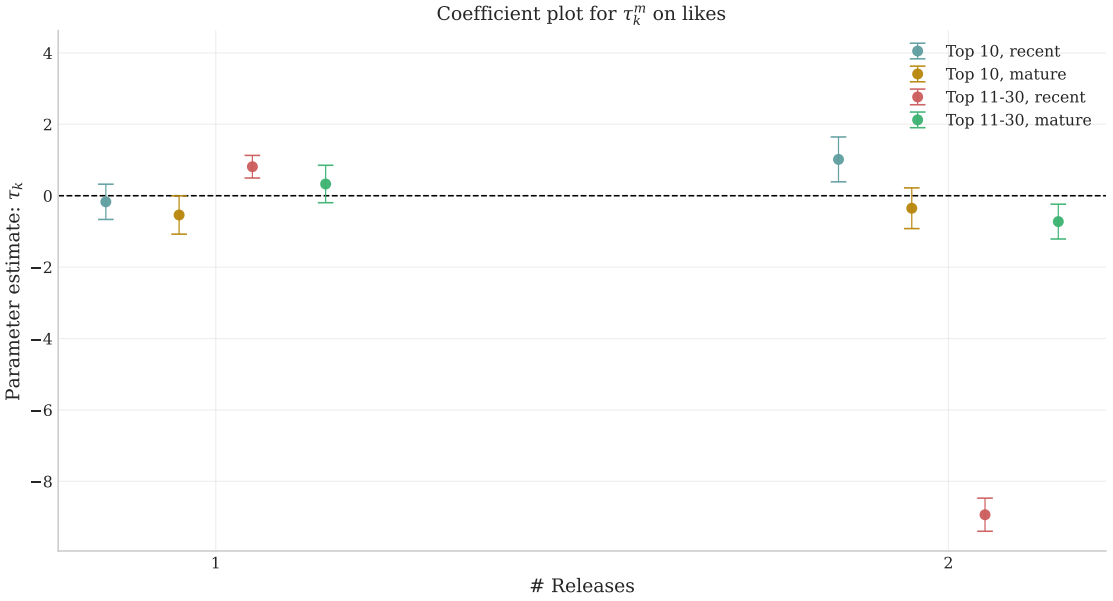


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.

Figure C.12: Spillover event study: same-author descendant releases, full sample

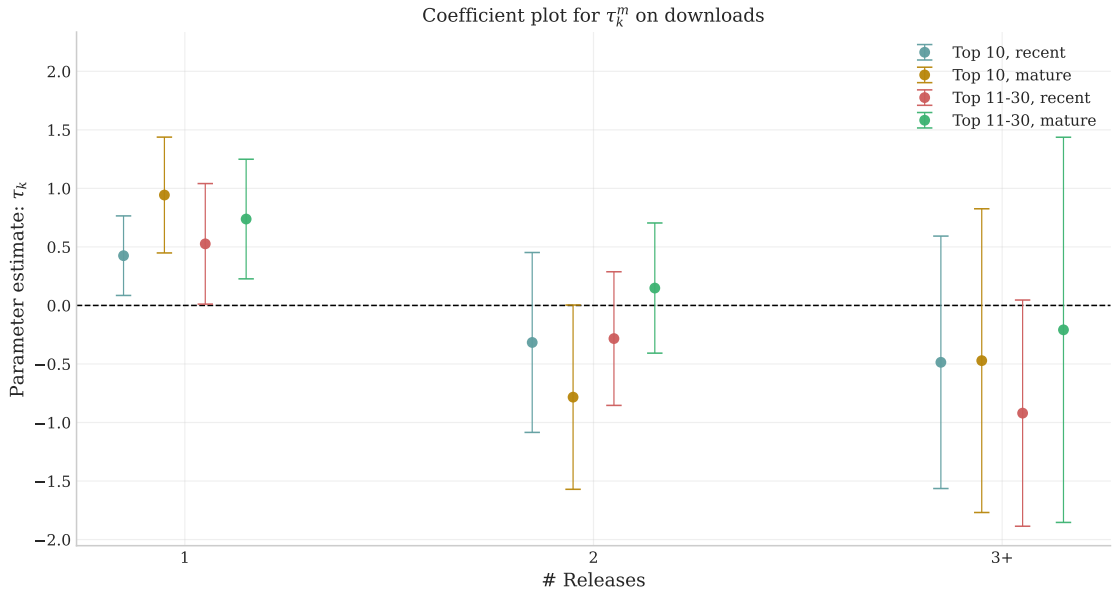


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects.

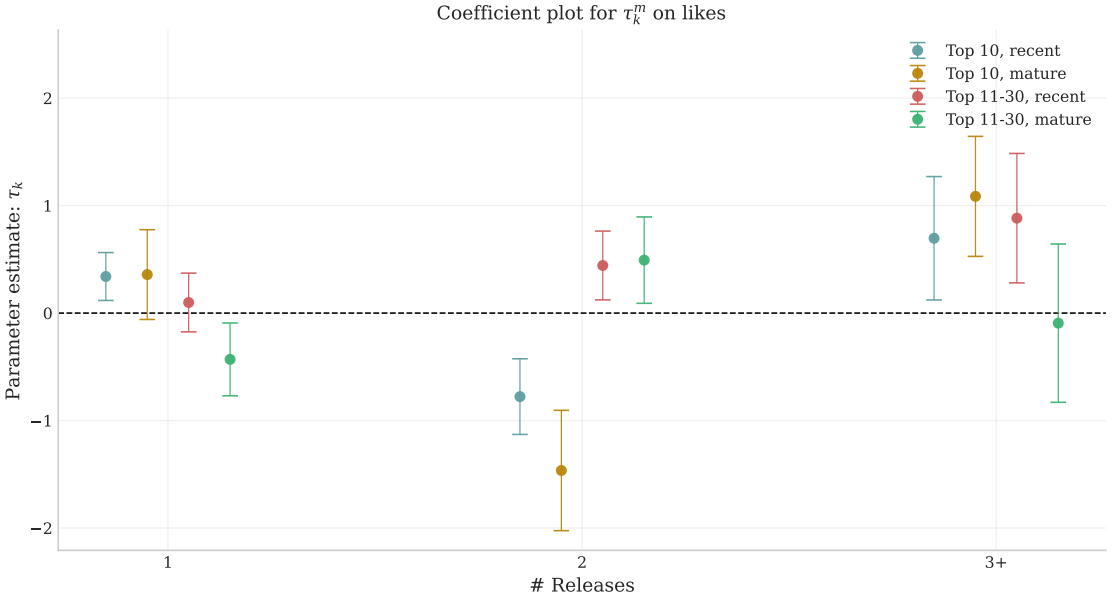


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects.

Figure C.13: Spillover event study: third-party descendant releases, top 1% of focal models

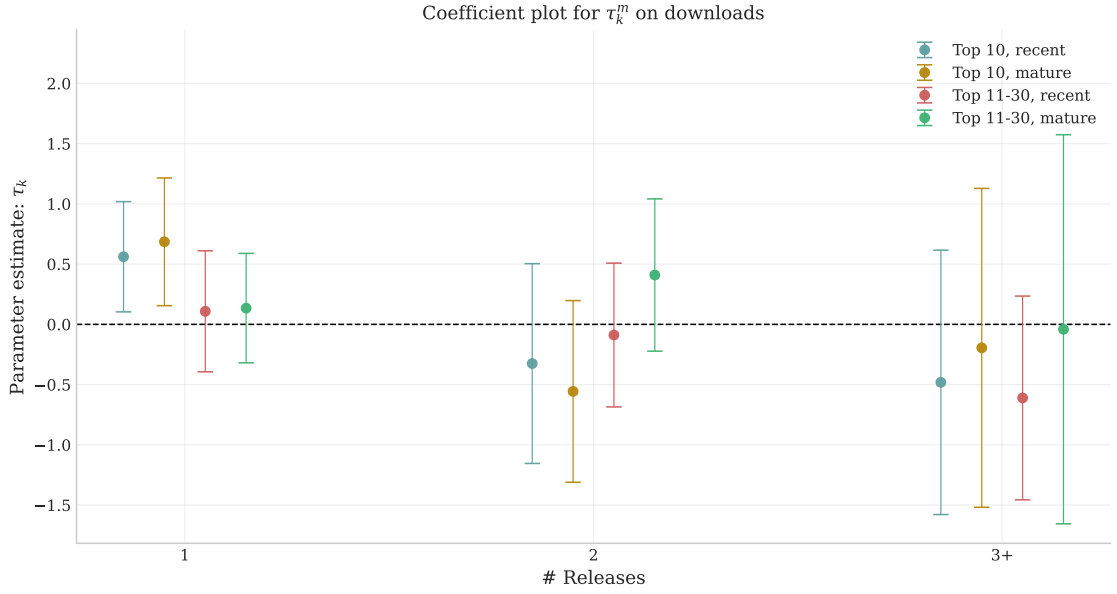


Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

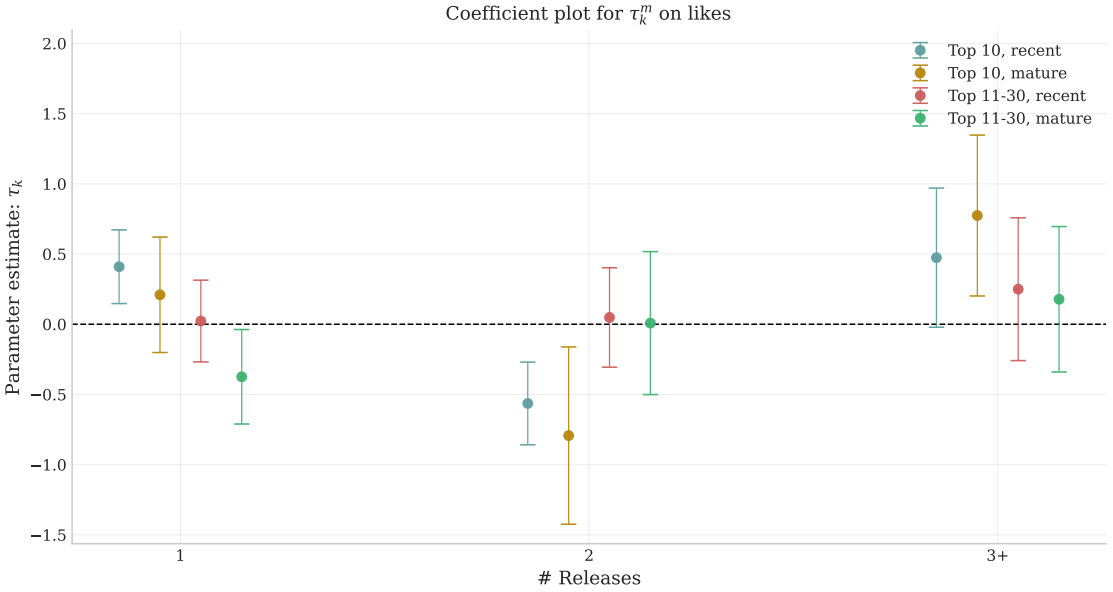


Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 1% of models by initial popularity.

Figure C.14: Spillover event study: third-party descendant releases, top 10% of focal models



Note: Plots parameter estimates from a Poisson regression of weekly downloads on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.



Note: Plots parameter estimates from a Poisson regression of weekly likes on dummies for cumulative number of top 10 and top 11-30 model releases, split into short-run and long-run (4+ weeks) effects. Includes model fixed effects, calendar week fixed effects, and model age fixed effects, restricting to top 10% of models by initial popularity.

D Death

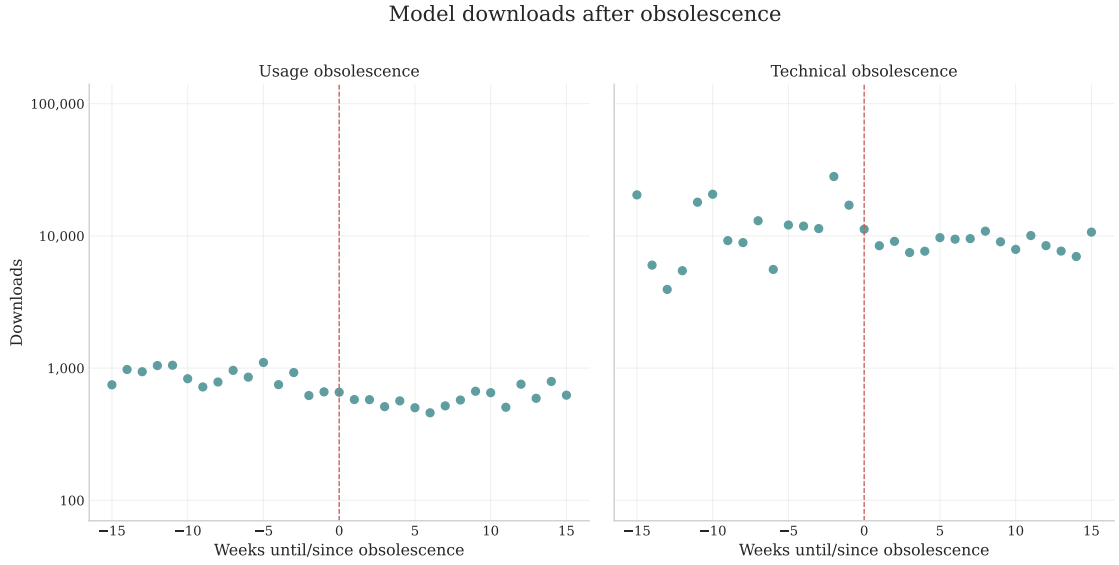
Table D.1: Usage obsolescence date differences

Model	Token share date	Dropout date	Difference (weeks)
TheDrummer/Anubis-70B-v1.1	2025-12-24	2025-12-31	1
allenai/molmo-7b-d-0924	2025-05-21	2025-11-05	24
google/gemini-flash-1.5-8b-exp	2025-04-30	2025-05-28	4
google/gemini-pro-vision	2025-05-14	2025-05-21	1
inclusionAI/Ling-1T	2025-11-26	2025-12-03	1
inclusionAI/Ring-1T	2025-11-26	2025-12-03	1
microsoft/phi-4-reasoning-plus-04-30	2026-01-28	2026-02-04	1
minimax/minimax-m1-40k	2025-06-18	2025-07-02	2
minimax/minimax-m1-80k	2025-06-18	2025-07-02	2
mistral/mistrall-small-2501	2025-02-05	2025-02-12	1
mistralai/Mixtral-8x7B-v0.1	2025-04-23	2025-04-30	1
moonshotai/Kimi-Linear-48B-A3B-Instruct	2025-12-24	2025-12-31	1
openai/shap-e	2024-11-06	2025-04-09	22
qwen/qwen2.5-7b-instruct	2025-05-21	2025-06-04	2
qwen/qwen3-1.7b-04-28	2025-06-04	2025-06-11	1
test/model	2026-01-14	2026-01-21	1
thudm/glm-4-32b-0414	2025-09-17	2025-10-01	2
thudm/glm-4-9b-0414	2025-05-28	2025-06-11	2
thudm/glm-z1-32b-0414	2025-08-27	2025-11-12	11
thudm/glm-z1-rumination-32b-0414	2025-07-16	2025-07-23	1

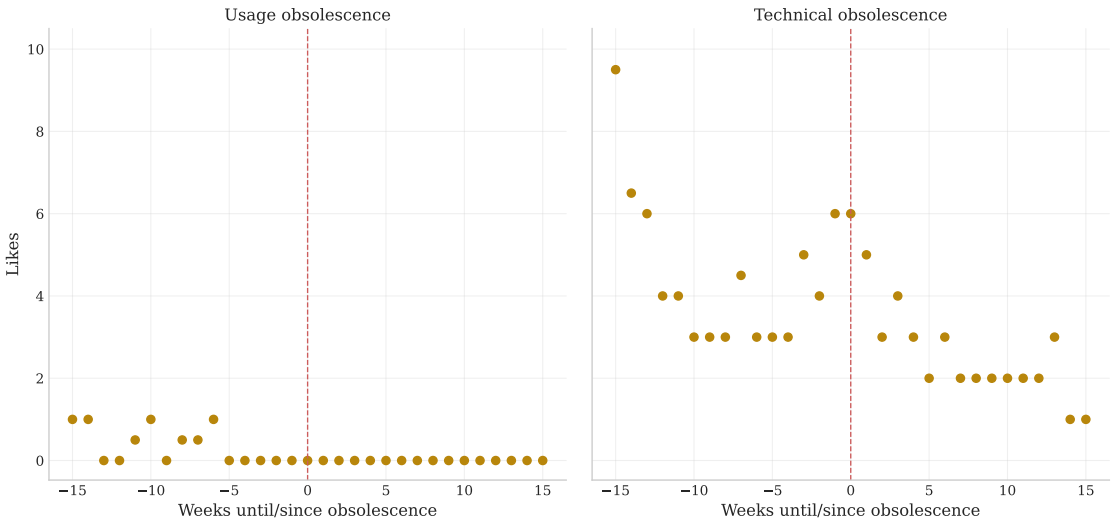
Table D.2: Best categories

Best category	Models
Artificial Analysis Benchmarks	
Intelligence index	162
Coding index	59
Math index	106
Agentic index	0
Total	327
Arena.ai	
Text style control	249
Vision style control	49
Search style control	19
Document style control	0
Webdev	15
Text to image	36
Image edit	23
Text to video	22
Image to video	19
Video edit	0
Total	432
Artificial Analysis Arena	
Text to image	92
Image editing	26
Text to video	54
Image to video	22
Total	194

Figure D.1: Demand and attention around obsolescence (medians)



Note: Plots median downloads by weeks until/since obsolescence.

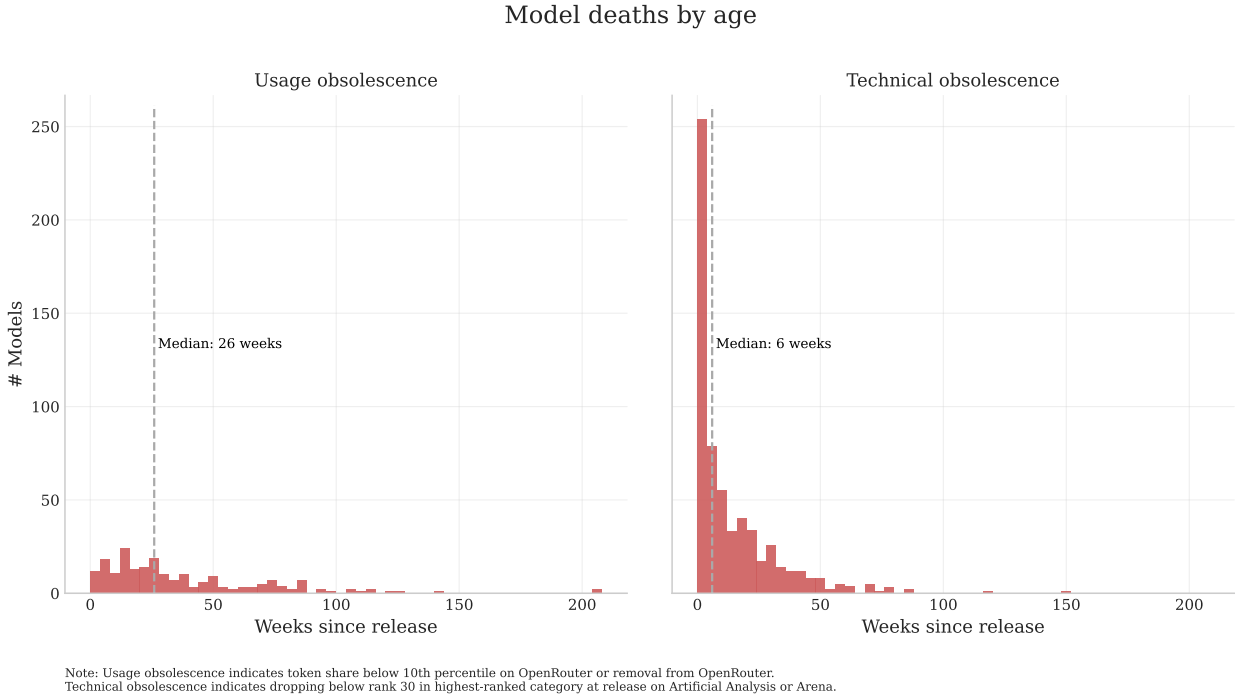


Note: Plots median likes by weeks until/since obsolescence.

D.1 Results for all models

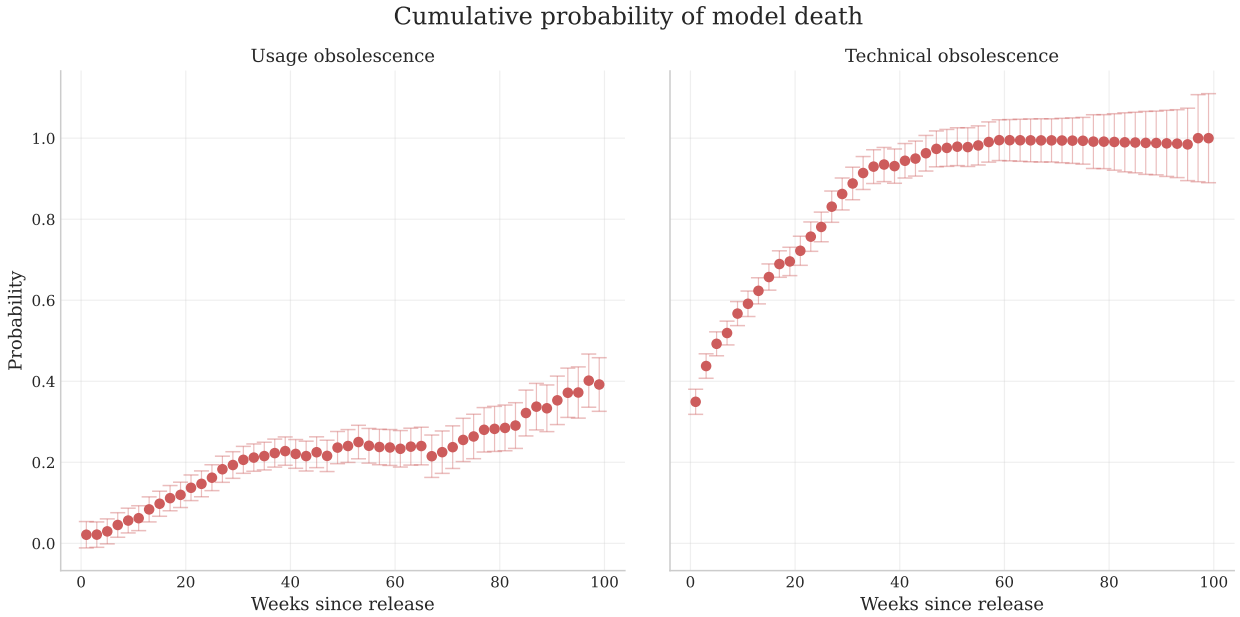
Here we present results from the analysis using all models on OpenRouter, Artificial Analysis, and Arena.ai, in contrast to the main body which shows results restricted to our primary sample of economically relevant models on Hugging Face. The starkly different patterns for age at obsolescence persist.

Figure D.2: Model age at obsolescence



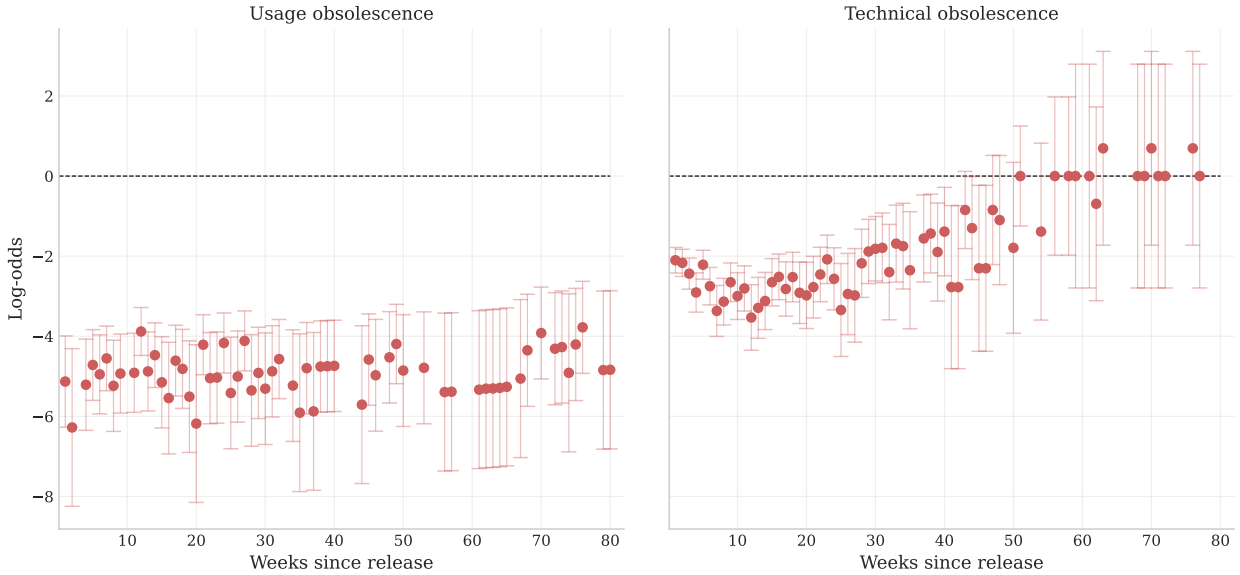
Note: Usage obsolescence indicates token share below 10th percentile on OpenRouter or removal from OpenRouter.
Technical obsolescence indicates dropping below rank 30 in highest-ranked category at release on Artificial Analysis or Arena.

Figure D.3: Share of models reaching usage obsolescence by weeks since release.



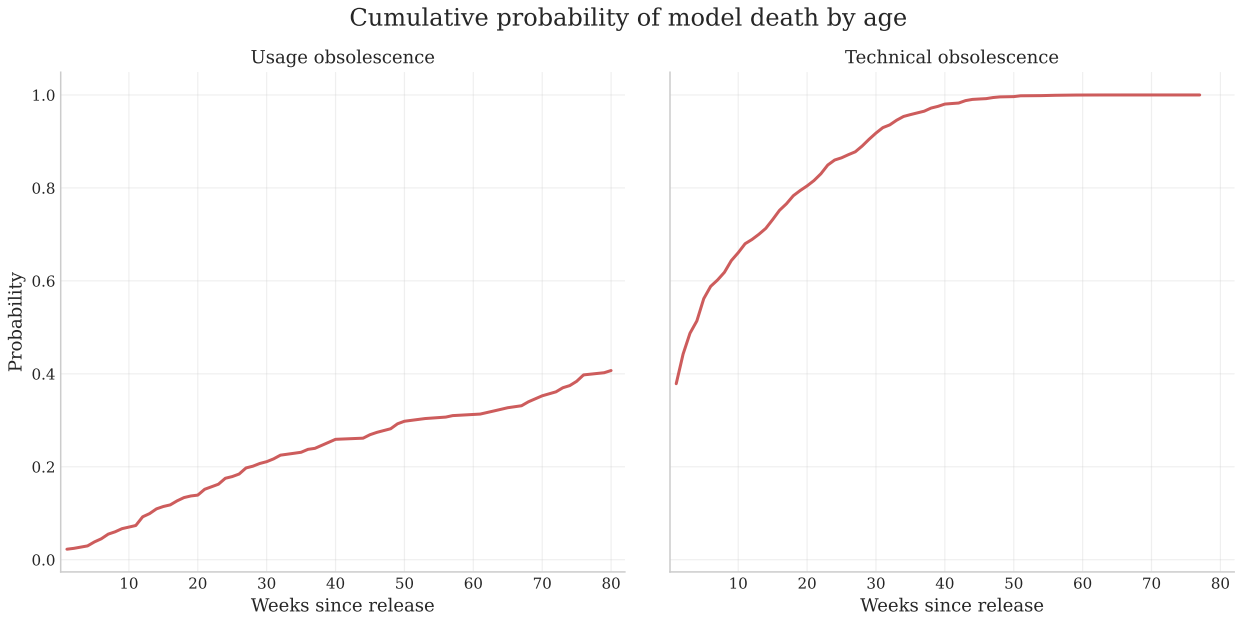
Note: Plots coefficients from the following ordinary least squares regression:
Outcome: Binary indicator for whether a model is obsolete in a given week.
Parameter of interest: Coefficient on week since release dummy (week 52 excluded).
Plot shows coefficients on every other week.
Includes models not matched to Hugging Face.

Figure D.4: Log-odds of obsolescence by age.



Note: Plots estimated log-odds of death from the following logit model:
Outcome: Binary indicator for a model dying in a given week.
Sample: Model-weeks up to and including the death week.
Parameter of interest: coefficient on week since release.
Log-odds of zero implies 50% probability of obsolescence.
Plot shows coefficients on every other week, excluding weeks with no variation in obsolescence.
Includes models not matched to Hugging Face.

Figure D.5: Cumulative probability of usage obsolescence by weeks since release.



Note: Cumulative probability implied by estimated conditional probabilities.
Includes models not matched to Hugging Face.