Birth, Life, and Death of AI Models

Aaron P. Kaye* **Kazimier Smith** **Neil Thompson**

Boston University & MIT MIT MIT

March 27, 2026

Preliminary and Incomplete. Please do not circulate or cite.

Abstract

We characterize the lifecycles: birth, life, and death of open-source and open-weight AI models using a novel dataset combining activity from Hugging Face and OpenRouter with performance data from Arena and Artificial Analysis. We document stylized facts regarding the supply and demand of AI models. First, regarding model "birth," we find that developers frequently use a strategy of batch releases, introducing models that are both vertically (e.g. size, capability, and latency) and horizontally (hardware and alignment) differentiated. Second, during the "life" of a model, we provide evidence of cumulative innovation through the creation of descendant models, including fine-tuned, quantized, and adjusted reasoning variants of base models. We also quantify model engagement through time and by model size. We then estimate spillover effects where new releases trigger demand for an author's previous works. Third, we analyze model "death." While interest in a model generally decays rapidly, we identify a persistent long-tail of downloads long after technological obsolescence. We find that even two years after release, 60% of models remain economically relevant. These patterns establish key institutional details of the open-weight AI market, providing an empirical foundation for research on AI competition, governance, and broader economic impact.

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

*Contact: apkaye@bu.edu.

1 Introduction

Artificial intelligence has become central to debates about productivity, competition, energy use, safety, and industrial policy, yet the market's basic institutional details remain poorly understood. In mature industries, researchers typically know what the product is, how firms version it, how new vintages are introduced, and when old vintages effectively exit. AI models do not fit neatly into those familiar templates. They are expensive to create but cheap to copy once released; they can be fine-tuned, quantized, distilled, and recombined into descendant models; and older vintages may remain in active use long after a technically superior successor arrives. Understanding the economics of this market requires institutional details that are not yet well established: how models enter the market, how they are used and built upon, and when they effectively exit. This paper aims to fill that gap. We document the lifecycle of open-weight AI models using novel data from Hugging Face, OpenRouter, and GitHub. We organize our analysis around 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).

This measurement-first approach follows a familiar tradition in economics. In markets where the relevant products, decisions, and competitive margins are not yet well understood, establishing key institution details can be critical in shaping research that furthers our understanding of the new market, and better informs policy. Griliches (1957) documented how hybrid corn diffused across U.S. states at sharply different rates, establishing the empirical patterns that shaped decades of work on technology adoption. Gort and Klepper (1982) traced entry and exit 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 IO 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, reframing a policy debate that had proceeded without a clear empirical anchor. In each case, getting the institutional picture right was a precondition for productive theory and policy. We aim to do something similar for AI. By pairing institutional measurement with regression evidence to characterize how models enter, compete, diffuse, and persist, we hope to provide an empirical foundation that disciplines subsequent modeling and reduces the risk of policy conclusions built on the wrong market picture.

We study these questions in the open-weight ecosystem, by which we mean models whose weights are publicly available even when their full training code or data are not. This setting offers an unusually visible window into a broader AI market that is otherwise difficult to observe. Our primary data come from Hugging Face, the largest platform for hosting and distributing open-weight models, supplemented with usage data from OpenRouter and additional information on downstream dependencies. Together these sources let us observe important upstream product-market choices and downstream adoption patterns: developers choose

what to release, how many versions to release together, how to differentiate those versions, and how long older models continue to be used after newer ones appear. They also let us distinguish what we think of as the hype cycle from the life cycle. Likes, mentions, and trending indicators capture attention around release; downloads and token usage more closely track adoption and continued economic use. The open-weight setting is especially useful because it makes visible a form of cumulative innovation that is harder to observe elsewhere: the same base model can seed fine-tunes, quantizations, reasoning variants, and other descendants that extend its technical and economic life.

These institutional facts matter because researchers and policymakers can easily make the wrong assumption at each stage of the lifecycle. They may assume that one training run produces one product, when in practice a single release often generates an entire family of vertically and horizontally differentiated models. They may assume that a new model cannibalizes and replaces older ones, when in practice new releases can spill demand onto earlier models from the same developer. And they may assume that technological obsolescence implies economic exit, when in practice old models can persist through downstream dependencies and embedded workflows. Those distinctions matter for first-order questions. The value of training data is mismeasured if one counts each model upload as an independent product rather than as part of a family generated from a common underlying asset. The value and governance of frontier compute are misjudged if economic activity is mediated largely through descendants, fine-tunes, and smaller variants rather than through a small set of frontier models in isolation. Antitrust analysis depends on whether models within a family are substitutes that should be aggregated into a single dominant product or differentiated complements that expand a developer's portfolio. And proposals to deprecate or sunset older models for safety reasons require understanding why obsolete models persist in use.

The lifecycle framing also clarifies how AI maps to, and departs from, more familiar markets. Model birth resembles product-line design. An iPhone launch provides a useful vignette: many consumers decide they want the next iPhone before choosing which version to buy, and the firm meets that demand with a portfolio differentiated by capability and price. Major AI releases often look similar, with families of models differentiated by parameter count, hardware compatibility, modality, task specialization, or alignment. More broadly, birth resembles multiproduct design under shared fixed costs and heterogeneous demand. Model life resembles cumulative innovation in software and platform ecosystems, where new products build on existing ones and can generate within-portfolio spillovers. Model death resembles legacy lock-in in enterprise software and durable systems, but with a distinctive twist: because old models remain downloadable and embeddable, technical supersession need not imply economic disappearance. In each phase, AI markets are familiar enough to connect to existing theory, but different enough that the measurement must be redone rather than assumed.

Guided by this birth-life-death framework, we document a set of stylized facts about how open-weight AI models enter, evolve, and exit the market. First, at birth, developers frequently employ a strategy of batch release, introducing families of models that are vertically differentiated by parameter count and capability and

horizontally differentiated by specialization, hardware compatibility, and alignment. This implies that the relevant unit of product design is often the model family rather than the individual model. Second, during life, we show that hype and use follow different dynamics: attention peaks quickly, but downloads persist much longer, and new releases can generate spillovers that increase demand for a developer's older models. We also document cumulative innovation through the rapid creation of descendant models. Third, at death, we show that while interest in a model decays rapidly, a persistent long tail of downloads remains well after benchmark obsolescence, driven in part by path dependence and downstream lock-in. Together, these patterns show that open-weight AI does not fit a simple one-model-per-training-run or vintage-replacement view of technological change. They also provide an empirical foundation for ongoing work on model demand, competition, and the economics of model development. The remainder of the paper proceeds as follows: Section 2 describes our main data sources and our method for identifying economically relevant models. Section 3 documents how models are often released in batches. Section 4 quantifies the model lifecycle and the effect of new model releases on existing models. Section 5 estimates how long models survive before they die. Section 6 concludes.

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 that examines open-weight ecosystems as settings of reuse, fine-tuning, and interdependent supply chains. Third, a 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 genuine 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.Kapoor et al. (2024) similarly argue that open foundation models have distinctive properties that create both innovation benefits and governance risks. We contribute to this literature by providing new measurement on how open-weight models actually enter and compete: major releases often arrive as portfolios of vertically and horizontally differentiated variants, implying that the relevant unit for competition analysis is frequently the model family rather than the single model.

A 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)

document sprawling Hugging Face family trees with rapid, directed mutation across model lineages. We connect to this literature by adding an economic lifecycle perspective: descendant creation, inherited lineage, and downstream adaptation are not just technical facts, but central determinants of how models create value after release.

A third strand begins to measure demand in model markets directly. 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 is common. We complement this literature by linking demand to lifecycle dynamics. We distinguish attention from use, show that salient releases can generate persistent backward spillovers to a developer's older models, and document that technical supersession need not imply economic exit. These facts help explain why AI markets do not resemble a simple sequence of vintage replacements, but instead operate as portfolios in which discovery, complementarity, and downstream lock-in matter.

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.

2 Data

Our primary data source is Hugging Face,[1] the largest online platform for hosting machine learning models. 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 offered a place to store the model weights along with an API to more easily use the 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 aims to make each model available to any developer that wants to use it, so it must make the core part of the model architecture publicly available. In practice this means offering the model "weights". 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*. The latter 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. In fact,

[1] huggingface.com

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"[2]. 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.

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 March 2026. See Section 2 for details.

Table 2.1 summarizes the key variables. 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.[3] 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 level.

[2]https://huggingface.co/datasets/cfahlgren1/hub-stats
[3]https://huggingface.co/models

2.1 Model downloads

The number of times each model on Hugging Face was downloaded is one of our key outcome measures. We consider downloads to be a proxy for how often and how intensively the model is used by developers. Since downloads mechanically record requests to the Hugging Face servers for specific model files, there are some caveats. For example, an automated pipeline in a machine learning tutorial might download a model every time it is run, or a package might download a model every time it is installed. In these scenarios, a download does not automatically equate to the developer actually using the model for local inference. However, Figure A.2 shows a positive correlation between downloads on Hugging Face and direct token usage on OpenRouter, a more direct measure of true usage.

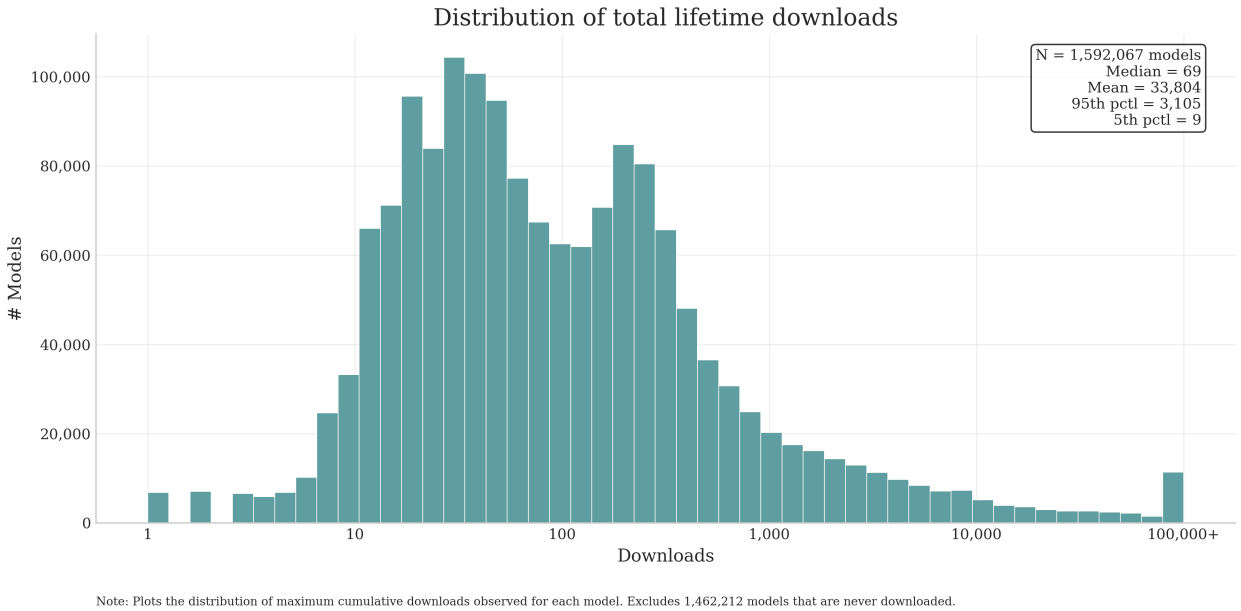
If we observed every model in the data starting on its creation date, we could use the rolling 30 day download count to calculate cumulative downloads. From that we could calculate the number of times the model was downloaded each week. However, our data is truncated at July 2024, so we do not observe 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.

$$D_{t-30} = D_t - R_t \tag{2.1}$$

Since we observe D_t and R_t at the daily level, we can use Equation 2.1 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 in week t. The algorithm in Section A.2 describes the procedure in detail. 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. After completing the backfilling process, we observe a monotonically increasing series of cumulative downloads for each model j and each week t in the data.

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 can upload a model to Hugging Face, and the model remains stored on the platform unless its creator removes it. This means many models on Hugging Face see little to no use during their lifetimes, and many models on the platform have long since been rendered obsolete. 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 immediately 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 still see very little activity. Figure 2.1 plots the



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

Figure 2.1: Distribution of total downloads

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 been downloaded more than 14 million times since it was uploaded to Hugging Face 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, 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. As before, this small number of downloads could obscure more interesting lifecycle patterns among the models that most developers actually use. We therefore further restrict the sample to those models that account for 99.5% of all downloads on Hugging Face. Figure A.1 shows that of the more than one million models that are ever downloaded, just 2,013 models account for nearly all downloads on the platform. For the remainder of the paper we restrict our analysis to these economically relevant models.

2.3 OpenRouter data

To supplement our data on open-weight models, we collect data from *OpenRouter*,[4] one of the largest intermediary platforms through which users can access AI models. 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.

[4] openrouter.ai

OpenRouter is useful for our purposes because it records direct usage in the form of tokens processed. 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. Table 2.2 documents the key variables.

Table 2.2: Key variables from OpenRouter

Variable	Description
ID	Unique identifier for the model on OpenRouter.
Hugging Face ID	Identifier for the corresponding model on Hugging Face. Used to merge the OpenRouter and Hugging Face datasets.
Tokens	Sum of prompt and completion tokens used and generated by the model. Observed daily, although there are some gaps in the data.

Note: Data retrieved from daily snapshots of OpenRouter, November 2023 to March 2026.

2.4 Artificial Analysis data

We obtain model quality data from *Artificial Analysis*,[5] an independent benchmarking platform that evaluates AI models on quality, speed, and price. Unlike Hugging Face and OpenRouter, which record usage, Artificial Analysis provides a standardized measure of model capability. The platform calculates an ELO score for each model based on its win rate in pairwise comparisons against other models across a battery of tasks. We treat this score as a time-invariant measure of model quality at the time of evaluation.

This quality measure is valuable because it allows 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. The ELO score gives us a way to control for quality when estimating the effects of other factors on model usage. Table 2.3 provides summary statistics for the models in our sample that appear on the Artificial Analysis leaderboard, and Table 2.4 documents the key variables.

Leaderboard	# Models	# Hugging Face models
Image editing	53	21
Text to image	119	35
Text to speech	65	8
Image to video	73	7
Text to video	79	10
Total (unique)	299	64

Table 2.3: Artificial Analysis leaderboard data

[5] artificialanalysis.ai

Table 2.4: Key variables from Artificial Analysis

Variable	Description
ID	Unique identifier for the model on Artificial Analysis.
Hugging Face ID	Identifier for the corresponding model on Hugging Face. Used to merge the Artificial Analysis and Hugging Face datasets.
ELO Score	Aggregated score based on the model's win rate in pairwise matchups against other models. Treated as a constant measure of model quality.

Note: Data retrieved from Artificial Analysis on March 24th, 2026.

2.5 Arena data

We also collect data from *Arena* (formerly LMArena and Chatbot Arena),[6] a platform that ranks AI models based on blind human preference judgments. On Arena, users submit prompts and receive responses from two anonymous models side by side. Users then vote for the response they prefer, and the platform aggregates these votes into an ELO rating for each model. Because the evaluations come from real users issuing diverse, self-selected prompts, the resulting scores reflect a broad notion of model quality as perceived by end users rather than performance on any fixed benchmark suite.

The Arena ELO scores complement the Artificial Analysis scores by capturing a different dimension of quality. Artificial Analysis evaluates models on structured tasks with well-defined criteria, while Arena reflects subjective human preferences in open-ended interaction. A model may score well on standardized benchmarks but perform less impressively in freeform conversation, or vice versa. Including both measures allows us to assess whether the lifecycle patterns we observe are robust to different notions of model quality. Table 2.5 provides summary statistics for the models in our sample that appear on the Arena leaderboard.

Leaderboard	# Models	# Hugging Face models
Code	56	7
Document	13	0
Image editing	39	4
Image to video	37	2
Search	22	0
Text to image	51	2
Text to video	37	3
Text	330	113
Video editing	4	0
Vision	103	23
Total (unique)	506	134

Table 2.5: Arena leaderboard data

[6] arena.ai

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 into the ecosystem. Compute costs and other significant fixed costs constrain the supply of AI models. At a given point in time training a 100 billion parameter model incurs compute costs orders of magnitude higher than a 100 million parameter model. Our data show a stark bifurcation in the size distribution of releases:

- **High-volume small models:** Small model releases (<10B parameters) approach 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 due to new capacity and entrants.

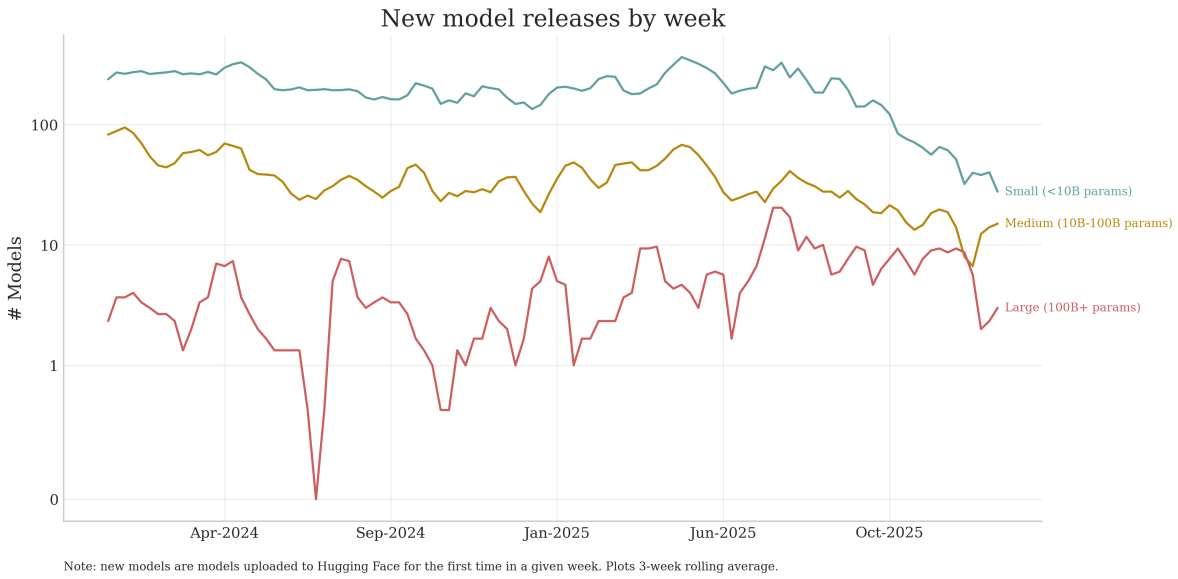


Figure 3.1: New model releases by week

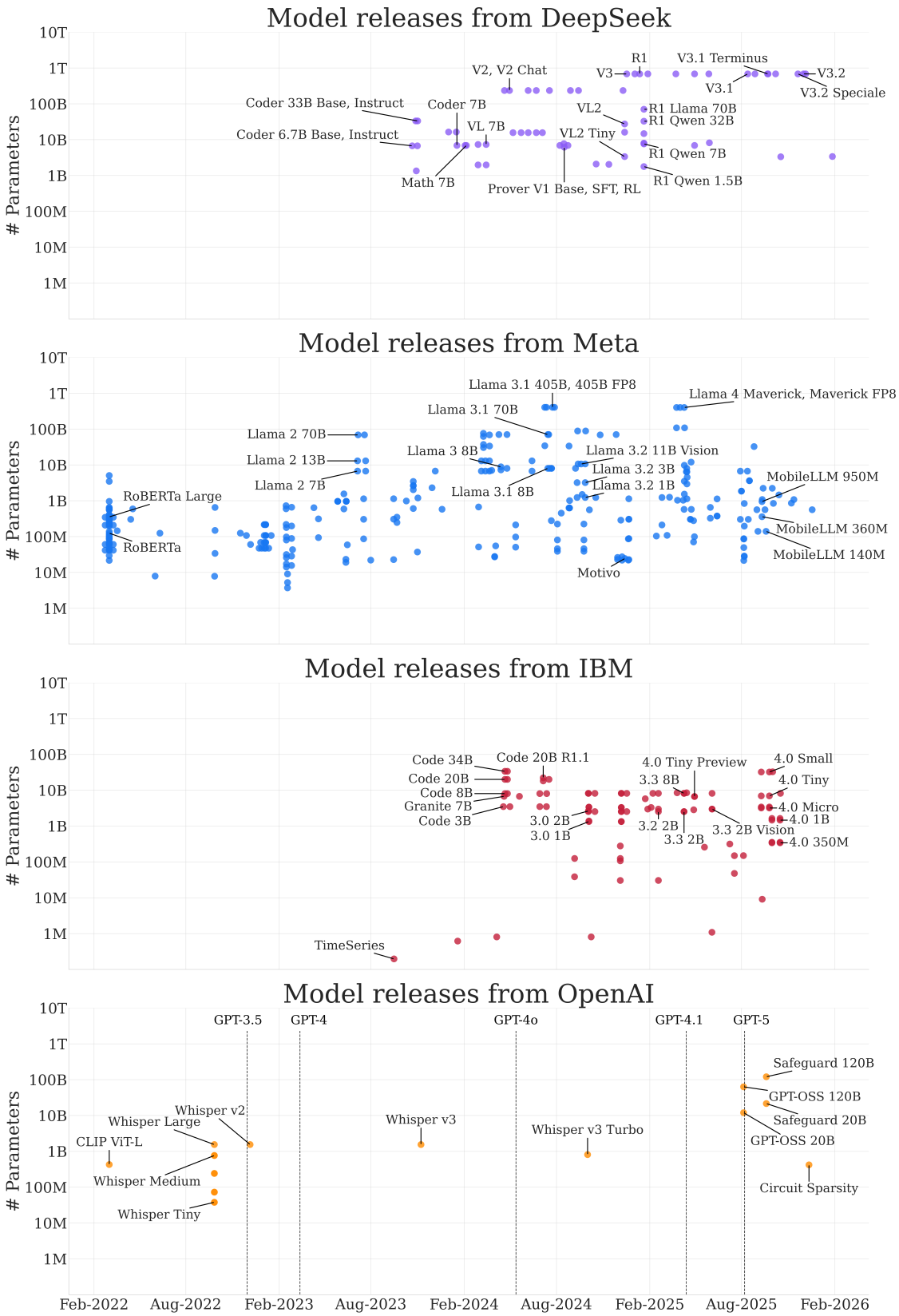
Despite the limitations on model development, the pace of new model releases remained relatively stable over the past year in all size classes. Large datacenters coming online might expand capacity to develop the largest models. Although the gap between small and medium model releases remained stable, the number of large model releases approached the number of medium model releases in the second half of 2025. This might reflect the changing composition of model developers, which we plan to explore in follow-on work.

3.2 Differentiation Strategies

Large authors rarely release a single model in isolation. We document a prevalence of *batch releases*, where authors typically upload multiple models simultaneously or in close succession. Examples of these batch releases include OpenAI's launch of gpt-oss-120b and gpt-oss-20b on the same day and IBM's release of base, speech-focused, and vision-focused version of its Granite 3.3 model within about one month. These batches exhibit two dimensions of differentiation:

1. **Vertical Differentiation:** Variation in capability and parameter counts. OpenAI's gpt-oss120b and gpt-oss-20b are vertically differentiated because the model with fewer parameters can tackle the same tasks as the larger version, but in general it will perform worse because it is has fewer parameters.

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, for example). The models should share the same basic vertical quality because they have the same parameter count.



Note: plots parameter count vs. release date for selected models.

Figure 3.2: Model releases over time

Figure 3.2 documents model releases on Hugging Face from four major developers: Meta, DeepSeek, IBM, and OpenAI. We plot the parameter count and creation date of each model each developer uploaded to Hugging Face. Note that although our panel data begins in July 2024, we observe earlier creation dates for many models in the sample. While it appears the Meta released many models around March 2022, some of these could be pre-existing models that Meta decided to upload at that time. Beyond this anomaly, Hugging Face creation dates appear to align with widely reported release dates (e.g. DeepSeek-R1 in January 2025). Text labels indicate major model releases and versions. Vertical differentiation appears from the very first models: BART Large and BART are higher and lower parameter count versions of the same model. These major developers appear to typically release three to four versions of major models vertically differentiated by parameter count to cater to varying needs and hardware limitations. Last, developers can use their base model to fine-tune smaller, more accessible models. DeepSeek, for example, released versions of Llama and Qwen fine-tuned on synthetic data from DeepSeek-R1.

Horizontal differentiation appears although it is less common. The most common type of horizontal differentiation apparent in this figure is base and instruct versions. The base version is the fundamental next-token-predictor version of the model meant for further fine-tuning and adaptation. The instruct version is fine-tuned with reinforcement learning or other methods to answer typical user prompts and to follow instructions. Since Hugging Face serves as a model repository where developers can build on existing models, model authors often upload both base and instruct versions. Other types of horizontal differentiation apparent in the figure include fine-tuning for specific tasks (e.g. DeepSeek Coder 7B and DeepSeek Math 7B) and different fine-tuning methods (e.g. DeepSeek Prover V1 SFT and DeepSeek Prover V1 RL, where the former uses Supervised Fine-Tuning and the latter uses Reinforcement Learning).

Do all model developers upload models in batches in this manner? 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 means it is very unlikely two models released within 30 days of each other are two distinct models pre-trained from the start. Instead, they are likely different iterations or versions of the same training run, like with the models fine-tuned using different methods described above. This idea motivates our batch-assignment algorithm. While it is possible that the procedure could assign two models that should be

in the same batch to different batches, Figure 3.2 indicates that batches typically release all their models in very quick succession. A 30-day window should therefore assign most batches correctly.

Figure 3.3 plots the distribution of batch size (in terms of the number of models in each 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 large batches. The pattern in the distribution of downloads across batches could be partly mechanical if each model in a batch gets some minimum number of downloads, in which case releasing larger batches would result in more downloads. On the other hand, if the number of users on the platform does not change in response to a batch release, then each model in the batch competes for the same set of users; in this scenario models in the same batch might simply cannibalize each others' use rather than increase total usage.

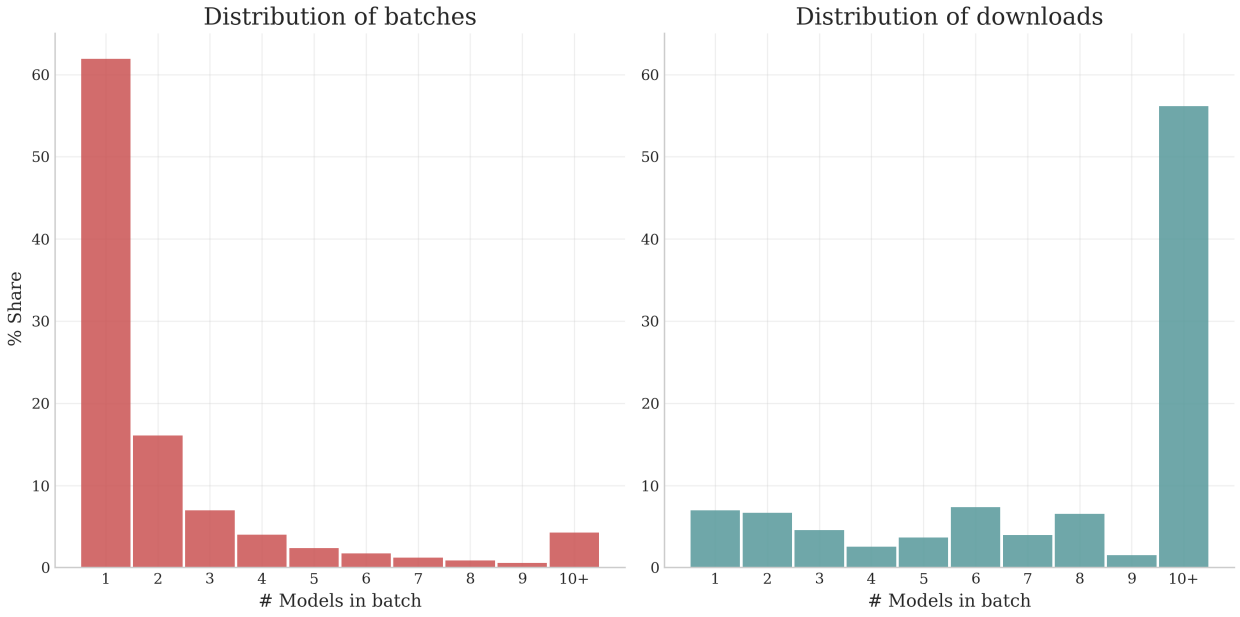


Figure 3.3: New model releases by batch size

However batch releases actually affect usage, the stark contrast between the two distributions in Figure 3.3 illustrates a recurring theme of severe inequality in model use. Even in our sample of economically relevant models, the typical model is released alone and likely receives relatively few downloads. A small number of models are released as part of major batches (like the many variants of DeepSeek-R1 released simultaneously). It appears that many users flock to these major batches. 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. In the next section, we analyze in

more detail what model releases look like and how each model from a developer affects the developer's other models.

4 Life

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

4.1 Model birth and 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. 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 utility. 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:

$$y_{jt} = \xi_j + \delta_t + \alpha_{t-t_0^j} + \alpha_{53}\mathbb{1}(t - t_0^j > 52) + \zeta_{dow}\mathbb{1}(t = t_0^j) + \varepsilon_{jt} \tag{4.1}$$

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 $t - t_0^j \in \{0, 1, \ldots, 52\}$. 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 ($\alpha_{52} = 0$).

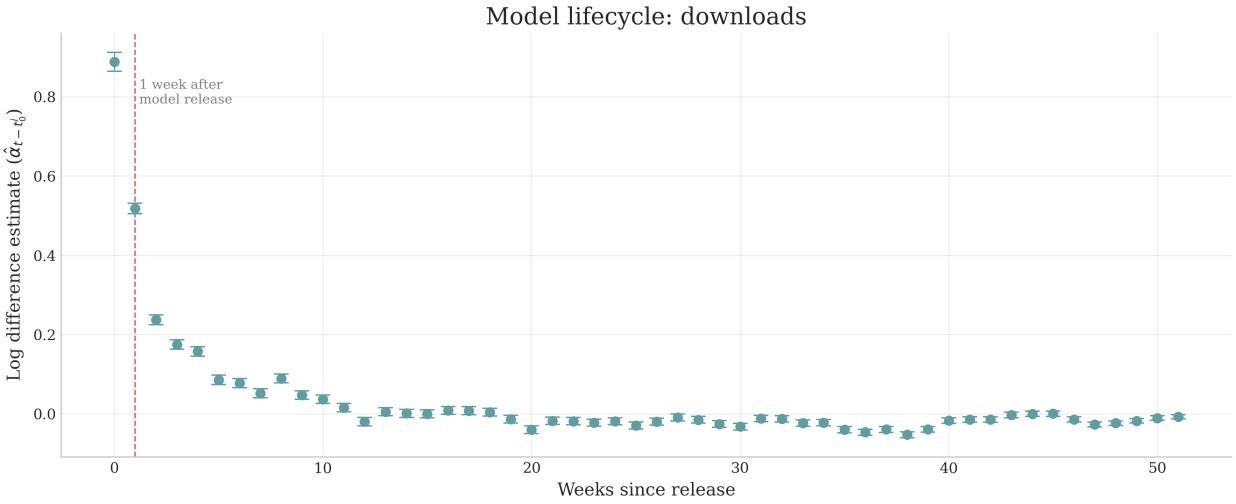
Because of the length of our sample, estimates of the model lifecycle beyond one year of release become increasingly noisy. We therefore focus our analysis on the first year of a model's life; this explains our choice to pool the model age variables beyond week 52. Appendix Section B.2 presents alternative specifications.

Figure 4.1 plots download and like counts on Hugging Face for the first year of the model lifecycle. Appendix Figure A.2 confirms that the pattern holds in the raw data, without any statistical analysis. Models receive more likes and downloads in the first week of their existence than in any subsequent week. Downloads

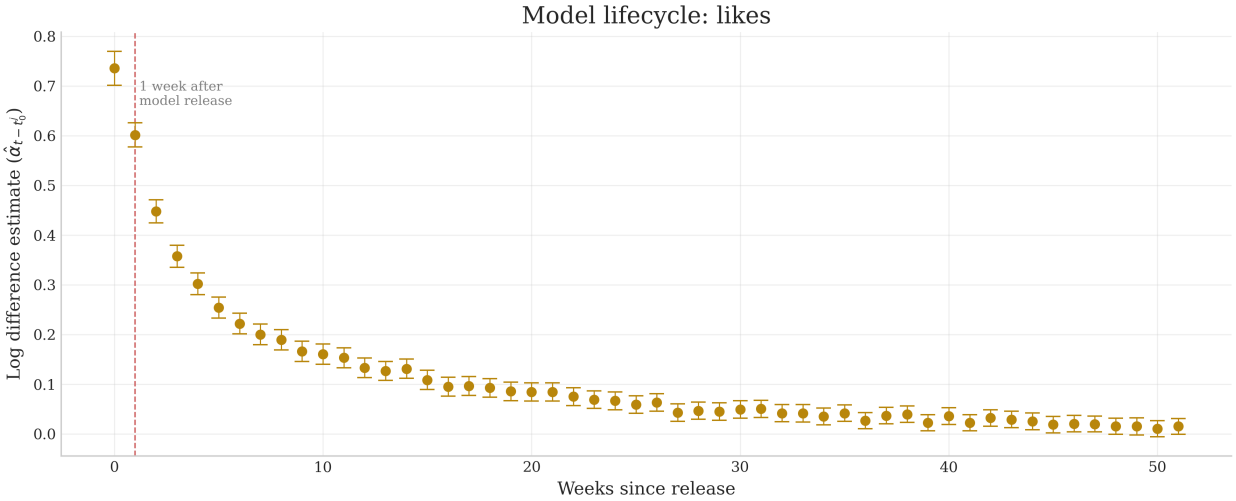
remain above zero for the duration of the model's life. On average, models receive almost as many downloads one year after release as they do one month after release. Instead of monotonically decreasing with age, download activity increases and decreases over the lifecycle. Even after the developer supersedes the model with a new, superior version, users still regularly download the old version. 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 a couple of possibilities include the following:

1. Llama 3 is part of an automated software pipeline and continues to see downloads whenever the pipeline is run.

2. Llama 3 has idiosyncratic features we do not observe that make it preferable to Llama 3.1 for some users, like the tone and personality of its responses. These factors might push users to stick with Llama 3 even if it performs worse than Llama 3.1 on all quantifiable outcomes.



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).

Figure 4.1: Utility and attention at model release

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* within model developers: when a developer releases a new model, does it cannibalize or complement the developer's existing models?

The answer is not obvious, and different markets point in different directions. 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 AI models worked this way, a developer's older models would see declining use after a new release. But 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).[7] 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.

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.

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

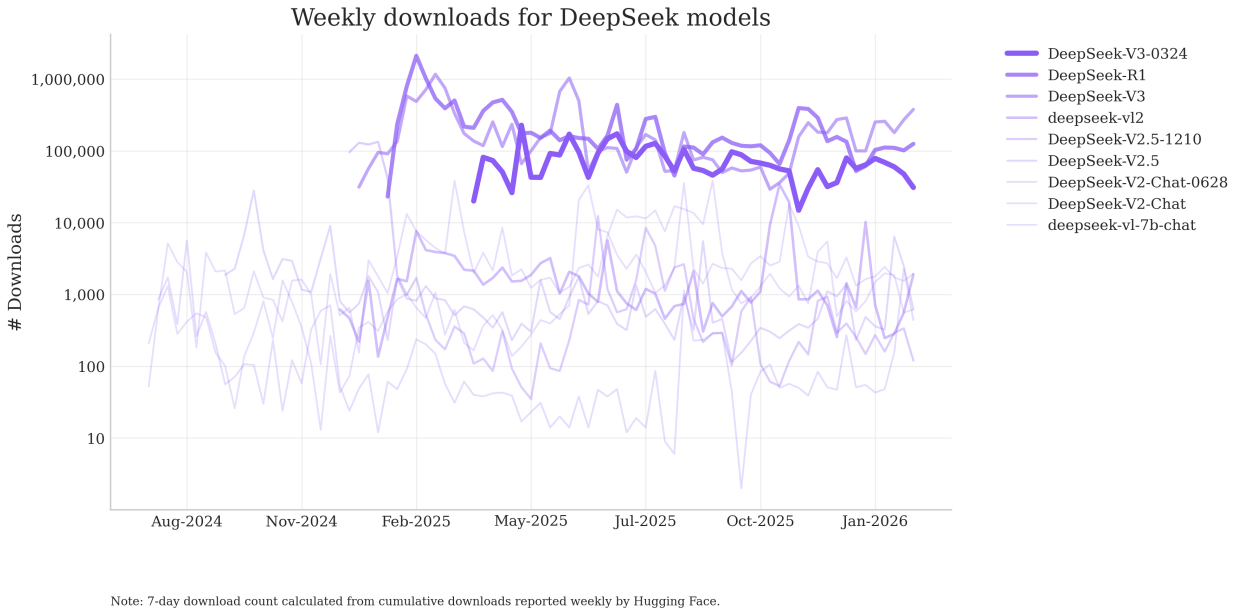


Figure 4.2: Downloads for DeepSeek models

To test for spillovers systematically, we estimate an event study 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 10 or 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 substantial noise and makes it difficult to identify meaningful effects.[8] 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 with $n = 10$ and $n = 30$:

$$y_{jt} = \xi_j + \delta_t + \alpha_{t-t_0^j} + \sum_{j' \in \mathcal{J}_j} \sum_{k=-15}^{15} \tau_k \mathbb{1}\left(s_{j',t_0^{j'}} \geq S^n_{t_0^{j'}}\right) \mathbb{1}\left(t - t_0^{j'} = k\right) + \varepsilon_{jt} \tag{4.2}$$

where y_{jt}, the outcome, is the inverse hyperbolic sine of the number of downloads model j received in week t. Our parameters of interest, τ_k, are the effect on the focal model's outcome k weeks before or after the release of a subsequent trending model from the same developer. ξ_j is a model fixed effect. δ_t is a time (week) fixed effect. t_0^j indexes the calendar week in which model j was released. $\alpha_{t-t_0^j}$ is a model age fixed effect.

[8] We present that analysis in Appendix Section B.3

$\mathcal{J}_j$ is the set of models from the author of model j released at least 30 days after model j. s_{jt} is model j's trending score in week t. S_t^n is the nth highest trending score among all models in week t. We use the inverse hyperbolic sine, which is well suited for situations where the outcome can have occasional zeros (Norton, 2022). In our setting, even popular models have weeks with zero downloads, and the distribution of downloads across models of a given age is highly skewed. 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. We restrict the subsequent release to models at least one batch ahead of the focal model, based on our definition of batches in Section 3.2. Without this restriction, a model released one day after the focal model would count as a treatment, 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.

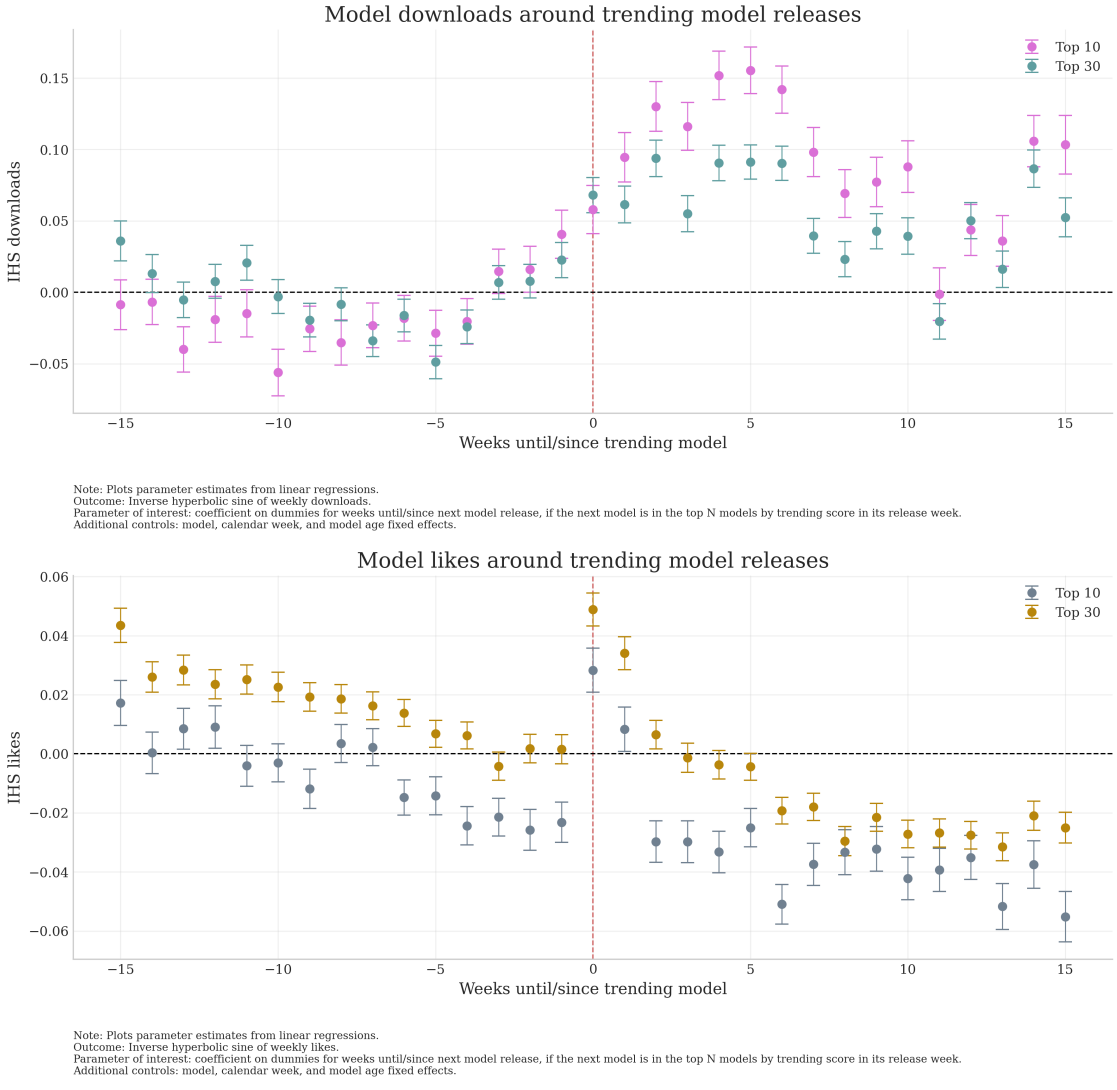


Figure 4.3: Utility and attention at subsequent model releases: trending models

Figure 4.3 plots the estimated coefficients $\hat{\tau}_{-15}, \ldots, \hat{\tau}_{15}$. The results reveal clear spillover pattern: popular new releases increase downloads of the developer's older models, and the effect is persistent. Similarly, we see a short lived increase in likes.

Downloads. The effect of a trending model release on the focal model's downloads is positive and statistically significant, beginning in the week of release and persisting for at least 15 weeks. The pre-period coefficients are close to zero and show no systematic trend, supporting the identifying assumption that the timing of trending releases does not generate anticipatory changes in downloads of older models. There is a possible small uptick in the weeks immediately before release, which could reflect pre-release announcements or leaks generating early interest in the developer's catalog. The magnitude of the post-release effect is economically meaningful: top 10 trending models generate effects roughly 50% larger than top 30 trending models, consistent with more prominent releases driving stronger discovery effects. The persistence is notable. Unlike a simple attention spike that dissipates within a few weeks, the increase in downloads lasts for months. This 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 long after the initial attention around R1 subsides.

Likes. The response of likes tells a different and instructive story. In the first two weeks after a popular release, likes on older models increase, consistent with users exploring the developer's catalog. But beyond the first two weeks, the effect turns negative: popular models appear to draw attention *away* from older models. Top 10 trending models have a larger negative effect than top 30 trending models, reinforcing the interpretation. Users have limited attention. A sufficiently prominent new release can monopolize the attention a developer's portfolio receives even as it increases actual use of older models. We interpret the likes result with more caution than the download result. The pre-period coefficients for likes show some instability, particularly for top 10 trending models, making it harder to attribute the post-release patterns entirely to the new release. Nonetheless, the sharp spike at $k = 0$ is distinct from any pre-period variation, and the overall divergence between downloads and likes 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 very different story than metrics that proxy for use.

Interpretation. Several mechanisms could drive the positive spillover in downloads. First, a prominent release may function as a brand event, raising awareness of a developer's full catalog among users who were previously unfamiliar with it. 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), 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.

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. Appendix Section B.4 presents additional specifications using model batches rather than individual models as the treatment; the qualitative patterns are similar.

We now turn to the final phase of the lifecycle: when and how models exit the market.

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 OpenRouter

Because OpenRouter is one of the largest platforms for accessing AI models, it offers a useful window into usage obsolescence. OpenRouter reports total tokens used by each model on a daily basis and actively 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,

$$s_{jt} = \frac{\text{tokens}_{jt}}{\sum_k \text{tokens}_{kt}},$$

is in the bottom 10% of the distribution of $\{s_{jt}\}_j$ in that week and in every week thereafter.

Death definition	# Models	# Deaths
Dropout	370	17
Market share	370	216
Total (unique)	370	223

Table 5.1: OpenRouter usage obsolescence summary

Table 5.1 summarizes how models become usage-obsolete on OpenRouter.

5.2 Artificial Analysis

To extend our ability to measure the end of a model's useful life, we add data from another major platform. Artificial Analysis[9] provides benchmark scores and other performance metrics for more than 400 proprietary and open-weight models. These data give us two complementary measures of technical obsolescence: pairwise user preferences from the Artificial Analysis *Arena* and benchmark-based index scores.

The first measure comes from the Artificial Analysis *Arena*, which allows users to make pairwise judgments between the outputs of two models in response to the same prompt. Figure A.3 depicts a typical interaction: 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 Artificial Analysis aggregates the resulting votes into an "ELO score" for each model. The ELO score captures how often a model wins pairwise matchups against its competitors. The platform ranks models using these scores across several categories, including image generation, video generation, and text-to-speech.

We define technical obsolescence on the Artificial Analysis Arena as follows. For each model, we identify the category in which it ranked highest in its release week. Once the model falls out of the top 30 in that category, we classify it as technically obsolete.

Best category	# Models	# Deaths
Text To Image	24	1
Editing	21	0
Text To Speech	8	0
Text To Video	8	0
Image To Video	3	0
Total	64	1

Table 5.2: Artificial Analysis Arena technical obsolescence summary

Table 5.2 summarizes technical obsolescence on the Artificial Analysis Arena.

[9] artificialanalysis.ai

Artificial Analysis also maintains a broader catalog of model performance across many dimensions. The platform reports benchmark scores for coding, math, reasoning, tool use, and related tasks, along with price and latency, and uses these inputs to construct an aggregate *Intelligence Index* for each model. It also reports scores for more specific tasks, including math, coding, and agentic use. We collect these index scores as an additional measure of technical quality. The arena data capture user preferences directly, while the index scores summarize performance on standardized evaluations.

To define technical obsolescence using index scores, we follow a procedure parallel to the one above. For each model, we identify the index on which it ranked highest in its release week. Once the model falls out of the top 30 on that index, we classify it as technically obsolete. Table 5.3 reports the number of models in each category and the number that eventually reach technical obsolescence. The fact that most models do so is consistent with the pace of technological progress: a model that ranked near the frontier on a task two years ago is likely to have been surpassed by many newer models by the present day.

Best category	# Models	# Deaths
Agentic	60	46
Intelligence	58	46
Math	38	20
Coding	21	9
Total	177	121

Table 5.3: Artificial Analysis index-based technical obsolescence summary

5.3 Arena

We further supplement these data with Arena, one of the most widely used platforms for evaluating large language models through direct human judgment.[10] On Arena, users submit prompts and receive responses from two anonymous models. After comparing the outputs, the user selects the response they prefer or declares a tie. These pairwise comparisons are aggregated into ELO ratings that reflect each model's relative quality as perceived by a broad user base. The platform maintains leaderboards across categories including overall performance, coding, mathematics, and instruction following.

Arena is a valuable complement to both OpenRouter and Artificial Analysis. OpenRouter captures commercial usage patterns, and Artificial Analysis combines benchmark scores with its own arena data. Arena adds a large-scale, crowd-sourced measure of technical obsolescence grounded in user preferences. Because researchers, developers, and practitioners routinely consult the Arena leaderboard when selecting models, continued presence on the platform is a meaningful signal of ongoing technical relevance.

We define technical obsolescence on Arena using the same logic as on the Artificial Analysis Arena. We identify the category in which the model ranked highest when it first entered the leaderboard. Once the model falls out of the top 30 in that category, we classify it as technically obsolete. Table 5.4 reports the number of models and the number reaching technical obsolescence by category. As with the Artificial Analysis measures, most models eventually become technically obsolete on Arena, reflecting the rapid pace of improvement in model capabilities.

Best category	# Models	# Deaths
Text	89	66
Vision	16	0
Image Edit	3	0
Image To Video	2	0
Code	2	0
Text To Image	1	0
Text To Video	1	0
Total	114	66

Table 5.4: Arena technical obsolescence summary

[10] arena.ai

5.4 Combined obsolescence definition

Each of the data sources above captures a different dimension of model obsolescence. OpenRouter measures usage obsolescence: models that lose traffic or are removed from the platform no longer attract meaningful commercial usage. 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 a model can remain active on one platform while already obsolete along another margin, combining these definitions gives a more complete picture of the model lifecycle.

Table 5.5 summarizes the combined obsolescence classifications. By taking the union of usage-obsolescence and technical-obsolescence definitions, we capture models that lose relevance through declining use, weaker benchmark performance relative to newer models, or worsening user preference. This combined measure forms the basis for the survival analysis in the sections that follow.

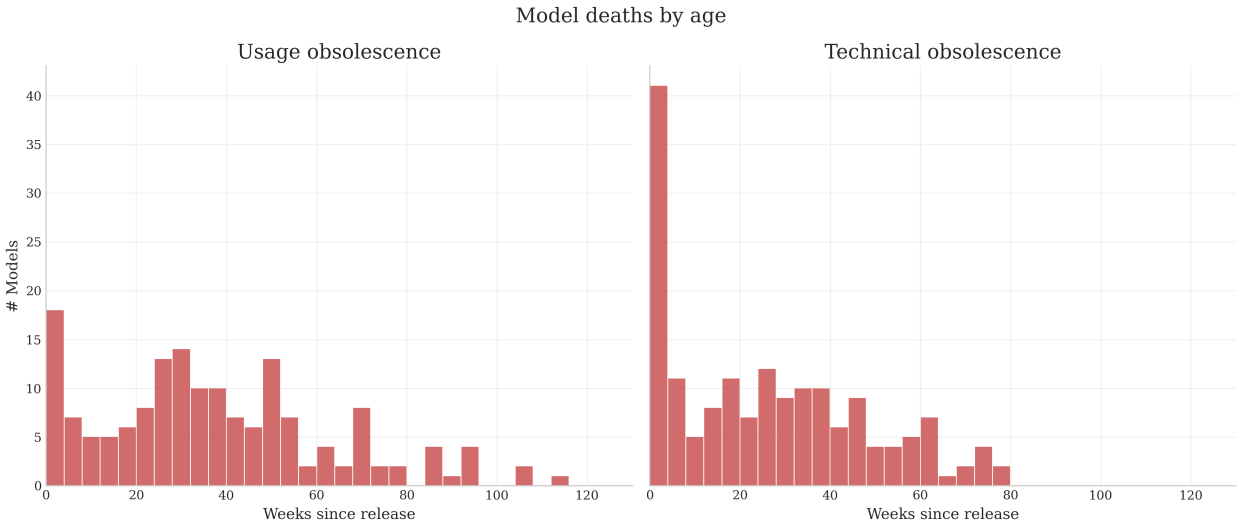
Reason for death	# Deaths	Median age (weeks)
OpenRouter dropout	7	26.1
OpenRouter market share	144	34.6
Artificial Analysis rank	1	35.1
Artificial Analysis intelligence	98	9.7
Arena rank	47	45.9
Total	297	29.0

Table 5.5: Model obsolescence summary (all definitions)

5.5 Age at obsolescence

Figure 5.1 plots the distribution of model age at obsolescence. The median model reaches usage obsolescence 39 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.

Figure 5.1: Model age at usage obsolescence



The distribution of age at usage obsolescence suggests that many models become economically irrelevant before they are one year old, but we would like 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 usage obsolescence by a given age. Specifically, we implement the following specification:

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

where $\texttt{obsolete}_{jt} \in \{0, 1\}$ is an indicator equal to 1 if model j has reached OpenRouter usage 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 $\hat{\beta}_0, \ldots, \hat{\beta}_{80}$; we do not observe enough technical-obsolescence events to estimate probabilities beyond 80 weeks of age. The probability of obsolescence increases with age. If we observed all models over their full lifecycle, these probabilities would increase monotonically. In practice, however, some models were only recently uploaded to OpenRouter, 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 OLS is not ideal for estimating probabilities bounded between 0 and 1, we also estimate a hazard model. The hazard model answers questions of the

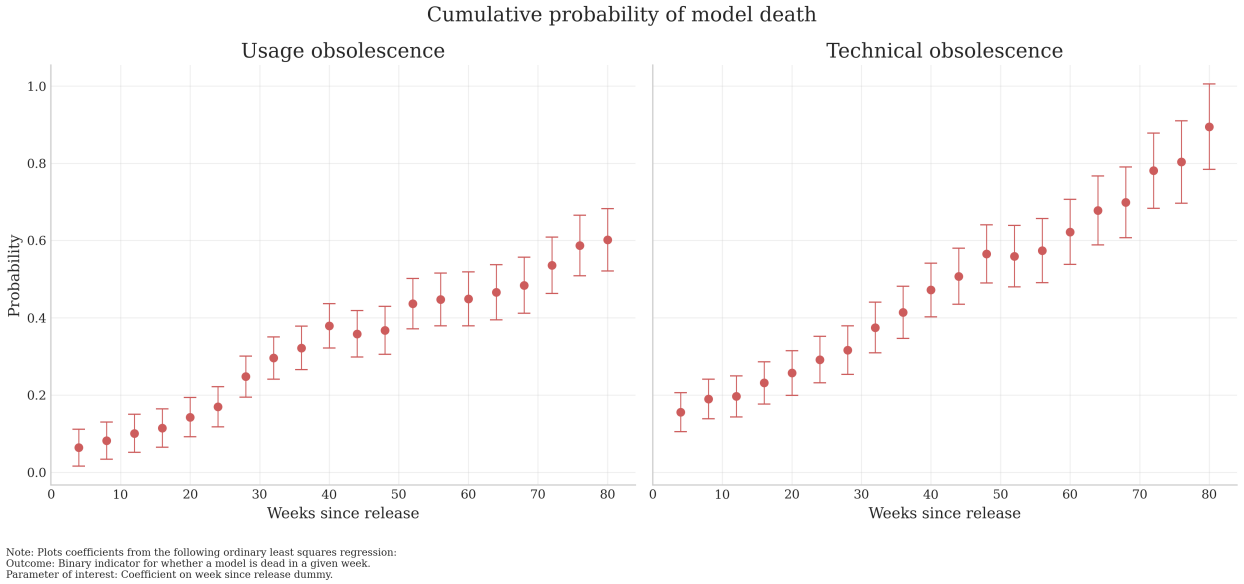


Figure 5.2: Share of models reaching usage obsolescence by weeks since release. Usage obsolescence defined as token share below the 10th percentile.

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:

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

where $o_{jt} \in \{0, 1\}$ is an indicator equal to 1 if model j reaches OpenRouter usage 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 for each $t - t_0^j \in \{0, 1, \ldots, 80\}$, with α_{52} as the excluded baseline. The estimation sample is restricted to model-weeks in which model j has not yet reached usage 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 $\hat{\alpha}_0, \ldots, \hat{\alpha}_{80}$. Again, we do not observe usage obsolescence beyond 80 weeks since model release, so we cannot estimate additional coefficients. We also do not observe obsolescence at every age from 0 to 80. 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 likely to reach usage obsolescence in the following week. As a final check, we compute the implied cumulative probability of usage obsolescence at each age using the hazard-model estimates. The implied survival curve as a function of model age is

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

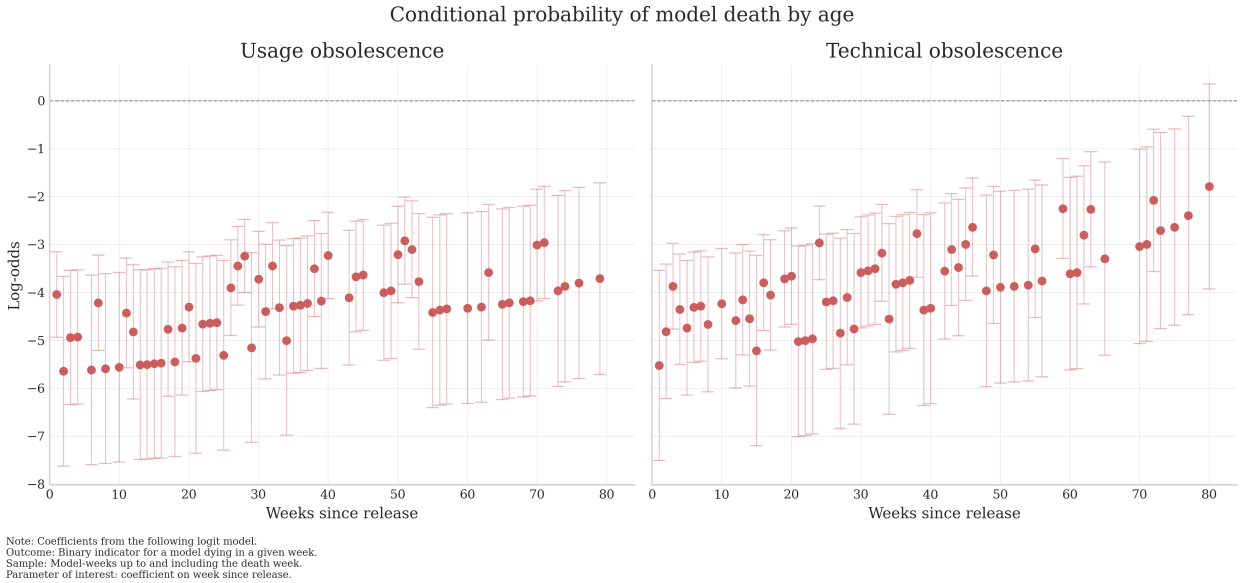


Figure 5.3: Log-odds of usage obsolescence by age. Usage obsolescence defined as token share below the 10th percentile.

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: about 50% of models reach usage obsolescence within one year of release, and by two years that share rises to about 80%. This is striking. Frontier labs release several new models per year, yet models often remain in use for a long time. More than 20% of models in our matched sample have still not reached usage obsolescence two years after release, long after higher-performing alternatives exist. Understanding why users continue to rely on older models remains an interesting direction for future research.

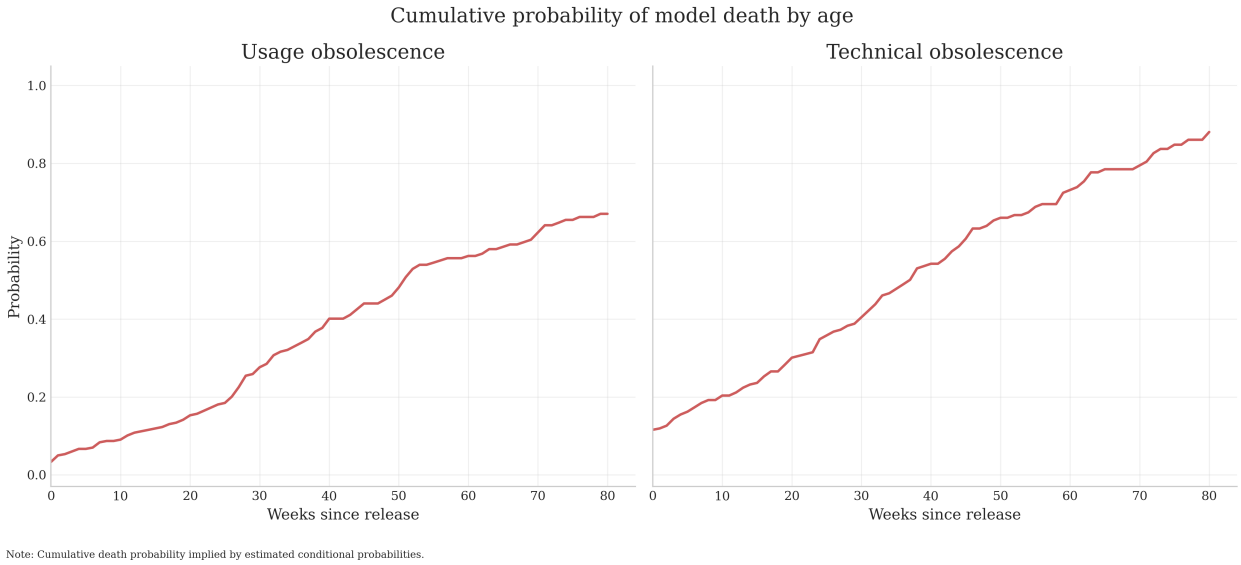


Figure 5.4: Cumulative probability of usage obsolescence by weeks since release. Usage obsolescence defined as token share below the 10th percentile.

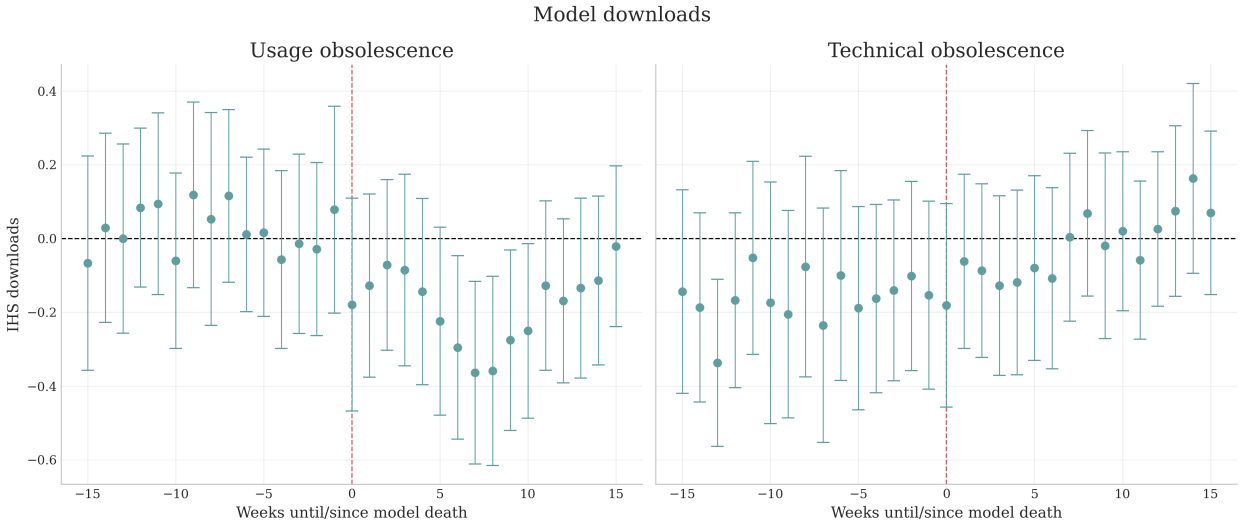
5.6 Effects of usage obsolescence

Because our obsolescence measure is defined outside Hugging Face, we can use it 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 utility on Hugging Face after a model reaches usage obsolescence on OpenRouter. To study Hugging Face activity around that event, we estimate an event study centered on the model's usage-obsolescence date on OpenRouter. We plot the estimated values of τ_k from the following specification:

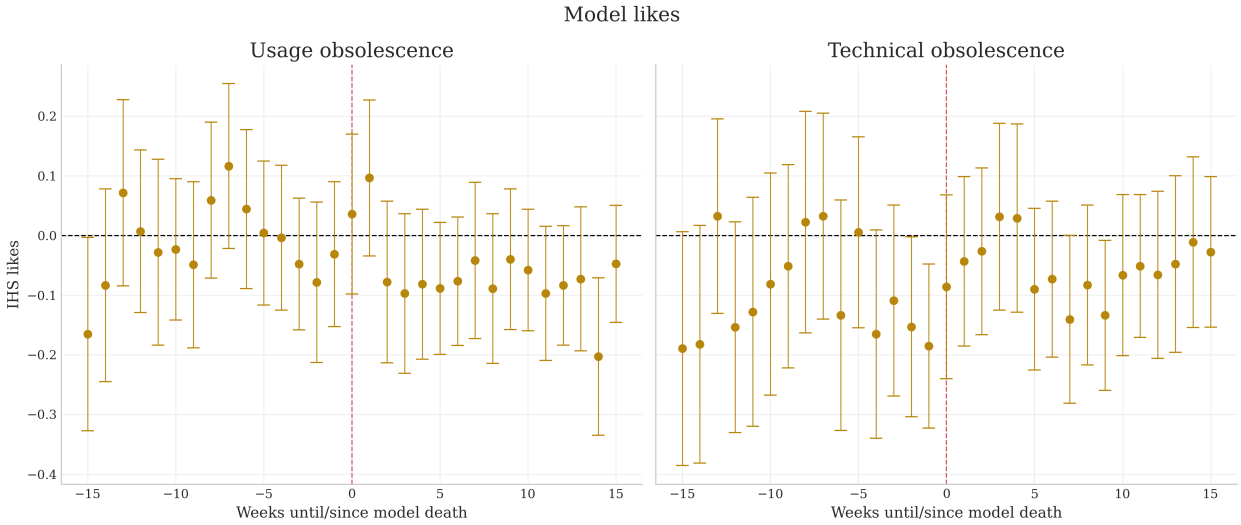
$$y_{jt} = \xi_j + \delta_t + \alpha_{t-t_0^j} + \sum_{k=-15}^{15} \tau_k \mathbb{1}\left(t - t_u^j = k\right) + \varepsilon_{jt} \tag{5.5}$$

where y_{jt} is the inverse hyperbolic sine of the number of downloads (or likes) model j received in week t. t_0^j is the calendar week of model j's release. t_u^j indexes the calendar week in which model j reaches usage obsolescence on OpenRouter. Our parameters of interest, τ_k, capture the effect on the focal model's Hugging Face outcome k weeks before or after it reaches usage obsolescence on OpenRouter. 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.

As in our event studies around subsequent model releases, we use the inverse hyperbolic sine because downloads and likes include many zeroes and remain highly skewed even among economically relevant models. We include our usual controls for model, calendar week, and model age to isolate how Hugging Face activity changes around usage obsolescence rather than around secular trends or the ordinary lifecycle of model usage.



Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly downloads.
Parameter of interest: coefficient on dummies for weeks until/since model death.
Additional controls: model, calendar week, and model age fixed effects.



Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly likes.
Parameter of interest: coefficient on dummies for weeks until/since model death.
Additional controls: model, calendar week, and model age fixed effects.

Figure 5.5: Utility and attention around OpenRouter usage obsolescence

Figure 5.5 plots the estimated coefficients $\hat{\tau}_{-15}, \ldots, \hat{\tau}_{15}$. Because we observe relatively few models and relatively few usage-obsolescence events, the estimates are imprecise. Although the estimates are not statistically significant, they suggest some decline in downloads and likes after a model becomes usage-obsolete on OpenRouter. 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. Appendix Figure C.1 estimates the same specification without controls for model age, calendar time, and model identity and shows larger negative effects on downloads and likes. 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 and likes 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.

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

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.

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

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.

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.

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.

Norton, E. C. (2022, September). The inverse hyperbolic sine transformation and retransformed marginal effects. *The Stata Journal: Promoting communications on statistics and Stata 22*(3), 702–712.

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].

Appendix

A Data

A.1 Economically relevant models: downloads threshold

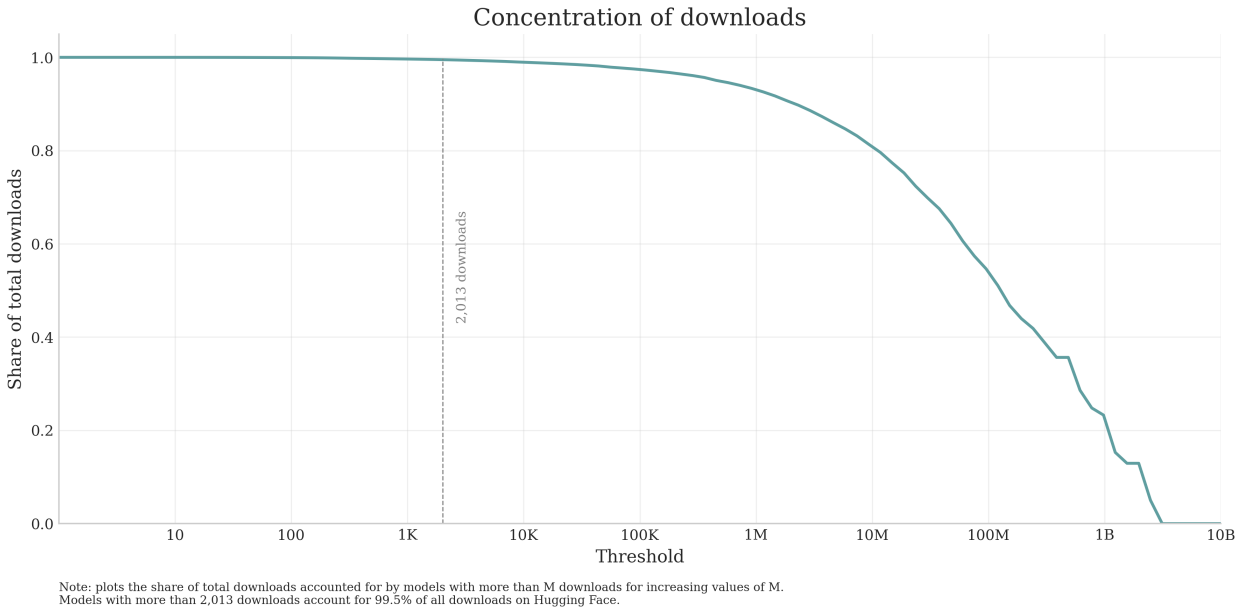


Figure A.1: Downloads concentration

A.2 Imputing missing download values

Hugging Face reports a rolling 30-day download count, which we denote by d_{jt}^{30} for model j on day t, and it sometimes reports cumulative downloads, denoted by c_{jt}. Because c_{jt} is missing in many daily snapshots, we construct an imputed cumulative series $\tilde{c}_{jt}$ as follows. We first drop duplicate (j,t) observations, keep only models whose observed cumulative downloads ever exceed 1,876, and process each model separately on the full daily grid between its first and last snapshot.

1. Initialize $\tilde{c}_{jt} = c_{jt}$ wherever c_{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 $\tilde{c}_{jt} - \tilde{c}_{j,t-30} \approx d_{jt}^{30}$ to backfill missing cumulative downloads. To allow for small timing mismatch across snapshots, form

$$b_{jt} = \text{mean}\left\{\tilde{c}_{jt} - d_{j,t-1}^{30}, \tilde{c}_{jt} - d_{jt}^{30}, \tilde{c}_{jt} - d_{j,t+1}^{30}\right\},$$

   where the mean is taken over available terms, and use b_{jt} to fill $\tilde{c}_{j,t-30}$ 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.

A.3 Matching OpenRouter and Hugging Face

Our goal is to measure model outcomes on Hugging Face using an "exogenous" definition of model death on OpenRouter. To achieve this 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 then 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.

A.4 OpenRouter and Hugging Face usage

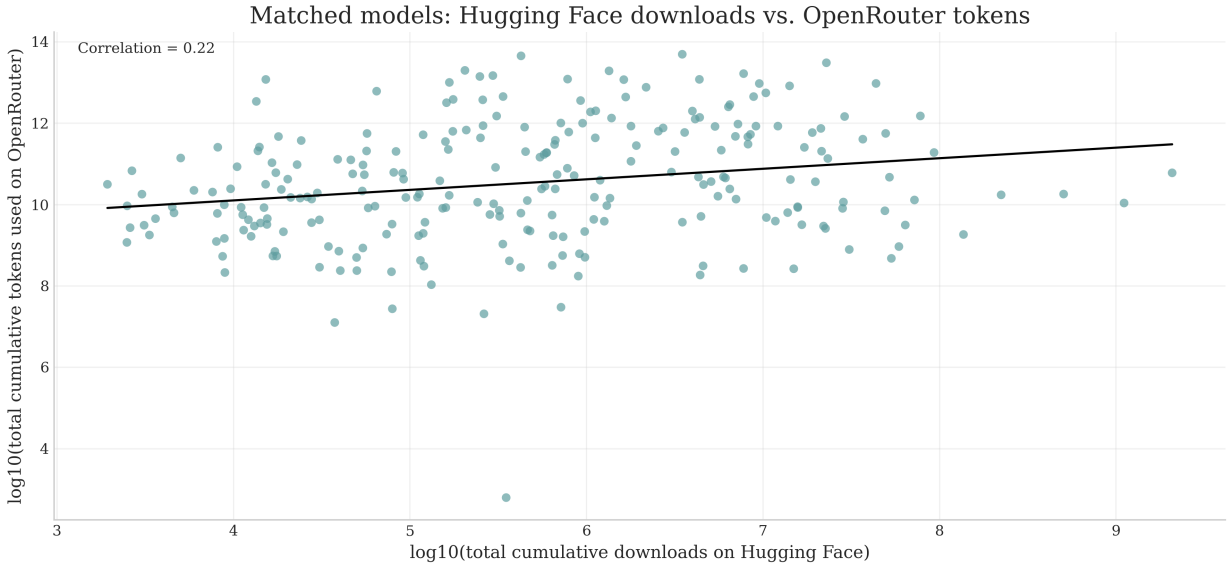


Figure A.2: Matched-model cumulative Hugging Face downloads and cumulative OpenRouter token use

A.5 Artificial Analysis

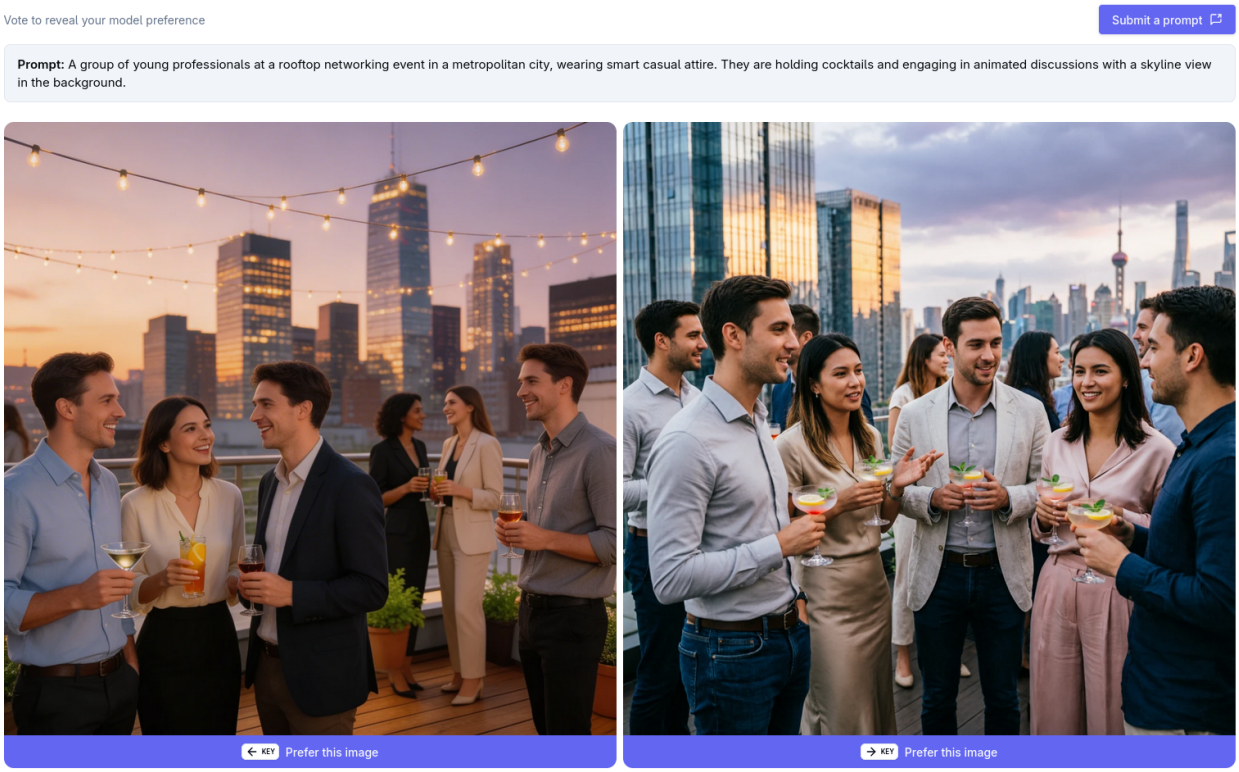


Figure A.3: Artificial Analysis Image Arena

B Life

B.1 Median model lifecycle

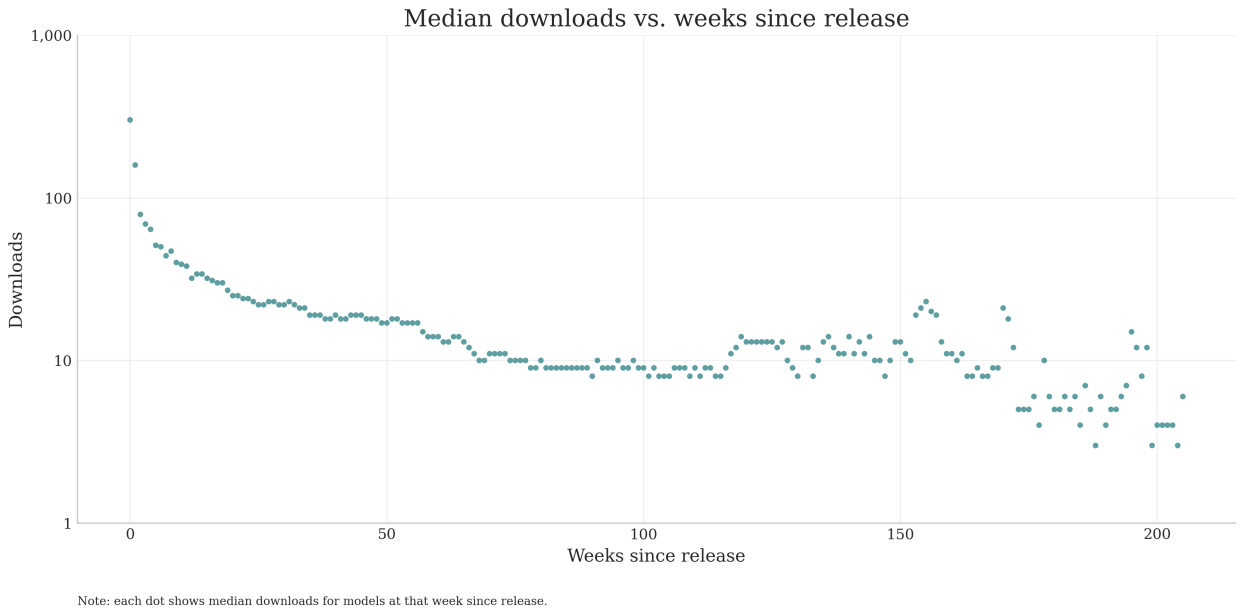


Figure B.1: Matched-model cumulative Hugging Face downloads and cumulative OpenRouter token use

B.2 Model lifecycle: alternative specifications

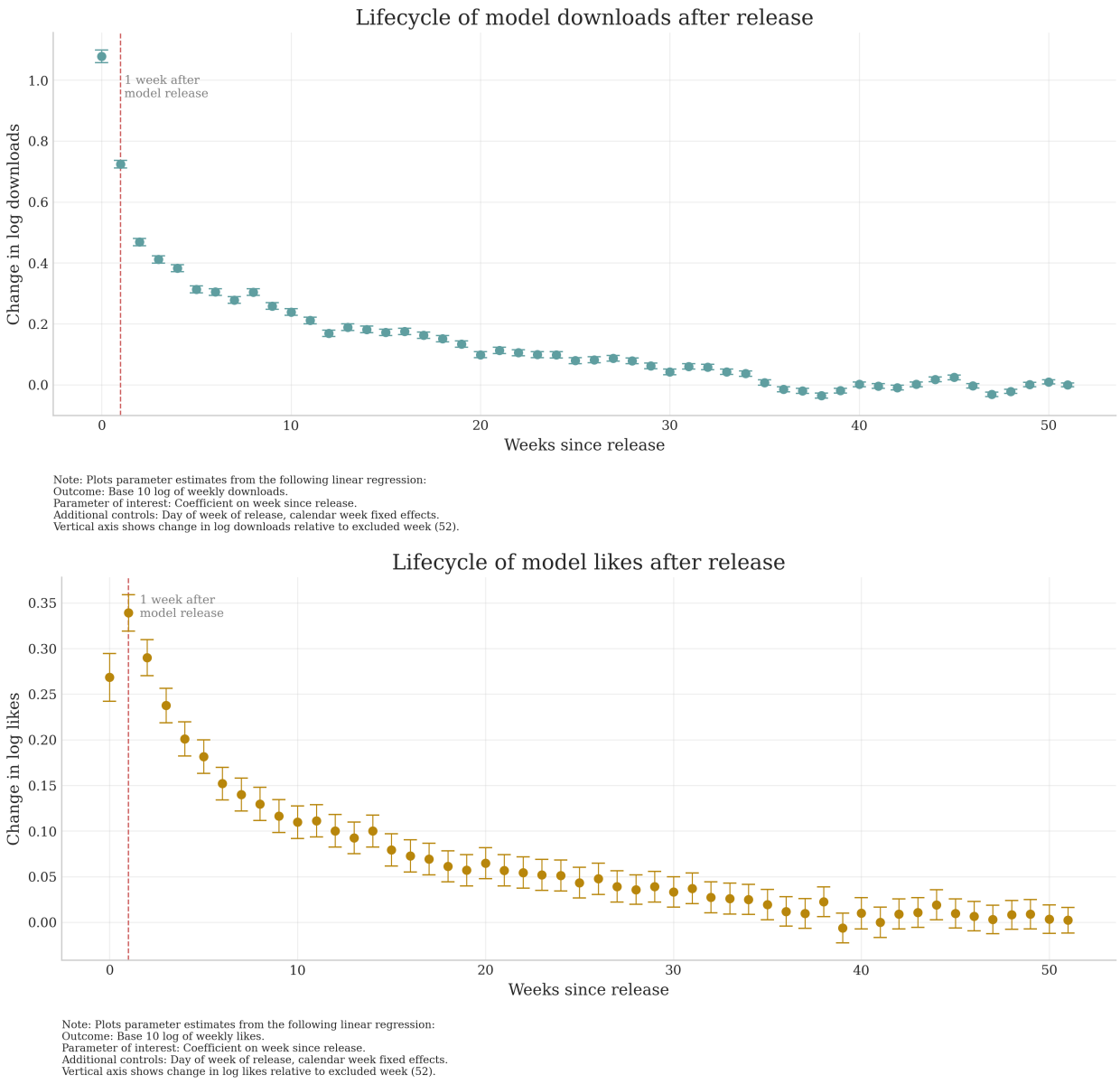


Figure B.2: Utility and attention at model release (no model fixed effects)

We estimate the following specification:

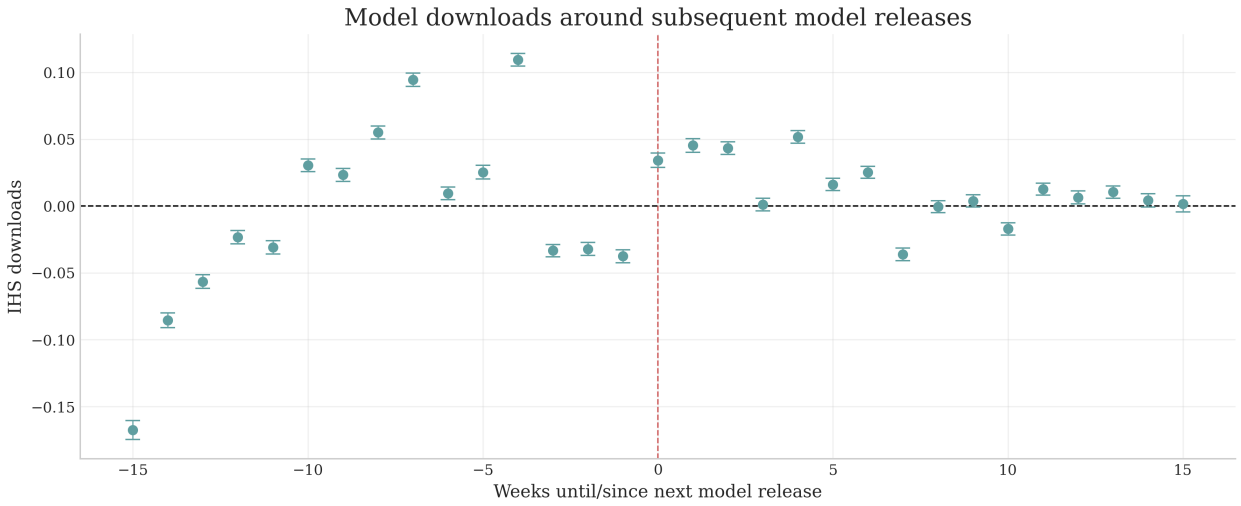
$$y_{jt} = \delta_t + \alpha_{t-t_0^j} + \alpha_{53}\mathbb{1}(t - t_0^j > 52) + \zeta_{dow}\mathbb{1}(t = t_0^j) + \varepsilon_{jt}$$

where

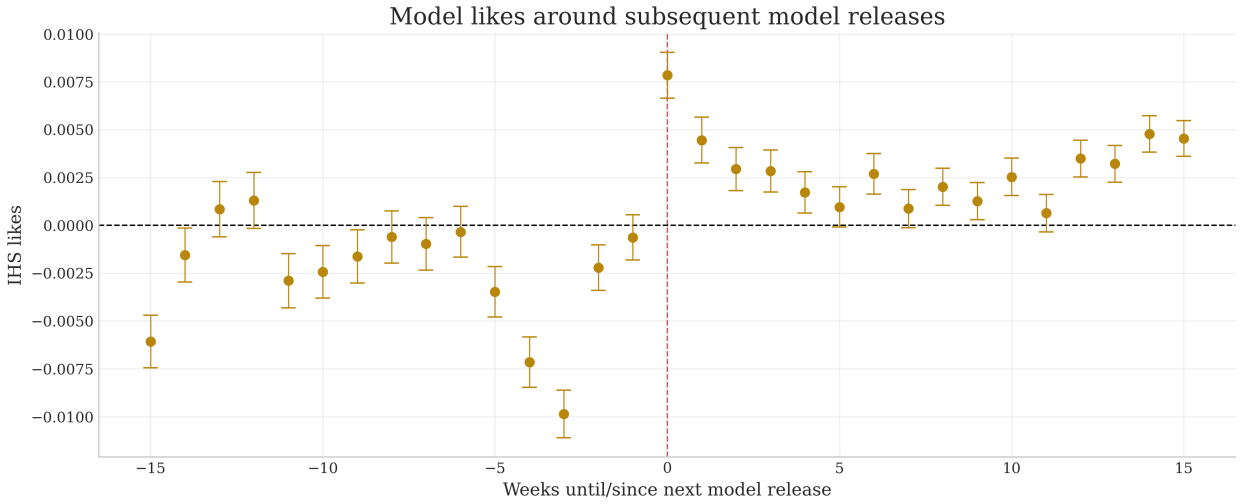
- $y_{jt} = \log_{10} p_{jt}$ where p_{jt} is the number of downloads model j received in week t.
- δ_t is a calendar week fixed effect.

- 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 $t - t_0^j \in \{0, 1, \ldots, 52\}$.
- ζ_{dow} is a day of week of release fixed effect.
- Standard errors are clustered at the model level.
- The excluded model age is α_{52}.

B.3 Effect of model releases: all models



Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly downloads.
Parameter of interest: coefficient on dummies for weeks until/since next model release.
Additional controls: model, calendar week, and model age fixed effects.



Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly likes.
Parameter of interest: coefficient on dummies for weeks until/since next model release.
Additional controls: model, calendar week, and model age fixed effects.

Figure B.3: Utility and attention at subsequent model releases

$$y_{jt} = \xi_j + \delta_t + \alpha_{t-t_0^j} + \sum_{j' \in \mathcal{J}_j} \sum_{k=-15}^{15} \tau_k \mathbb{1}\left(t - t_0^{j'} = k\right) + \varepsilon_{jt} \tag{B.1}$$

where

- y_{jt} is the inverse hyperbolic sine of the number of downloads model j received in week t.
- ξ_j is a model fixed effect.

- δ_t is a calendar week fixed effect.
- $\alpha_{t-t_0^j}$ is a model age fixed effect.
- $\mathcal{J}_j$ is the set of models from the author of model j released at least 30 days after model j.

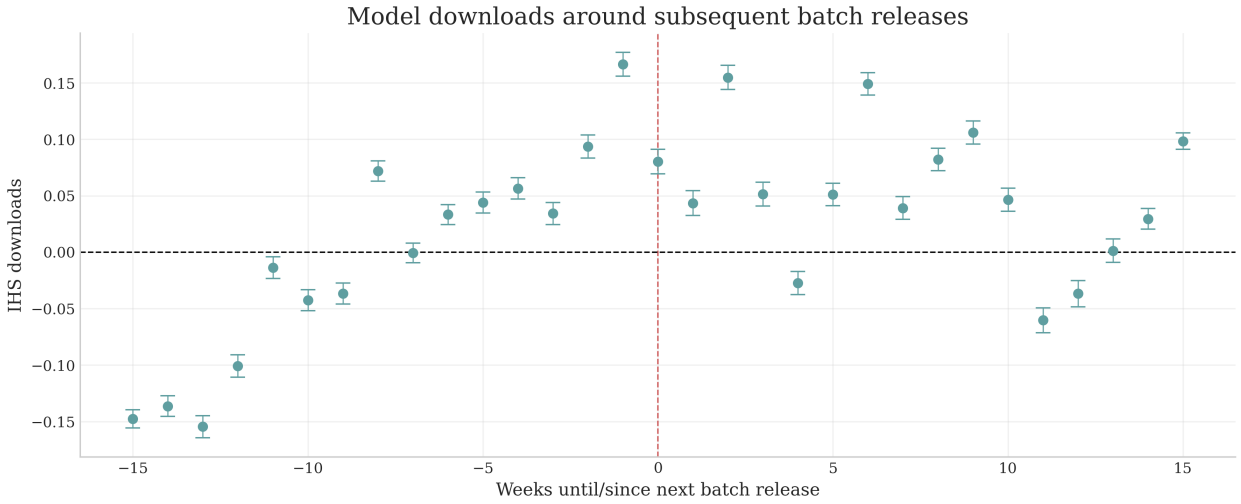
B.4 Effect of model releases: batch versions

Cohort specific version:

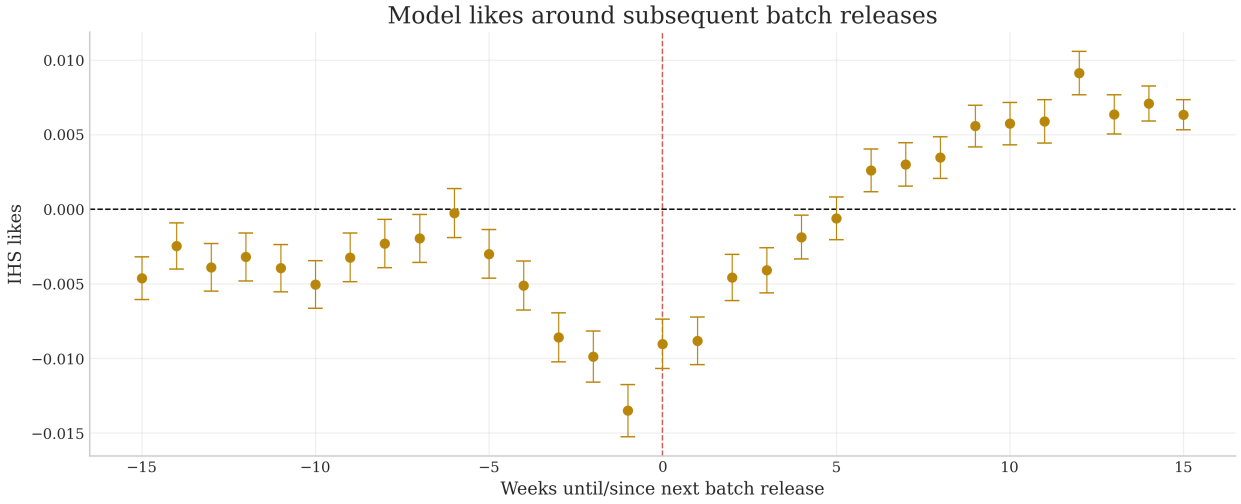
$$y_{jt} = \xi_j + \delta_t + \alpha_{jt} + \sum_{g=1}^{G} \sum_{j' \in \mathcal{J}_j^g} \sum_{k=-15}^{15} \tau_{gk} D_{gj't}^k + \varepsilon_{jt} \tag{B.2}$$

Special Case: Only next batch

$$y_{jt} = \xi_j + \delta_t + \alpha_{jt} + \sum_{g=1}^{1} \sum_{j' \in \mathcal{J}_j^g} \sum_{k=-15}^{15} \tau_{gk} D_{gj't}^k + \varepsilon_{jt} \tag{B.3}$$

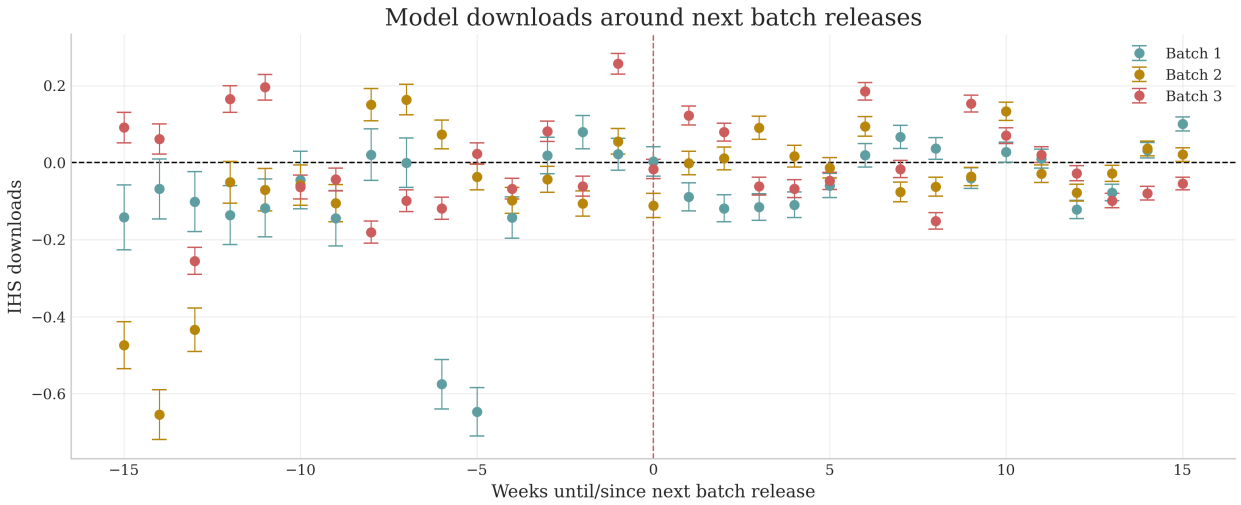


Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly downloads.
Parameter of interest: coefficient on dummies for weeks until/since next batch release.
Additional controls: model, calendar week, and model age fixed effects.

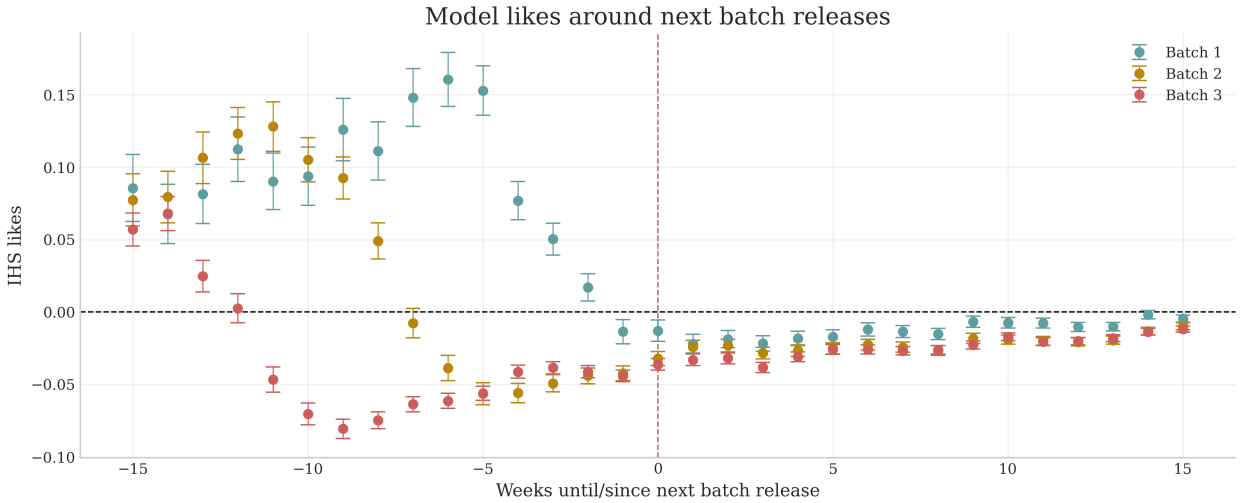


Note: Plots parameter estimates from the following linear regression:
Outcome: Inverse hyperbolic sine of weekly likes.
Parameter of interest: coefficient on dummies for weeks until/since next batch release.
Additional controls: model, calendar week, and model age fixed effects.

Figure B.4: Effect of batch release on model utility and attention



Note: Plots parameter estimates from linear regressions.
Outcome: Inverse hyperbolic sine of weekly downloads.
Parameter of interest: coefficient on dummies for weeks until/since next batch release.
Additional controls: model, calendar week, and model age fixed effects.



Note: Plots parameter estimates from linear regressions.
Outcome: Inverse hyperbolic sine of weekly likes.
Parameter of interest: coefficient on dummies for weeks until/since next batch release.
Additional controls: model, calendar week, and model age fixed effects.

Figure B.5: Effect of batch release on model utility and attention by batch

C Death

C.1 Death event study: no controls

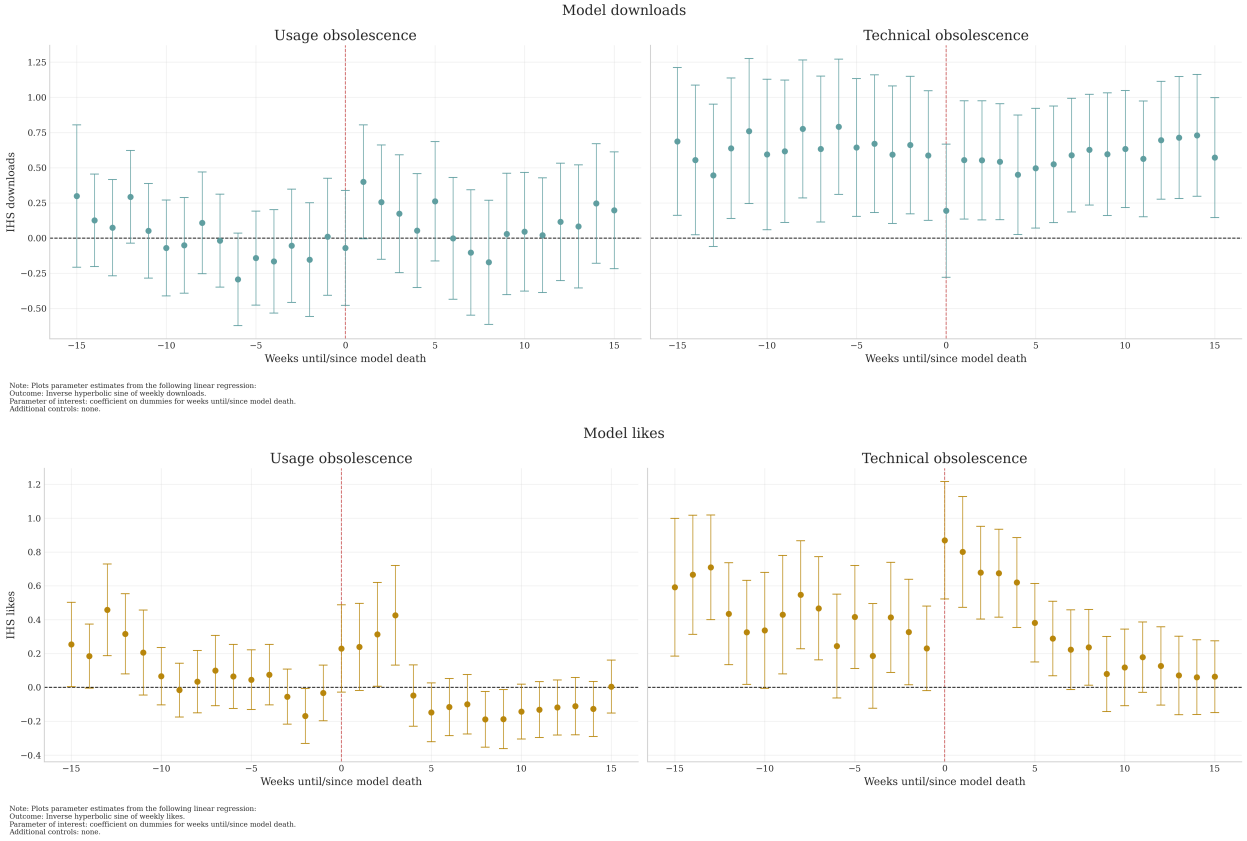


Figure C.1: Utility and attention around model death (no controls)

We estimate the following specification:

$$y_{jt} = \alpha + \sum_{k=-15}^{15} \tau_k \mathbb{1}\left(t - t_d^j = k\right) + \varepsilon_{jt}$$

where

- y_{jt} is the inverse hyperbolic sine of the number of downloads model j received in week t.
- α is an intercept.
- t_d^j is the calendar week in which model j dies on OpenRouter according to our definition in the main text.
- Standard errors are clustered at the model level.

Unlike Equation 5.5, this specification does not include model fixed effects, calendar week fixed effects, or model age fixed effects. It therefore shows the raw relationship between Hugging Face activity and weeks relative to model death.