Product Differentiation and Consumer Surplus in the Market for LLMs: Evidence from Software Developers*

Mert Demirer† Leon Musolff‡ Max Schnidman§

November 7, 2026

For NBER IO 2027 Spring Meeting submission only.
Please do not distribute.

Abstract

How differentiated are large language models (LLMs), how much consumer surplus do they create, and does it change through better models or lower prices? We answer these questions by estimating a structural demand model using GitHub Copilot data on developers' daily model choices, together with a randomized experiment that temporarily cut the prices of individual LLMs. The model treats LLMs as differentiated products under a monthly allowance, extending the single-product three-part-tariff model of Nevo et al. (2016) to a menu of 39 LLMs from six providers. When a frontier model's price rises, two-thirds of lost requests go to the same provider's other models; when a cheap model's price rises, developers instead reduce usage overall. Consumer surplus averages three times the subscription fee but is highly heterogeneous. Entry of new models raised consumer surplus by 7% over ten months, half through new models' improved capabilities and half through their lower price per unit of intelligence.

JEL: L11, L13, L86, D12, O33

Keywords: large language models, product differentiation, demand estimation, three-part tariffs

*We thank Shane Greenstein, Aviv Nevo, Sida Peng, Rob Porter, and seminar participants at MIT Sloan and Microsoft Research for helpful comments. Roman Shrestha provided excellent research assistance. Disclosures: Mert Demirer and Leon Musolff both previously held a postdoctoral research position at Microsoft and now work as paid research consultants for the company. Max Schnidman is employed by Microsoft.

†MIT, mdemirer@mit.edu

‡Wharton, lmusolff@wharton.upenn.edu

§Microsoft, maxschnidman@microsoft.com

1 Introduction

A market for large language models (LLMs) is emerging. Dozens of providers offer hundreds of models through APIs and subscriptions, new frontier models arrive every few months, and the price of intelligence falls quickly (Demirer et al., 2025). This market has far-reaching implications for the economy, as LLMs are adopted across a growing range of economic tasks (Handa et al., 2025) and have been shown to raise the productivity of workers across many occupations (Cui et al., 2026; Demirer et al., 2026; Chen and Stratton, 2026; Sarkar, 2026). Yet despite its size, impact and pace, little is known about demand for LLMs and the structure of this market.

In this paper, we study three questions about the market for LLMs: how differentiated LLMs are, how much consumer surplus they create, and how that surplus depends on the structure of the industry. We answer these questions with data from GitHub Copilot on software developers' daily model choices across several providers, together with a randomized experiment that temporarily cut the prices of individual models. We first present descriptive evidence on how developers substitute between models and providers, how they adopt new models, and whether their choices show learning or habit formation. We then estimate a structural model of demand for differentiated products under a monthly allowance. This model extends the three-part-tariff model of Nevo et al. (2016) from a single good to a menu of models from several providers, connecting the literature on demand under nonlinear tariffs (Lambrecht et al., 2007; Grubb and Osborne, 2015) to the literature on demand for differentiated products (Berry et al., 1995; Nevo, 2000, 2001).

GitHub Copilot is a popular AI coding assistant that offers, for a monthly subscription fee, a menu of LLMs from several providers, including Anthropic, OpenAI and Google. For each request, the developer picks a model from the menu, and the request is charged against a monthly allowance at its model's price in points. All models draw on one allowance, so a model's effective price depends on how much of the allowance a developer has left. The menu and prices also change often: new models enter every few weeks and prices are revised. Because developers choose among competing models within one application, we observe substitution between them directly. We observe data on all individual subscribers to this platform between November 2025 and August 2026.

The descriptive evidence from observational and experimental data points to three patterns in how developers use LLMs. First, developers spread their use across models and providers. In a typical month, 75% of developers use more than one model and 56% use models from more than one provider, although on any given day they usually work with a single model. Second, use is intermittent and highly volatile: the average developer is active on 33% of days, and her requests vary more from one day to the next than they

do across developers. Third, new models are adopted quickly but not by everyone: within a month of launch, the share of developers who have tried a new model ranges from 0% to 39% across the 23 launches in our sample. Adoption is about twice as high when the new model comes from the provider a developer already uses most: 21% against 10%, suggesting provider-specific preferences.

The data also suggest that developers differ in their sophistication. Many developers are myopic: they treat the allowance as free until it runs out, and many of them then stop rather than pay for overage. Others are forward-looking and respond to the shadow price of the allowance. Finally, we find no sign of learning or habit formation: developers drawn to a model by a temporary price cut return to their previous models within weeks of the price cut ending, so their preferences for models look stable rather than formed by use.

We then develop a structural model of demand for LLMs that builds on this evidence. Each day, a developer decides whether to code, which model to use, and how many requests to make under the monthly allowance. On an active day, a developer chooses one model and a number of requests. The value of requests is concave, reflecting diminishing marginal returns from additional requests, and each request has a time cost, so a developer weighs the value of another request against both its (shadow) price and the time it takes. Each request draws on the allowance at its model's price; once the allowance is exhausted, subsequent requests are billed as overage. Developers can be forward-looking, myopic, or overage-averse. Forward-looking developers solve a finite-horizon dynamic problem within each billing month and weigh each request against the expected value of the allowance it uses up; myopic developers treat the allowance as free until it is exhausted and then pay for overage; and overage-averse developers also treat the allowance as free but never pay for overage.

The value a developer gets from a model has three parts. A model-specific quality component common to all developers captures vertical differentiation: some models are better for everyone. Two random coefficients capture horizontal differentiation: developers differ in how much they value a specific provider's models and in how much they care about benchmark performance, so a developer with a strong taste for one provider values all of its models more, and one with a strong taste for benchmark performance values the frontier models most. Finally, a persistent match value between each developer and each model captures horizontal differentiation at the model level, reflecting the possibility that a particular model may suit a specific developer's working style or choice of coding language better than its rivals.

We estimate the model by simulated method of moments. For given candidate parameters, we simulate a large population of developers, solving each developer's dynamic

problem under the menus and prices she faced, and simulating her daily choices over the ten months of our sample. We then choose the parameters so that the simulated choices reproduce moments computed from the data on usage, on responses to prices and behavior at the allowance, on the persistence of model choice and the adoption of new models, and on the request shares of the models.

The estimates allow us to quantify how differentiated LLMs are. First, demand for individual models responds little to their prices: the own-price elasticities of the frontier models average -1.5 and those of the cheaper models are between -0.01 and -0.70, because the time cost of a request is a large part of its full cost and price matters little to developers far from the allowance. When a frontier model's price rises, developers mostly switch to another model, usually of the same provider; when the price of one of the cheapest models rises, they mostly send fewer requests. Second, LLMs are differentiated more by model than by provider. Quality varies more within providers than across them, because each provider sells both frontier and cheap models. Tastes, too, differ more by model than by provider.

The estimates also speak to how much developers are willing to pay for intelligence. The value of AI models is usually measured through benchmarks that score models on standardized tasks (Jimenez et al., 2024; Stanford HAI, 2025), and little is known about how well these scores capture what users value. Our estimates of model quality measure how much developers are willing to pay for each model, as revealed by their choices and usage. Benchmark scores are positively but weakly related to quality: the Artificial Analysis Coding Index, the benchmark closest to what developers do, explains only 10% of the variation in quality across models (Figure 8). The implication is that much of what developers value in a model, such as speed, coding style and tool use, lies outside benchmark scores, and a price index for AI that adjusts only for benchmark performance understates the quality improvements that developers value.

We turn next to the consumer surplus that LLMs create for developers. Our estimates imply that it is large: the average developer-month yields a surplus of \$30.9, about three times the monthly fee of \$10, and 76% of developer-months yield more than the fee. It is also very unequal: the median developer-month yields \$18.8, developers in the top decile of expected demand gain \$110.3 a month, 26 times as much as those in the bottom decile, and the top decile accounts for 36% of total surplus.

Surplus also grew quickly over our sample. Between November 2025 and August 2026, surplus per developer rose by 7%, with the gain concentrated among the heaviest users: about 31% for developers around the 99th percentile of expected demand against 2% at the median. We decompose this gain into three possible sources: changes in the prices of

models already on the menu, the higher quality of new models, and the prices at which those new models entered, relative to what their quality would imply. Price changes of existing models play no role, because providers rarely reprice a model once it is released. The entire gain comes from new models, 46% from their quality and 54% from their lower price per unit of intelligence.

Finally, we ask how this surplus would change if the industry were to integrate vertically. Today, tools that carry models from all the major providers, such as GitHub Copilot and Windsurf (Devin), are common, but the providers are building their own coding tools through which they sell their models directly. If each provider sold its models only through its own plan, developers would sort to the provider they already use most, and a forward-looking developer would lose $0.45 per month, 1.4% of her surplus, because models within a provider are close substitutes for most of what she does. Counterintuitively, myopic developers benefit as restricting the choice set prevents them from over-spending on pricey models early in the month.

Relation to Literature Our work relates to three strands of literature. First, a growing literature studies demand for LLMs.[1] Demirer et al. (2025) document the supply, prices and usage of LLMs sold through APIs, Egan et al. (2026) estimate demand for consumer AI subscriptions from card transactions, which show which providers a consumer pays but not which models she uses, and Caoui (2026) estimates preferences for individual LLMs from randomized pairwise comparisons of model responses at zero prices. We differ from these papers in the data and in what we estimate. We observe developers' daily choices of model and quantity at the prices they face, among models from several providers within one subscription. This lets us estimate substitution between individual models, within and across providers, and the consumer surplus each model creates.

Second, our model builds on demand estimation under three-part tariffs. Lambrecht et al. (2007) study how uncertainty about usage shapes the choice of Internet plans, Grubb and Osborne (2015) estimate a model of cellular plan choice, usage and learning with biased beliefs, and Nevo et al. (2016) use the shadow price created by broadband allowances to estimate a dynamic model of usage within the billing cycle. In these papers, the allowance covers a single good, so approaching the limit changes only the amount consumed. In our setting, the allowance is shared across differentiated products sold at different prices, so the shadow price also affects which product is chosen. This links the literature on three-part tariffs to the literature on differentiated-product demand (Berry et al., 1995; Nevo, 2000, 2001). Like Grubb and Osborne (2015), we allow consumers to depart from fully forward-looking behavior, here through myopic and overage-averse consumer types.

[1] Bergemann et al. (2026) study how LLM providers should design menus of token budgets.

Finally, we contribute to a long literature that measures the value of product innovation and new goods, from CT scanners to personal computers (Trajtenberg, 1989; Hausman, 1996; Petrin, 2002; Goolsbee and Petrin, 2004; Eizenberg, 2014). For generative AI, Brynjolfsson et al. (2026) measure how much consumers value access to it as a whole, using online choice experiments that ask consumers how much they would need to be paid to give it up for a month. By estimating demand for individual models, we measure the consumer surplus from LLMs and decompose it: how much each model adds to the menu, how much is lost when a provider's models are removed, and how much developers gain from having several providers' models in one subscription.

2 Background

Developers in our sample use LLMs through a GitHub Copilot subscription, which bundles models from several providers and charges for them through a monthly allowance and overage prices. This section describes the product and its model menu, the request-based tariff in place through May 2026, the token-based billing that replaced it on June 1, 2026, and a price experiment. Appendix A reports menus and prices month by month.

2.1 GitHub Copilot and Its Menu of Models

GitHub launched Copilot in June 2022 as an autocomplete tool that suggests code as a developer types. Since then it has added chat and agent modes, in which the developer writes a prompt in the editor and an LLM answers questions (chat) or edits the files directly (agent). We call each such prompt a *request*. When sending a prompt, the developer picks the model for that request. The available models come from different providers, including Anthropic, OpenAI and Google, and GitHub serves them inside the same editor, with the same interface, so a developer can switch between models easily.

Copilot is available through both individual and enterprise subscriptions. We focus on individual subscriptions, because enterprise subscriptions are managed partly by the developer's company, which decides which models its developers can use, pays for their usage, and can pool usage across its developers. Individual subscriptions come in several tiers, including Copilot Free, a limited plan at no charge, a Student plan, Copilot Pro at $10 per month, Copilot Pro+ at $39 per month, and, from June 2026, Copilot Max at $100 per month. We study monthly subscribers to Copilot Pro.[2]

Between November 2025 and August 2026, 39 models from six providers are available on the Pro menu. Figure 1 shows when each of them is included in the menu, grouped by

[2]Developers can also reach the same models outside Copilot, through providers' own tools such as Claude Code or through the providers' APIs.

provider, and Table OA-9 lists their benchmark scores and prices. The menu mixes each provider's flagship models with smaller, cheaper models and with models specialized for coding, and the models differ widely in capability: scores on SWE-bench Verified, a benchmark of real software-engineering tasks (Jimenez et al., 2024), range from 8.7% for GPT-4o mini to 81.4% for Claude Opus 4.6. The menu also changes often, as GitHub adds the models released by the providers and retires older models. Over our sample there are 31 entries and 20 exits. Because these changes alter the choice set, they are an important source of identifying variation in our model.

2.2 Pricing

Copilot Pro is priced as a three-part tariff (Lambrecht et al., 2007): a monthly fee buys an allowance of usage, each request draws down the allowance at a price that depends on the model, and requests beyond the allowance are billed as overage. The allowance resets on the first day of each month, and unused allowance does not carry over.

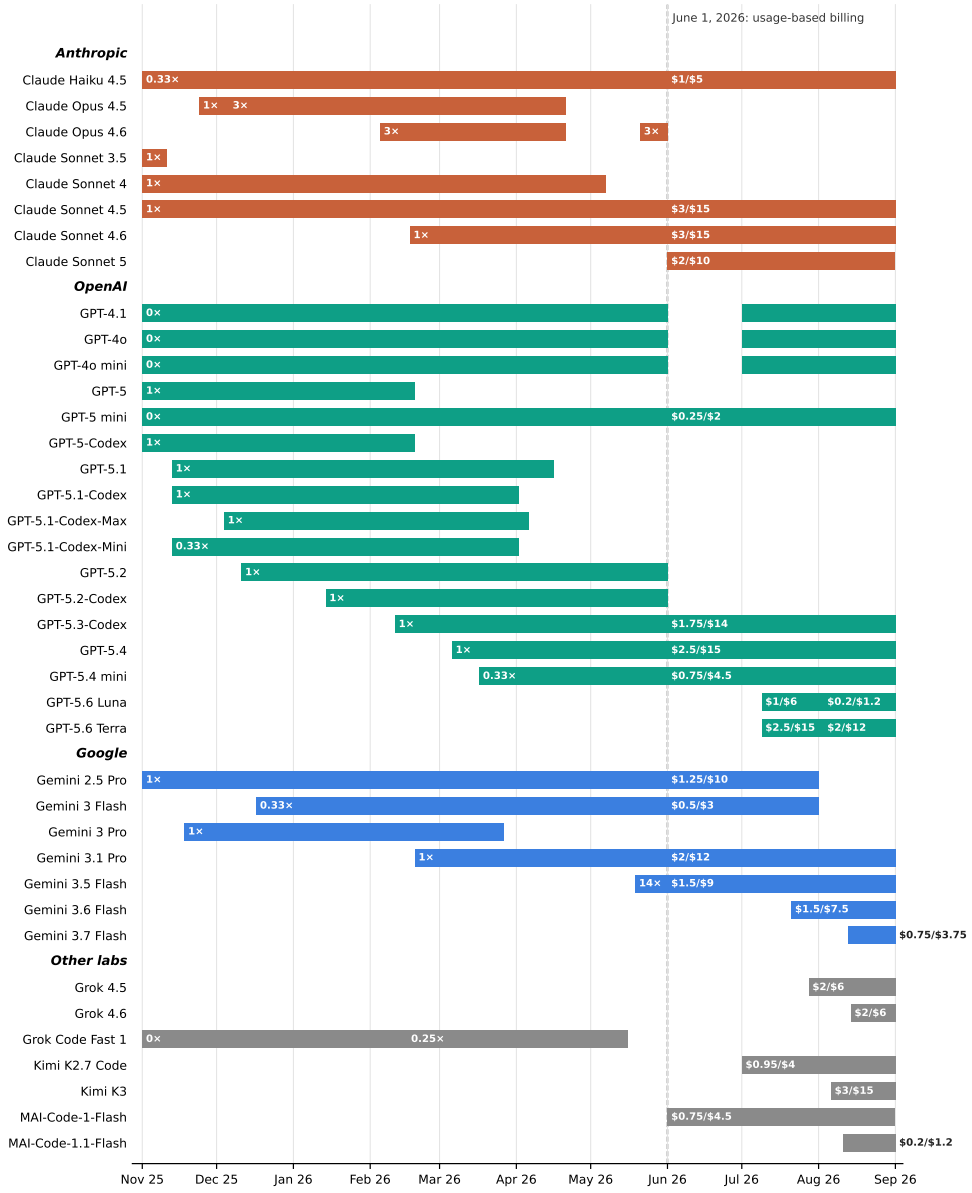
Between November 2025 and May 2026, Copilot Pro charges a monthly fee of $10, includes an allowance of 300 *points* per month, and charges $0.04 for each point beyond the allowance.[3] Models differ in how many points a request uses: some models, such as GPT-4.1, GPT-4o and GPT-5 mini, are free and cost 0 points; small models such as Claude Haiku 4.5 and GPT-5.4 mini cost 0.33 points; most flagship models cost 1 point; and the most capable models cost more, such as Claude Opus models at 3 points. Prices rarely change during this period, and the few changes occur when GitHub ends a reduced promotional price at which it offered a new model for its first weeks on the menu. Table 1 summarizes the tariff, Table OA-9 reports each model's price, and Table OA-5 summarizes menus and prices by month. Developers can also let Copilot choose the model for each request, in what GitHub calls auto mode; requests in auto mode receive a 10% discount on their price.

Developers can track their usage closely. The status dashboard in the editor shows the share of the monthly allowance used, the price of each model in points is shown in the editor's model settings and on each response, and Copilot notifies the developer when the allowance runs out. Developers therefore know how much of the allowance remains when they choose a model.

Once a developer has used her 300 points in a month, further requests to premium models are billed as overage, at $0.04 per point, while the free models remain free. To make requests to premium models beyond the allowance, a user needs to set an overage

[3] GitHub calls these points premium request units. We call them points to distinguish them from the requests a developer sends to a model, which can use several points each.

Figure 1: Models on the Copilot Pro Menu, November 2025 to August 2026



Notes: This figure shows the days on which each model is on the Copilot Pro menu between November 2025 and August 2026, with models grouped by provider (Anthropic, OpenAI, Google and other providers). Each bar marks the days on which the model is on the menu; a break means the model is off the menu for a period. Menus are those we use in estimation, so dates can differ by a few days from GitHub's announcements (Appendix A). Labels give each model's price when it enters the menu and whenever the price changes: before the billing change, the number of points a request uses; after the billing change, GitHub's list price in dollars per million input and output tokens for prompts in the default context tier. List-price changes within a month are dated by the first archived copy of GitHub's pricing page that shows them. The dashed line marks June 1, 2026, when GitHub switched Copilot from request-based to token-based billing (Section 2.2). Benchmark scores, prices in points, and prices after the billing change for each model are in Table OA-9.

budget and can then use premium models up to that budget. This budget is set to $0 by default, so the user must take an explicit action to keep using premium models beyond

Table 1: The Copilot Pro Tariff before and after the Billing Change

	Points (Nov 2025–May 2026)	Token-based billing (Jun–Aug 2026)
Unit of account	Point	AI credit ($0.01)
Monthly fee ($)	10	10
Included usage	300 points	$15
Overage price	$0.04 per point	List price per token
Price of a request	Points per request, by model	Tokens × list price per token
Included (free) models	GPT-4.1, GPT-4o, GPT-4o mini, GPT-5 mini, Grok Code Fast 1 (until Feb. 16, 2026)	None
Default overage budget ($)	0	Not documented[a]
Auto-mode discount (%)	10	10
Promotions	Reduced launch prices	Reduced list prices (July, August)
Allowance reset	First of month	First of month

Notes: This table summarizes the terms of Copilot Pro under request-based billing (November 2025 to May 2026) and token-based billing (from June 1, 2026), from GitHub's documentation and announcements (Section 2.2). Each model's price in points and list price over time are shown in Figure 1. [a]GitHub's documentation for individual plans does not state the default budget for additional usage after the billing change; we read it as $0, as under the previous tariff.

the allowance.

The June 2026 billing change. On June 1, 2026, GitHub switched monthly Copilot subscriptions from request-based to token-based billing; annual subscribers, whom we exclude (Appendix A), stayed on request-based billing. Under the new tariff each request is charged for the tokens it uses, including input, output and cached tokens, at the API list price of the model that serves it. The fee stays at $10, and it now gives the developer $15 worth of tokens per month, valued at the models' list prices. Copilot also no longer offers free models. Overage works as before: a developer who exhausts the $15 can keep using Copilot by setting a budget for additional usage, which is billed at the same list prices.[4] Developers can still track their usage: they see the credits ($0.01 each) that each request uses, and they know the per-token prices before sending a request.

The billing change altered both the level of prices and their relative structure. Figure 1 shows each model's price in points before June 1 and its list price per million tokens after it. Given the number of tokens a typical request uses, the new tariff amounts to a price increase for almost every model. The increase is larger for premium models that

[4]GitHub's documentation for individual plans states that a developer "can continue working by setting a budget for additional usage" but does not state the default budget. We read this as a $0 default, as under the previous tariff.

use more tokens per request, such as reasoning models, which generate long chains of intermediate output, so models that cost the same number of points before June now differ substantially in price. This price change is an important source of variation for identifying price sensitivity and substitution between models. It also raises a measurement problem. Before June a request has a fixed price in points, but after June its price depends on its token consumption, necessitating the construction of a per-request price (see Section 5.5).

2.3 The Price Experiment

In September 2026, GitHub ran a randomized price experiment on Copilot Pro. Developers with a monthly Pro subscription were randomly assigned to a control group or to one of eight treatment groups. Each treatment group received a discount of 10 or 25% on the list price of one of four models: GPT-5.6 Luna, GPT-5.6 Terra, Haiku 4.5 and Sonnet 5. Treated developers were notified of the discount on their model; control developers received no notice and faced the usual prices. Discounts applied to both consumption within the allowance (reducing the rate at which requests used up the budget included in the subscription) and to overage consumption (directly reducing the developer's overage bill).

The discounts began on September 3 and ended on September 20. For each developer in the experiment, we observe her daily requests to every model from four weeks before the discounts began to five weeks after they ended, so we can compare treated and control developers before, during and after the discount.

3 Data

This section describes our data sources and sample restrictions, and presents summary statistics.

3.1 Data Sources

Our main data are developer-level usage records for a large subsample of GitHub Copilot's individual Pro and Pro+ subscribers.[5] Detailed data construction procedures are provided in Appendix A.

The usage records run from November 1, 2025 to August 31, 2026. For each anonymized developer, we observe the number of requests she sends to each model every day, the input, output, and cache tokens these requests use, and the cost they are charged against her

[5]Our data-use agreement prevents us from disclosing commercially sensitive information, such as the number of Copilot subscribers or the market shares of models and providers. We therefore report many quantities in relative terms rather than in levels.

allowance, in points before the billing change and in dollars of list-price usage afterwards.[6] These records give us the two developer-level objects our model needs. The first is the chosen model and the number of requests on each day. The second is cumulative spend since the start of the month, which determines how much of the allowance remains and is therefore the state variable of the developer's problem (Section 5).[7]

To construct model characteristics, we use data from Artificial Analysis: for each model, we obtain the provider that trained it, its release date, its list prices and its Coding Index. When a model or a score is not available from Artificial Analysis, we use the providers' model cards, and for a few models without a reported SWE-bench score we fill in the score of a closely related model. SWE-bench scores come from the providers, the benchmark maintainers and third-party evaluations; Appendix A lists the source for each model.

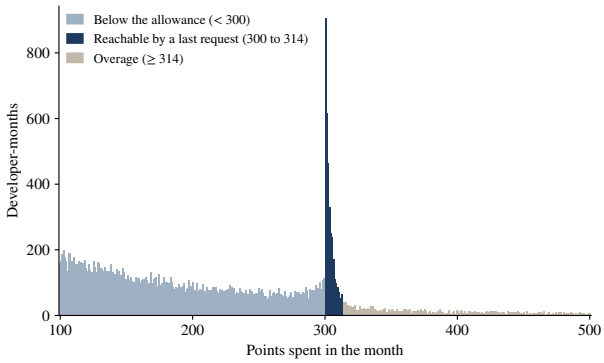
3.2 Estimation Sample

We keep developers who are present in every month and stay on the same individual monthly plan. This allows us to abstract from plan choice. We further exclude developers who send more than ten auto-mode requests in any month, because in auto mode Copilot rather than the developer chooses the model, and our model does not include auto mode. Finally, we keep only Copilot Pro subscribers.[8]

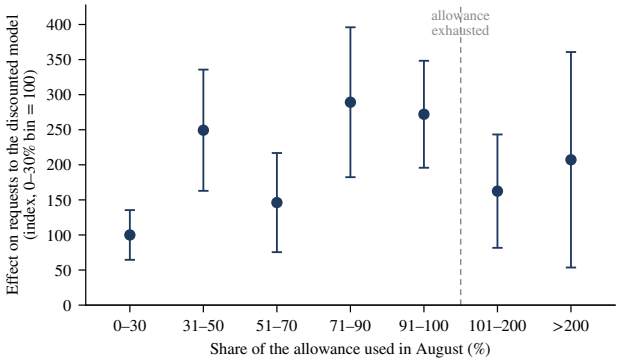
The resulting sample has three features. First, its developers are Copilot Pro subscribers rather than Pro+ or Max subscribers, who pay more for larger allowances, so the sample is unlikely to include the heaviest users of Copilot. Second, they are engaged with the product: they appear in the data in every month of our sample. Third, they subscribe as individuals rather than via their employer, so they bear the cost of their usage and may be more price-sensitive than enterprise users. Our estimates therefore describe individual

Figure 5: Behavior at the Allowance

(a) Monthly use around the allowance



(b) Response to the discount by use of the allowance



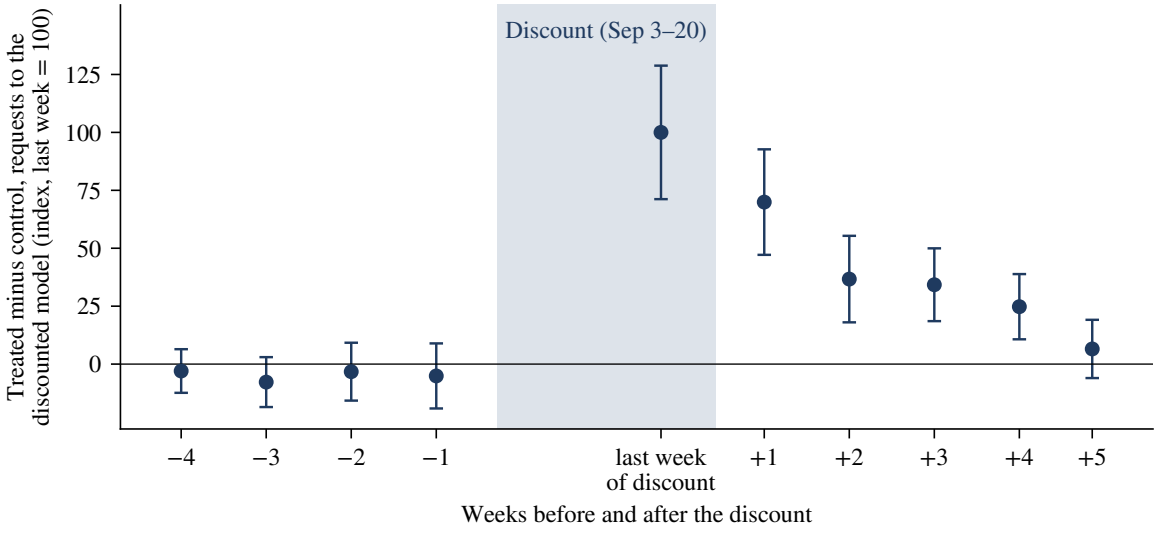
Notes: Panel (a) plots the distribution of points used per developer-month in the estimation sample between November 2025 and May 2026, when the allowance is 300 points, in bins of one point, for developer-months that use between 100 and 500 points. Dark bars mark use of 300 to 314 points, the range a developer can reach with one last request after the allowance is nearly exhausted, since the most expensive model on the menu costs 14 points per request. Panel (b) plots the effect of the discount on requests to the discounted model during the discount, in requests per developer-day and pooled across the four discounted models, by the share of the allowance a developer used in August 2026, the month before the discounts. Estimates are indexed so that the effect for developers who used at most 30% of the allowance equals 100; bars are 95% confidence intervals. Estimates compare treated and control developers, controlling for their use before the experiment; standard errors are clustered by developer.

since paying for it requires setting a budget; we call such developers overage-averse. Consumers on three-part tariffs are known to differ in this respect (Grubb and Osborne, 2015), and developers may differ as well. We now present two pieces of evidence on which of these types are present: how developers behave when they reach the allowance, and how their response to a price cut depends on their use of the allowance.

Panel (a) of Figure 5 plots the number of developer-months by points spent in the month, from 100 to 500 points. The number of developer-months declines smoothly up to 300 points and shows no dip just below the allowance. It then jumps: a large mass of developer-months ends just above 300 points, within reach of one last request, and beyond this range the density falls to a low level. Most of these developer-months contain no spending after the allowance is used up. This pattern indicates that some developers are myopic or overage-averse: they treat the allowance as a discontinuous jump in the price and stop consuming when they reach it.

Panel (b) of Figure 5 turns to the price experiment and plots the effect of the discount on requests to the discounted model by the share of the allowance a developer had used in the month before the experiment. Two patterns stand out. First, even developers who had used less than a third of the allowance sent more requests to the discounted model, which indicates that these developers face uncertainty about their usage and some probability of reaching the allowance. Second, while noisy, the response increases with

Figure 6: Requests to the Discounted Model before, during and after the Discount



Notes: This figure plots, by week, the difference in requests to the discounted model between treated and control developers, pooled across the four discounted models and indexed so that the difference in the last week of the discount equals 100. The four weeks before the discount, the last week of the discount and the five weeks after it are shown; the shaded band marks the discount period. Estimates control for developers' use before the experiment; bars are 95% confidence intervals based on standard errors clustered by developer.

the share of the allowance used. Because a price cut matters to a developer within the allowance only through its shadow price, both patterns suggest that some developers are forward-looking.

4.3 Evidence on Learning and Habit Formation

This section tests whether a temporary change in a model's price has a lasting effect on which model developers use. A temporary discount can change developers' choices permanently under several behavioral mechanisms, such as learning that raises the payoff from a model, switching costs and habit formation. The price experiment allows us to distinguish a lasting effect from a temporary one, because the discount was short-lived and ended on September 20, 2026.

Figure 6 follows the difference in requests to the discounted model between treated and control developers before, during and after the discount. The two groups move together before the discount, and once it starts, treated developers' requests jump relative to control. After the discount ends, the gap declines monotonically: within a week it has shrunk to 70% of its size in the last week of the discount, and by the fifth week it is no longer distinguishable from zero. Five weeks after the discount, developers had returned

to the models they used before.[10]

The shift toward the discounted models was therefore a response to their price, not a lasting change in preferences. We see no evidence that using a model for up to 18 days led to habit formation or taught developers anything that changed their subsequent choices. Developers' preferences for models instead look stable. This, together with the other evidence in this section, motivates our modeling choices, to which we turn next.

5 Model

We now develop a model of how developers choose between LLMs and decide how much to use them. The fitted model then allows us to measure substitution between models, express the value of models in dollars, and predict choices under tariffs and menus that developers have not faced.

On each day she codes, a developer chooses one model from the menu and a number of requests to send to it. Requests draw on her monthly allowance of points, and she pays for requests beyond the allowance at the overage price. A request using points has no dollar cost, but an opportunity cost in covering requests later in the month. A forward-looking developer therefore weighs the benefit of using points today against the value of preserving them to avoid reducing requests or paying overage prices. We call the marginal value of preserving a point of allowance the shadow price, which lies between zero and the overage price.

This shadow price links the developer's choices across days, and it depends on the remaining allowance, the days to the allowance reset, and her expected future demand. It also connects her model choice to her usage decision, because models use different amounts of points per request. As these points become more valuable, cheaper models become more attractive.

We develop the model in three steps: a daily problem that determines model choice and quantity at a given shadow price (Section 5.1), the dynamics within the month that determine the (evolution of the) shadow price (Section 5.2), and the heterogeneity across developers in how they treat the allowance and in how they value each model (Sections 5.3 and 5.4). Appendix B collects the notation, the full solution, and all proofs.

Our model builds on the model of usage under a three-part tariff in Nevo et al. (2016) and extends it in two directions. First, in Nevo et al. (2016) the allowance covers a single good, broadband content, so the shadow price of the allowance changes only how much a subscriber consumes. In our setting, as described above, the shadow price also determines

[10] Three of the four discounted models were replaced by a successor from the same provider after the promotion. Counting requests to the successor as requests to the discounted model gives the same picture.

which product the developer chooses. Second, developers differ in how they treat the allowance, and we allow for forward-looking, myopic, and overage-averse types. As in Grubb and Osborne (2015), these departures from the forward-looking benchmark are motivated by behavior near the allowance–e.g., bunching– that a model with only forward-looking consumers cannot generate.

We write the model in terms of the request-based tariff, which applies through May 2026; Section 5.5 describes how the same model applies under the token-based billing that replaced it in June 2026.

5.1 Environment and the Daily Problem

Developer i subscribes to GitHub Copilot and faces a three-part tariff: a fixed monthly fee, a monthly allowance of $\bar{C} = 300$ points, and an overage price of $p = \$0.04$ per point beyond the allowance. A request to model m uses α_m points. Requests draw down the allowance until it is exhausted, and further requests are charged at the overage price. Hence, after the allowance is exhausted, the price of a request to model m is $p\alpha_m$.

A month has T days, indexed by t. We divide each month into regimes, contiguous blocks of days with a fixed menu of models, $\mathcal{M}_r$, and fixed prices per request, α_{mr}; a new regime starts whenever a model enters or exits the menu or its price changes, and we denote the regime of day t by $r(t)$. At the start of each day, developer i is active with probability π_i, independently across days. If she is active, she observes her demand shock for the day, v_t, and then chooses a model $m_t \in \mathcal{M}_{r(t)}$ and a number of requests $c_t \geq 0$. The shock is $\ln v_t = \mu_i + \sigma_i z_t$, where z_t is drawn independently across days, capturing the fact that some days involve much more coding, or much harder coding, than others. If the developer is inactive, she sends no requests and her allowance is unchanged. Appendix A.4 lists the regimes, and Appendix B.4 describes how we solve the model across them.

The utility of developer i from sending c requests to model m on a day with shock v is

$$u_{im}(c; v) = v \, \frac{(\gamma_{im}\, c)^{1-\beta_i}}{1-\beta_i} - \kappa_i \, c, \tag{1}$$

where γ_{im} is the quality of model m for developer i, which we specify in Section 5.4, and $\beta_i \in (0, 1)$ governs curvature. The first term measures the value of c requests to model m; it rises with the day's demand shock v and with the model quality γ_{im}. Quality scales the value of each request, so a request to a better model is worth more than a request to a worse one. We assume that marginal utility declines with the number of requests. The second term captures the time cost of using the model, κ_i dollars per request, encompassing time spent writing prompts, waiting for output, and reviewing it. As in Nevo et al. (2016), $\kappa > 0$

implies that the developer has a satiation point even when requests are free: marginal utility declines while the time cost is constant, explaining why developers send only a finite number of requests to free models. Utility is quasilinear in money with a marginal utility of one, so u_{im} and κ_i are measured in dollars.

We emphasize that the structure of our choice model implicitly assumes that each developer uses a single model on each day. This makes the daily problem a discrete choice of model followed by a continuous choice of requests, as in models of discrete-continuous demand. We motivate this assumption with the evidence in Section 4.1, where we found that developers usually work with a single model on a given day. We discuss the assumption and its implications in Section 5.5.

Suppose for now that the developer values a point at a shadow price λ, which we derive in Section 5.2; λ lies between zero and the overage price p. The full marginal cost of a request to model m is then $\tau_m = \kappa_i + \lambda\alpha_m$, the time cost plus the value of the allowance the request uses up. On an active day, the developer chooses c to maximize her net payoff $u_{im}(c; v) - \lambda\alpha_m c$, which gives the number of requests in closed form[11]:

$$c^*_{im}(v, \lambda) = \gamma_{im}^{\frac{1}{\beta_i}-1} \left(\frac{v}{\kappa_i + \lambda\alpha_m} \right)^{1/\beta_i}. \tag{2}$$

Equation (2) implies that a developer with curvature β_i has a demand elasticity of $-1/\beta_i$ with respect to the full marginal cost of a request ($\kappa_i + \lambda\alpha_m$). The elasticity with respect to the money price alone, $\lambda\alpha_m$, is $-(1/\beta_i)\,\lambda\alpha_m/(\kappa_i + \lambda\alpha_m)$, which is smaller in absolute value than the elasticity with respect to the full marginal cost. Intuitively, money is only part of what a request costs the developer, so a developer whose time cost is large relative to the price of a request barely responds when that price changes.

Substituting Equation (2) into the net payoff gives the value of using model m,

$$\bar{u}_{im}(v, \lambda) = \frac{\beta_i}{1 - \beta_i}\, v^{1/\beta_i} \left(\frac{\gamma_{im}}{\kappa_i + \lambda\alpha_m} \right)^{(1-\beta_i)/\beta_i}. \tag{3}$$

Because the demand shock v enters this value multiplicatively and separately from the model's quality and cost, the ranking of models does not depend on it. It follows that, at a given shadow price λ, the developer chooses the model with the highest log quality minus log (time cost inclusive) price,

$$m^*_i(\lambda) = \arg\max_{m \in \mathcal{M}_r} \Big[\ln \gamma_{im} - \ln(\kappa_i + \lambda\alpha_m) \Big]. \tag{4}$$

[11] We treat the number of requests as a continuous choice, although requests in the data are integers.

Equation (4) is exact when the shadow price does not depend on the day's choice, as in overage ($\lambda = p$) or for a myopic developer inside the allowance ($\lambda = 0$). Otherwise, and in particular when the developer exhausts the allowance exactly, the shadow price differs across models, and she compares the value of each model directly (Section 5.2); our solution maximizes over models and requests exactly (Appendix B.5).

Model choice is a comparison of quality per dollar of full marginal cost, and within a month this comparison changes only when the shadow price λ or the menu changes. When λ rises toward p, as it typically does when the allowance runs down, the full marginal cost of a model with a high price rises proportionally more than that of a cheap model, so the developer re-ranks models toward cheaper ones.

5.2 Dynamics within the Month

We now solve for the shadow price of the allowance. Denote by R_t the points remaining at the start of day t, with $R_1 = \bar{C}$. A day on which the developer sends c_t requests to model m_t uses $q_t = \alpha_{m_t r(t)} c_t$ points, of which $\max\{q_t - R_t, 0\}$ are paid at the overage price p, and leaves $R_{t+1} = \max\{R_t - q_t, 0\}$ points for the rest of the month.

As pointed out above, menus and prices can change mid-month, which raises the question of what developers expect about future menus. We assume that the developers expect the current menu to persist, and hence on each day the developer solves the rest of the month as if the current menu and prices held until the end of the month. When there is a change, she re-optimizes from her remaining allowance. Under this assumption, it is sufficient for the developer to track only the remaining allowance and the calendar day, (t, R_t). Model launches in Copilot typically take effect on the day they are announced, so treating them as unanticipated seems natural. The largest change, the June 2026 billing change, was announced in advance but falls at a month boundary, so its anticipation would not affect the within-month problem: nothing carries over across months because the allowance resets.

Define the shadow price of a point as

$$\lambda_{it}(R_t, q_t) = \begin{cases} \dfrac{\partial\, EV_{i,t+1}(R_t - q_t)}{\partial R} & \text{if } q_t < R_t, \\ p & \text{if } q_t > R_t, \\ \dfrac{v_t \gamma_{im_t}^{1-\beta_i} \left(R_t/\alpha_{m_t}\right)^{-\beta_i} - \kappa_i}{\alpha_{m_t}} & \text{if } q_t = R_t, \end{cases} \tag{5}$$

where $EV_{i,t+1}$ is the expected value of the rest of the month, defined below. The three cases

describe where the day's requests leave the developer relative to the allowance, and each can occur on any day of the month.

1. If her requests fit within the allowance, she stays inside it and sends the requests of Equation (2) at $\lambda = \partial EV_{i,t+1}(R_t - q_t)/\partial R$. This is the usual case for a developer far from the allowance; on the last day of the month, when the continuation value is zero, her requests are those of Equation (2) at $\lambda = 0$.

2. If even her requests at the overage price exceed the remaining allowance, $\alpha_m c^*_{im}(v_t, p) > R_t$, as after a high demand shock or once the allowance is exhausted, she enters overage, the shadow price equals p, and she sends requests until marginal utility equals $\kappa_i + p\alpha_m$.

3. Between these cases, when $\alpha_m c^*_{im}(v_t, p) \le R_t \le \alpha_m c^*_{im}\big(v_t, \partial EV_{i,t+1}(0)/\partial R\big)$, she uses exactly the remaining allowance, $c_t = R_t/\alpha_m$, at the shadow price in the third line of Equation (5), which lies between $\partial EV_{i,t+1}(0)/\partial R$ and p.

Because the shadow price in the first two cases depends on the requests she sends, it can differ across models; the developer then compares the value of each model in Equation (6) below directly, and Equation (4) gives her choice exactly when the shadow price is the same for all models, as in overage.

The value of an active day with shock v and remaining allowance R is

$$V_{it}(v, R) = \max_{m \in \mathcal{M}_{r(t)},\, c \ge 0} \Big\{ u_{im}(c; v) - p \max\{\alpha_m c - R, 0\} + EV_{i,t+1}\big(\max\{R - \alpha_m c, 0\}\big) \Big\}, \quad (6)$$

and the expected value of entering day t with R points remaining is

$$EV_{it}(R) = \pi_i \, \mathbb{E}_v\big[V_{it}(v, R)\big] + (1 - \pi_i)\, EV_{i,t+1}(R), \qquad EV_{i,T+1}(R) = 0. \quad (7)$$

The terminal value is zero because unused points expire. The expectation is taken with respect to the daily demand shock, which also provides a motive for holding back some spending until later in the month in case a day arrives later on which the developer could disproportionately benefit from LLM usage. We do not discount within the month, since we model daily decisions over a finite and short horizon. For each type of developer and each regime, we solve Equations (6) and (7) by backward induction from the last day of the month. Appendix B.2 gives the general problem with an overage state, and Appendix B.7 shows that, with no cost of entering overage, the shadow price lies between zero and the overage price.

5.3 Behavioral Types

Our model allows for two departures from the forward-looking benchmark: myopia and overage aversion. Both are motivated by what happens when developers exhaust the allowance. A forward-looking developer's shadow price is already close to the overage price before she reaches the allowance, so her behavior changes continuously as she crosses it. In the data, by contrast, many developers stop right after they exhaust the allowance (Section 4.2). Since forward-looking behavior does not generate these patterns, we let developers differ in how they treat the allowance.

The first departure is myopia. We introduce a continuation weight $\rho_i \in \{0, 1\}$ that multiplies the change in the continuation value, $EV_{i,t+1}(R_{t+1}) - EV_{i,t+1}(R_t)$, when the developer makes her daily choice in Equation (6). Forward-looking developers have $\rho_i = 1$ and solve the problem of Section 5.2. Myopic developers ($\rho_i = 0$) treat the allowance as free until it runs out: they choose models and requests as if the shadow price were zero, and then pay the overage price.

The second departure is overage aversion. Overage-averse developers are myopic developers who never pay for overage. Once they exhaust the allowance, they stop using models that consume points. Overage aversion naturally reflects a feature of Copilot's tariff: paying for overage required the developer to set a positive overage budget, and the default budget was zero (Section 2.2). In the model, we implement overage aversion as a prohibitive cost F_i of entering overage, set large enough that no developer ever pays it, so overage-averse developers never pay for usage beyond the allowance; for the other types $F_i = 0$ (Appendix B.2).

Each developer's type is fixed across months. A developer is forward-looking with probability p_ρ; a myopic developer is overage-averse with probability p_F, and forward-looking developers are never overage-averse, so the model has three behavioral types with shares p_ρ, $(1-p_\rho)(1-p_F)$, and $(1-p_\rho)p_F$. Table OA-11 summarizes how each type behaves inside the allowance, at the allowance, and beyond it. These types differ only in decision utility, but we still evaluate welfare based on the same flow utility of Equation (1), and myopia and overage aversion change only the choices they make. We evaluate every type's welfare at the utility she actually receives from those choices, net of overage payments, which includes the cost of running out of the allowance early.

5.4 Model Quality and Developer Heterogeneity

We now specify how the quality of each model varies across developers. Following Berry et al. (1995), we decompose log quality into a component common to all developers and

developer-specific deviations from it:

$$\ln \tilde{\gamma}_{im} = \delta_m + b_i^A A_m + b_i^S \, \text{SWE}_m + \varepsilon_{im}, \qquad \gamma_{im} = \frac{\exp(\ln \tilde{\gamma}_{im})}{\sum_{k \in \mathcal{U}} \exp(\ln \tilde{\gamma}_{ik})}, \tag{8}$$

where $\mathcal{U}$ is a fixed universe of 48 models, including models never on the Pro menu, so that a model's entry or exit does not change the quality of the others (Appendix B.6), δ_m is a model fixed effect, A_m indicates models made by Anthropic, SWE_m is the model's score on SWE-bench Verified, the share of tasks it solves, between 0 and 1, and ε_{im} is a persistent match value described below. The random coefficients b_i^A and b_i^S measure developer i's tastes for Anthropic models and for benchmark performance. A developer with a strong taste for Anthropic values all Anthropic models more, and if people disagree in their tastes for this dummy, that makes Anthropic models closer substitutes for each other than for models of other providers; a developer with a strong taste for SWE-bench performance values the frontier models that score highest on it.

The fixed effects have a simple interpretation. The denominator in Equation (8) is common to all models for a given developer, so by Equation (4) a developer with zero tastes and equal match values is indifferent between models m and k when

$$\delta_m - \delta_k = \ln(\kappa_i + \lambda \alpha_m) - \ln(\kappa_i + \lambda \alpha_k). \tag{9}$$

The gap between two fixed effects is therefore the log ratio of full marginal costs that leaves an average developer indifferent between the two models: a fixed effect higher by 0.1 means the developer would pay about 10% more per request, in time and money, to use the model. Fixed effects are log price equivalents, and they measure vertical differentiation. The tastes and match values measure horizontal differentiation, at the level of the provider and of the individual model. The question of how differentiated LLMs are thus becomes a question about the dispersion of fixed effects, tastes, and match values, which we estimate.

The match value of developer i for model m is $\varepsilon_{im} = s_i \, G_{im}$, where G_{im} is drawn independently from a standard Gumbel distribution and its scale $s_i = \bar{s} \exp(g \, z_i^{\mu})$ increases with the developer's demand level, with z_i^{μ} the standard normal draw behind her mean log demand μ_i. Match values indicate that a model may suit a particular developer's choice of programming language or work style better than its rivals. They are drawn once and fixed across months, so like the other taste components, they generate persistence in model choice. Indeed, the model generates no between-day or between-month switching between LLMs except through variation in the shadow price or menu, as Equation (4) implies.

5.5 Discussion

One model per day. In the model, a developer does not use several models on the same day. This rules out, for example, starting a task with a cheap model and moving to a frontier model when the cheap one fails. We make this assumption for two reasons. First, it matches how developers use Copilot: 63% of active developer-days with at least two requests use a single model. Second, it keeps the daily problem tractable—a discrete choice of model, followed by a continuous choice of the number of requests—which lets us solve the dynamic problem for many developers and menus. A more general model would let each request draw its own task and let the developer choose a model task by task, as in models of multiple-discrete choice and of demand with complementarities (Hendel, 1999; Dubé, 2004; Gentzkow, 2007; Thomassen et al., 2017; Iaria and Wang, 2020; Ershov et al., 2025), but this requires observing the type of each task, which our data do not record.

Token-based billing. We write the model for the request-based tariff, but the same structure applies after the June 2026 billing change, with the allowance measured in dollars rather than in points. From June 1, 2026, the allowance is $\bar{C}$ = \$15 of usage at list prices, the price of a request to model m, α_m, is its list price in dollars, and the overage price is p = \$1 per dollar of list-price usage, so $p\alpha_m$ is again the money price of a request beyond the allowance. To keep the model the same throughout our sample, from June 1 we convert token-based prices into prices per request: α_m is the average cost of a request to model m at list prices (Appendix A).[12] This is the expected price of a request to that model, which is what a developer weighs when she chooses it.[13] The other change is that no model is free after June 1, so overage-averse developers who exhaust the allowance stop using Copilot's models until it resets rather than falling back to included models.

Plan choice. We do not model the choice of plan or the decision to stop using Copilot. Our sample consists of developers who hold a Pro subscription throughout the sample period (Section 3.2), and the model describes their choices conditional on the plan. This has two implications for the counterfactuals. First, because the fee is a lump sum, it does not affect a subscriber's daily choices, so the model has nothing to say about how the fee level affects demand; a change in the fee would matter only through the unmodeled decision to subscribe. Second, the counterfactuals that change the set of offered models in Section 8 hold plan membership fixed, so they omit developers who would move to another plan or cancel their subscription. We return to the direction of the resulting bias

[12] Tokens include the backend work that a request triggers, and prices are net of documented promotions.

[13] The implicit assumption is that (i) other than through choosing the model, developers have little control over how many tokens a request consumes, and (ii) the monetary amounts involved are sufficiently small that we do not need to model risk aversion over the lottery of prices that a request technically implies.

when we discuss each counterfactual.

6 Estimation and Identification

We estimate the parameters of the model by simulated method of moments. For a candidate set of parameters, we simulate a large population of developers, solve each developer's dynamic problem under the menus and prices of our sample, simulate her daily choices over the ten months, and compute the moments from the simulated choices exactly as we compute them from the data. We choose the parameters to make the simulated moments as close as possible to the data. In this section we first specify the distributions of developer characteristics (Section 6.1), then describe the estimator, the targeted moments and the standard errors (Section 6.2), and finally explain which variation in the data identifies each group of parameters (Section 6.3). Complete details are in Appendix C.

6.1 Parameterization

We parameterize the distributions of developer characteristics as follows. The probability of being active, π_i, has a beta distribution; mean log demand, μ_i, is normal; the dispersion of daily shocks, σ_i, and the time cost, κ_i, are log-normal; and curvature, β_i, follows a beta distribution on a subinterval of $(0, 1)$, although below we remove the dispersion of the time cost and of curvature to aid in identification. The daily shock z_t is standard normal, truncated to exclude the top 0.5% of its distribution.[14] The tastes b_i^A and b_i^S are normal with mean zero, and the match values are as in Section 5.4. The vector $(\pi_i, \mu_i, \sigma_i, \kappa_i, \beta_i, \rho_i, F_i, b_i^A, b_i^S, \{\varepsilon_{im}\}_{m \in \mathcal{U}})$ describes a developer and is fixed across months; only the activity draws and the demand shocks change from day to day. All developer characteristics are drawn independently, except that the scale of match values rises with the demand level (Section 5.4).

We estimate 52 parameters: 14 that govern the distributions of developer characteristics and the type shares, and 38 model fixed effects. We fix the others (Table OA-8). We normalize quality by a softmax over a fixed universe $\mathcal{U}$ of 48 models, including models never on the Pro menu, so that a model's entry or exit does not change the quality of the others (Appendix B.6), and we set the fixed effect of GPT-4o to zero. The means of the tastes are zero because the fixed effects absorb them.

In practice, we set the dispersion of the time cost to zero and fix the concentration of the curvature distribution at a large value (Table OA-8), so both are, in effect, common to all developers. We make this choice because the data separate these dispersions only weakly. When the shadow price is near zero, a developer's requests depend on her demand

[14] As in Nevo et al. (2016), this restriction helps with the numerical stability of our solution routine.

level and time cost only through their ratio, and the day-to-day variation in her requests reflects her daily shocks scaled by curvature (Equation (2)). In the moments that describe usage, dispersion in the time cost therefore trades off against dispersion in demand, and dispersion in curvature against dispersion in the shocks. The parameters differ only through nonlinearities: the time cost adds to the money price in the full marginal cost, so developers with the same usage respond differently to a positive price depending on their time cost, and curvature scales both the level and the spread of log requests. These differences identify the dispersions in a large enough sample, but in a sample of our size the separation is noisy, especially between demand and the time cost. We therefore attribute this dispersion to demand and the shocks. The common time cost and curvature are identified mainly by developers who pay for overage (Section 6.3), so applying them to all developers is an assumption, which matters most for the counterfactuals that charge developers for every request.

6.2 Estimation

We choose the 52 free parameters, θ, to satisfy

$$\hat{\theta} = \arg\min_{\theta} \sum_{k=1}^{53} \left(\frac{g_k(\theta) - \hat{g}_k}{w_k} \right)^2, \tag{10}$$

where $\hat{g}_k$ is moment k in the data, $g_k(\theta)$ is its simulated counterpart, and w_k scales the deviation.

In choosing which moments to match, we focus on the variation that identifies each group of parameters. We target 53 moments, listed in Table 3: 14 moments that describe how developers use Copilot, in three groups, and the request shares of the 39 models on the menu. The first group matches activity and the volume of usage: the mean and cross-developer standard deviation of developers' mean log daily requests, the mean and standard deviation of the within-developer standard deviation of log daily requests, and the mean and standard deviation of the share of inactive days. The second group matches responses to prices and behavior at the allowance: the within-developer gap in log requests between overage days and days early in the allowance, the change in log requests on overage days from May to June 2026, and the ratios of spending and of the probability of being active on the day after to the day before a developer exhausts the allowance. The third group matches persistence and launches: the share of developers who keep their most-used model from May to June 2026, overall and among developers in overage in both months, the adoption of Anthropic launches by users of other providers' models relative to all developers, and the distance in SWE-bench score between adopters'

previous model and the new model relative to all developers.

The 53 moments identify the 52 free parameters: 14 behavioral parameters and 38 model fixed effects. Table OA-8 lists the parameters we fix and their values.

For the 39 model shares, $w_k = 1$, so deviations are in percentage points; for the other 14 moments, $w_k = |\hat{g}_k|$, so deviations are relative to the data value. We do not use the efficient weighting matrix. The moments on activity and model shares are measured far more precisely than the price contrasts and the moments at the allowance, so efficient weights would load almost entirely on the former and leave the parameters that govern price responses to be determined by functional form. We compute standard errors of the data moments of the estimation sample (Section 3.2) by bootstrapping over developers.[15]

To compute the simulated moments for a given set of parameters, we draw the characteristics of 100,000 developers from the distributions given in Section 6.1, solve each developer's dynamic problem in each month of our sample, and simulate her daily activity, model choices and requests over the ten months. We solve the dynamic problem by backward induction, representing the value function on a grid of 100 points for the remaining allowance and integrating over the daily demand shock by Gauss–Legendre quadrature (Appendix B.5). We then compute the moments from the simulated developer-days exactly as we compute them from the data. The draws of developer characteristics and daily shocks are held fixed across evaluations, so the objective changes only with the parameters.[16] We minimize the objective with a derivative-free trust-region method for least squares problems (Cartis et al., 2019).[17]

We calculate standard errors with the sandwich formula for a method-of-moments estimator with non-optimal weights,

$$\widehat{V}(\hat{\theta}) = (D'D)^{-1} D' \left(\Omega_{\text{data}} + \Omega_{\text{sim}}\right) D \left(D'D\right)^{-1}, \tag{11}$$

where D is the Jacobian of the scaled moments with respect to the parameters, Ω_{data} is the covariance of the scaled data moments, and Ω_{sim} is the covariance of the scaled simulated

[15] The behavioral moments average the seven months before the June 2026 billing change with equal weights; the months after the change enter through the May-to-June contrasts and the model shares. Because the data panel keeps only developers who use Copilot in every month, a simulated developer-type path with no requests in some month receives zero weight (Appendix C.1). As in Nevo et al. (2016), we sample developers with replacement, keeping all of a developer's days in each draw, and recompute the moments in each of 200 bootstrap samples.

[16] Rather than drawing one type per developer, we compute the moments on paths weighted by the type probabilities, so that the moments move smoothly with the type shares.

[17] The moments at the allowance are computed on the small subsample of developer-months that exhaust it, and they jump when a single simulated developer changes behavior, so the objective is deterministic but rough.

Table 3: Targeted Moments and the Parameters They Identify

Moment	Parameters it mainly identifies	Parameter
Panel A. Activity and Usage Volume		
Mean of developers' mean log daily requests	Mean demand level	m_μ
SD of developers' mean log daily requests	Dispersion of demand level	v_μ
Mean of within-developer SD of log daily requests	Mean volatility of daily demand	m_σ
SD of within-developer SD of log daily requests	Dispersion of volatility	v_σ
Mean share of inactive days	Mean activity probability	m_π
SD of share of inactive days	Dispersion of activity probability	c_π
Panel B. Price Response and Behavioral Types		
Log requests, overage days minus days early in the allowance	Time cost, curvature	m_κ, m_B
Change in log requests on overage days, May to June 2026	Time cost, curvature	m_κ, m_B
Spend, day after reaching the allowance relative to day before	Shares of myopic and overage-averse types	p_ρ, p_F
Probability active, day after reaching the allowance relative to day before	Share of overage-averse types	p_F
Panel C. Persistence and Launches		
Share keeping most-used model, May to June 2026	Scale of match values	$\bar{s}$, g
Same, among developers in overage in both months	Scale of match values	$\bar{s}$, g
Adoption of Anthropic launches by other providers' users, relative to all	Dispersion of Anthropic taste	v_A
SWE-bench proximity of adopters of new models, relative to all	Dispersion of SWE-bench taste	v_S
Panel D. Model Shares (39 Moments)		
Share of requests to each model on the menu	Model quality	δ_m

Notes: The table lists the moments targeted in estimation, grouped as in Section 6.2, and the parameters each moment mainly identifies (Section 6.3); all parameters are estimated jointly. Behavioral moments average the months before the June 2026 billing change with equal weights, except the May-to-June contrasts. Model shares are computed for each menu regime. Table OA-12 reports, for each behavioral moment, the deviation at the estimates from its value in the data, and summarizes the deviations of the model shares.

moments. We estimate Ω_{data} from the 200 bootstrap samples of developers described above and Ω_{sim} from 100 bootstrap samples of simulated developers at the estimates, so the standard errors account for both sampling error in the data and simulation error. Appendix C.5 describes how we compute the Jacobian.

6.3 Identification

The parameters fall into five groups, each identified mainly by one block of moments in Table 3: the distributions of activity and demand shocks, from usage within the month; the time cost and curvature, from price contrasts at different wedges between time cost and money; the shares of behavioral types, from behavior at the allowance; the scale of match values and the variances of tastes, from persistence in model choice and from launches; and the model fixed effects, from model shares.

The first group is straightforward, as the moments on activity and usage volume in Panel A map closely to the distributions of activity and demand shocks. The share of inactive days and its dispersion across developers pin down the distribution of the activity probability, the within-developer spread of log daily requests pins down the scale of daily shocks given curvature, and the cross-developer mean and dispersion of mean log daily requests pin down the mean and dispersion of mean log demand.

The first two moments in Panel B identify the time cost and curvature. If the full marginal cost of a request were known, curvature could be inferred from how requests change when that cost changes. However, the full marginal cost includes the time cost, which we do not observe. When the allowance does not bind, the shadow price is near zero and Equation (2) shows that the level of usage identifies only the demand level net of time cost, $(\mu_i - \ln \kappa_i)/\beta_i$, plus a term in quality. Consider two sets of parameters that give the same usage on such days, one with a high time cost and high demand and one with a low time cost and low demand. Under the first, developers barely respond to money prices, because money is a small part of the full marginal cost, while under the second they respond strongly. We separate them using two price contrasts at different wedges. The first is the within-developer gap in log requests between days on which a developer pays for overage and days on which she has used less than half of the allowance:

$$\Delta_1 \approx -\frac{1}{\beta} \ln\left(\frac{\kappa + p\bar{\alpha}}{\kappa}\right), \tag{12}$$

where $\bar{\alpha}$ is the points per request of the models she uses on overage days, κ and β are common across developers (Section 6.1), and the approximation ignores switching between models across the two kinds of days. The second is the within-developer change in log

requests on days in overage from May to June 2026, when the overage price of a request rose from \$0.04 per point to the model's list price:

$$\Delta_2 \approx -\frac{1}{\beta} \ln\left(\frac{\kappa + p'\bar{\alpha}'}{\kappa + p\bar{\alpha}}\right). \tag{13}$$

The ratio Δ_1/Δ_2 does not depend on β and is monotone in the ratio of the money price to the time cost, so it pins down κ, after which either contrast gives β.

Our identifying assumption for the time cost and curvature is that the elasticity of requests with respect to the full marginal cost, time cost plus money, is constant. The assumption is needed because, without it, an additive component of the time cost could not be separated from curvature. The gap between overage days and days early in the allowance partly reflects mean reversion, since developers reach overage after days of unusually high demand; the simulated moment reproduces this selection, but it makes the moment noisy. In addition, only 11.8% of developer-months pay for overage (Section 3.3), so both contrasts rest on a small group of developers.

The last two moments in Panel B identify the shares of behavioral types, which we expect to be determined primarily by behavior at the allowance. On the day before and the day after a developer exhausts the allowance, the three types behave differently. A forward-looking developer already faces a shadow price close to the overage price before she reaches the allowance, so her spending changes little when she crosses it. A myopic developer treats the allowance as free until she exhausts it and then faces the overage price, so she keeps coding but spends less. An overage-averse developer stops using models that consume the allowance. The ratio of spending on the day after to the day before the crossing therefore informs the share of forward-looking developers, and the ratio of the probability of being active informs the share of overage-averse developers among the myopic ones.

The first two moments in Panel C identify the match values, through how many developers keep their most-used model when the June 2026 billing change shifts relative prices. The change raised the price of some models much more than others, so it re-ranks models for a developer who cares only about their common quality and price. With no dispersion in match values, all developers would re-rank models in the same way, and many would switch; as the dispersion increases (holding all other parameters fixed), the predicted share who keep their model rises, because their match with it outweighs the change in price. We therefore target the share of developers whose most-used model in May 2026 remains their most-used model in June 2026, among developers whose May model is still on the menu. The same share among developers in overage in both months pins down how the

dispersion of match values increases with the demand level, since these developers have high demand levels and face the full change in price.

The last two moments in Panel C identify the variances of the Anthropic and SWE-bench tastes, through the developers whom a launch draws. The models from which a new model draws its users identify the dispersion of tastes, much like the second-choice data in Berry et al. (2004): the model a developer used before adopting a new one shows which models she treats as close substitutes (see also Berry and Haile, 2021, Section 6.2). Adoption would differ between previous users of Anthropic models and of other providers' models even without dispersion in the Anthropic taste, because developers also sort on SWE-bench performance and price tiers; as the dispersion increases, adoption becomes more concentrated among previous Anthropic users. We target the adoption rate of an entering Anthropic model among developers whose previous most-used model is not an Anthropic model, relative to the overall adoption rate, pooled over Anthropic launches isolated from other menu changes (Appendix C.2 defines adoption and the previous model). For SWE-bench performance, we target the distance in SWE-bench score between adopters' previous model and the new model, relative to the average distance across all developers.

Finally, the model shares in Panel D identify the model fixed effects, each pinned down by that model's request share on the days it is on the menu.

7 Results

We now present the results of the model. We begin with the parameter estimates and then show how well the model fits the moments used in estimation. We then use the estimates to derive price elasticities and diversion ratios between models and the consumer surplus from Copilot. Finally, we relate the estimated model fixed effects to public benchmarks, to measure how much developers are willing to pay for model capability as measured by benchmarks.

7.1 Parameter Estimates

Table 4 reports the estimates of the parameters other than the model fixed effects, grouped into request volume, activity, costs and curvature, behavioral types, and tastes and match values. For the fixed effects, the table reports their standard deviation within and across providers, and Table OA-13 in the appendix reports each estimate.

Starting with the parameters that govern usage (Panels A and B), the median developer is active on 9.0 days a month and sends 5.6 requests on an active day. Usage differs widely across developers: the standard deviation of active days per month is 7.1, and mean daily requests are 5.7 times higher at the 90th percentile than at the 10th. Demand also varies

Table 4: Parameter Estimates

Parameter	Symbol	Estimate	Implied quantity: Quantity	Implied quantity: Value
Panel A. Request Volume				
Mean of log demand level, μ_i	m_μ	0.368 (0.193)	Median requests per active day	5.6
Variance of log demand level	v_μ	0.168 (0.028)	90/10 ratio of mean daily requests	5.7
Location of log SD of daily shocks, σ_i	m_σ	-0.525 (0.083)	Median σ_i	0.59
Variance of log SD of daily shocks	v_σ	0.062 (0.003)	90/10 ratio of σ_i	1.88
Panel B. Activity				
Mean daily activity probability, π_i	m_π	0.252 (0.006)	Median active days per month	9.0
Concentration of activity probability	c_π	2.18 (0.046)	SD of active days per month	7.1
Panel C. Costs and Curvature				
Location of log time cost, κ_i	m_κ	-1.789 (0.147)	Time cost, κ (\$ per request)	0.167
Mean of curvature draw, B_i	m_B	0.354 (0.117)	Full-cost elasticity, $-1/\beta_i$	-1.52
Panel D. Behavioral Types				
Probability forward-looking	p_ρ	0.692 (0.065)	Forward-looking (%)	69.2
Probability overage-averse if myopic	p_F	0.621 (0.204)	Overage-averse (%)	19.2
Panel E. Model Quality				
SD of quality δ_m within providers		0.337	–	–
SD of provider means of quality δ_m		0.191	–	–
Panel F. Tastes and Match Values				
Variance of Anthropic taste, b_i^A	v_A	0.008 (0.006)	–	–
Variance of SWE-bench taste, b_i^S	v_S	0.228 (0.080)	–	–
Scale of match values	$\bar{s}$	0.033 (0.005)	–	–
Loading of match scale on demand level	g	1.503 (0.196)	–	–

Notes: This table reports estimates of the parameters other than the model fixed effects; Panel E summarizes the fixed effects, which are reported in Table OA-13. Standard errors in parentheses combine sampling and simulation error (Appendix C.5). The last two columns translate each estimate into an economic quantity for the simulated estimation sample; at an hourly wage of \$75, the time cost corresponds to about 8 seconds per request. Parameters held fixed in estimation are listed in Table OA-8.

from day to day for a given developer. The median standard deviation of the log daily demand shock is 0.59, so on a day one standard deviation above her mean a developer's demand is about 80% higher, and this volatility itself differs across developers, with a 90/10 ratio of 1.88.

For costs and curvature (Panel C), the time cost of a request is \$0.167. At a wage of \$75 an hour, it corresponds to about 8 seconds of a developer's time per request. This is plausible because many requests are short follow-ups within a chat or agent session, and developers can keep working while longer requests run. The time cost is an important part of the cost of a request: for example, it is a little more than half the overage price of a Claude Sonnet 4.6 request, and inside the allowance, where the shadow price is low, it is most of the cost. The curvature estimate implies an elasticity of requests with respect to the full marginal cost of $-1/\beta = -1.52$, so a 10% increase in the full marginal cost lowers requests by 13.5%.[18]

Among the behavioral types (Panel D), 69.2% of developers are forward-looking. Of the myopic developers, 62.1% are overage-averse: once they reach the allowance they stop using premium models rather than pay for overage. Overage-averse developers are therefore 19.2% of all developers, and myopic developers who pay for overage 11.7%. This is consistent with the inattention that Grubb and Osborne (2015) find among cellular phone users, who do not track their past usage. Because myopic developers treat requests inside the allowance as free and respond to prices only once they pay for overage, aggregate usage is less price elastic than it would be if all developers were forward-looking.

The model fixed effects capture the vertical side of differentiation. Panel E reports the standard deviation of quality within providers and across the provider means, and Table OA-13 reports the quality of each model, expressed in dollars per request.[19] Quality varies much more within providers than across them, because each provider sells both frontier and routine models: the standard deviation within providers is 0.337, 1.8 times that of the provider means. The relevant unit of differentiation is therefore the model, not the provider. The estimates are consistent with the reputation of these models among developers: Anthropic's Sonnet and Opus models and OpenAI's newest frontier models have the highest quality. A developer with average match values would pay \$0.09 more per request for GPT-5.4 than for GPT-5.4 mini and \$0.07 more for Claude Sonnet 4.6 than

[18] The full marginal cost of a request is the time cost κ plus the money price of the request, which inside the allowance is its shadow price and after the allowance is the overage price (Section 5.1). Because utility is isoelastic in requests, the elasticity of requests with respect to the full marginal cost is $-1/\beta$ for every developer and model.

[19] Within a day, a developer ranks models by quality relative to the full marginal cost of a request (Section 5.1), so a difference in quality is equivalent to a proportional difference in full marginal cost and can be expressed in dollars at a given price.

for Claude Haiku 4.5, 55% and 30% of the respective price differences. We return to the relation between the estimated qualities and public benchmarks in Section 7.5.

Turning to tastes and match values (Panel F), these parameters capture how much developers disagree about models around the qualities in Panel E, the horizontal side of differentiation. Developers largely agree on the value of the provider: the standard deviation of the Anthropic taste is 0.09, so for a developer one standard deviation above the mean, Anthropic models are worth about 9% more, equivalent to a 9% lower full cost of a request. They disagree more on the value of coding ability: the standard deviation of the SWE-bench taste is 0.48, so for a developer one standard deviation above the mean, a model that scores 0.1 higher on SWE-bench is worth about 5% more than it is to the average developer, and about 22% more for the difference between Claude Sonnet 4.6 and GPT-4o.

Developers also differ in their fit with individual models. At the median demand level, the standard deviation of match values is 0.04, so a model that fits a developer one standard deviation better is worth about 4% more to her. Because the dispersion of match values rises with the demand level ($g = 1.50$), the same comparison at the 90th percentile of demand gives about 29%, almost as large as the standard deviation of quality within providers (0.337), so heavy users disagree most about which model is best.

7.2 Model Fit

Table 5 compares the moments simulated from the model at the estimated parameters with their values in the data. Because the levels of the data moments are confidential, we report for each moment only the deviation, the simulated moment minus the data moment, in percentage points for shares and in percent otherwise.

The model fits the targeted moments well. The moments that describe activity and usage volume deviate from the data by at most 0.7% or 0.11 percentage points. The model reproduces the response of requests to the overage price and to the June 2026 billing change almost exactly, with deviations of at most 0.1%, and the change in spending at the allowance within 2.1%. The share of developers who keep their most-used model through the billing change is within 1.5 percentage points of the data, and the 39 model shares deviate by 0.26 percentage points on average and by at most 0.60 points.

7.3 Price Elasticities and Substitution Between Models

In this section we use the estimates to study substitution patterns between models. We report two measures: price elasticities, which show how strongly requests respond to prices, and diversion ratios, which show where the requests a model loses after a price

Table 5: Model Fit

	Deviation
Activity and usage volume	
Mean of developers' mean log daily requests (%)	-0.72
SD of developers' mean log daily requests (%)	-0.4
Mean of within-developer SD of log daily requests (%)	-0.7
SD of within-developer SD of log daily requests (%)	-0.3
Mean share of inactive days (pp)	0.11
SD of share of inactive days (%)	-0.4
Price response and behavioral types	
Log requests, overage minus early-allowance days (%)	0.04
Change in log requests on overage days, May–June (%)	0.10
Spend, day after vs. day before reaching the allowance (%)	2.1
Activity, day after vs. day before reaching the allowance (%)	-0.1
Persistence and launches	
Share keeping most-used model, May–June (pp)	0.21
Same, among developers in overage in both months (pp)	-1.45
Anthropic launch adoption by other providers' users, relative (%)	-0.6
SWE-bench proximity of new-model adopters, relative (%)	1.2
Model shares (39 moments)	
Mean abs. deviation of model shares (pp)	0.26
Largest abs. deviation of any model share (pp)	0.60

Notes: We report deviations rather than levels, which are confidential. Deviation is the simulated moment at the estimated parameters minus the data moment, in the unit given in each row: percentage points (pp) for shares and percent (%) otherwise. For moments measured in logs or log differences, the percent deviation is 100 times the difference in logs. The table reports the moments targeted in estimation (Table 3); for the 39 model shares it reports the mean and largest absolute deviation. Table OA-12 reports the full set of targeted moments.

increase go.

Table 6 reports price elasticities (Panel A) and diversion ratios (Panel B) for the Anthropic and OpenAI models. We calculate both by simulation. For each model, we raise its price and simulate every developer's choices from June to August 2026, the three months after the billing change, under the token-based prices, holding the prices of all other models fixed. We then compare requests with and without the increase, averaging over developer-days. The elasticities are arc elasticities, and the diversion ratios are the change in requests to each other model divided by the fall in requests to the model whose price rises. Within each provider, models are ordered by their SWE-bench Verified score.

Panel A reports own-price and cross-price elasticities. The own-price elasticities of the premium models average -1.5. These elasticities are small relative to those typically estimated for differentiated products, for two reasons. First, the time cost is a large part of the full cost of a request, so a given change in the money price is a smaller change in the full cost. Second, price matters less to developers far from the allowance: developers who never reach it respond about half as much as those who do. The cheaper models have own-price elasticities that are smaller in absolute value, between -0.01 and -0.70, because

their prices have an even smaller effect on the full cost of a request. Cross-price elasticities are small, mostly below 0.1, because each model's lost requests are spread over several substitutes and the price is only part of the full cost of a request.

Panel B reports diversion ratios. Diversion to the Google, xAI, Moonshot and Microsoft models is grouped as other models, and the last three columns summarize each row: the share of lost requests that go to other models of the same provider, the share that go to models of other providers, and the share that are not sent at all. Some diversion ratios are negative because of an implicit income effect: all models draw on one allowance, so an increase in the price of one model raises the shadow price of every other model and can lower requests to them as well.

The results suggest that developers substitute mostly toward models with similar capabilities and prices. For example, 45% of the requests Claude Sonnet 4.6 loses go to Sonnet 4.5, 40% of those Sonnet 5 loses go to Sonnet 4.6, and 23% of those GPT-5.4 loses go to GPT-5.3-Codex. Among the cheapest models, GPT-5 mini sends 37% of its lost requests to GPT-4.1, and GPT-4.1 sends 35% to GPT-4o mini and 29% to GPT-5 mini.

Substitution is concentrated within providers. For the frontier models of both providers, between 57 and 71% of lost requests go to other models of the same provider. The pattern is weaker at the low end of Anthropic's line-up: Claude Haiku 4.5 sends only 8% of its lost requests to other Anthropic models and 29% to GPT-5 mini. At the frontier, providers also compete directly: 24% of the requests GPT-5.4 loses go to Anthropic models, and 33% of those Sonnet 4.6 loses go to other providers. The share of lost requests that are not sent at all rises as models become cheaper and less capable. It is between 8 and 13% for the frontier models, but between 43 and 87% for the cheapest models, such as GPT-4.1 and GPT-4o. After a price increase, developers of frontier models therefore keep coding with a slightly less capable model, while developers of the cheapest models mostly send fewer requests.

7.4 Consumer Surplus

The estimates allow us to measure how much developers gain from Copilot. In this section we quantify the consumer surplus from the subscription, describe how it is distributed across developers, and relate it to how intensively developers use AI.

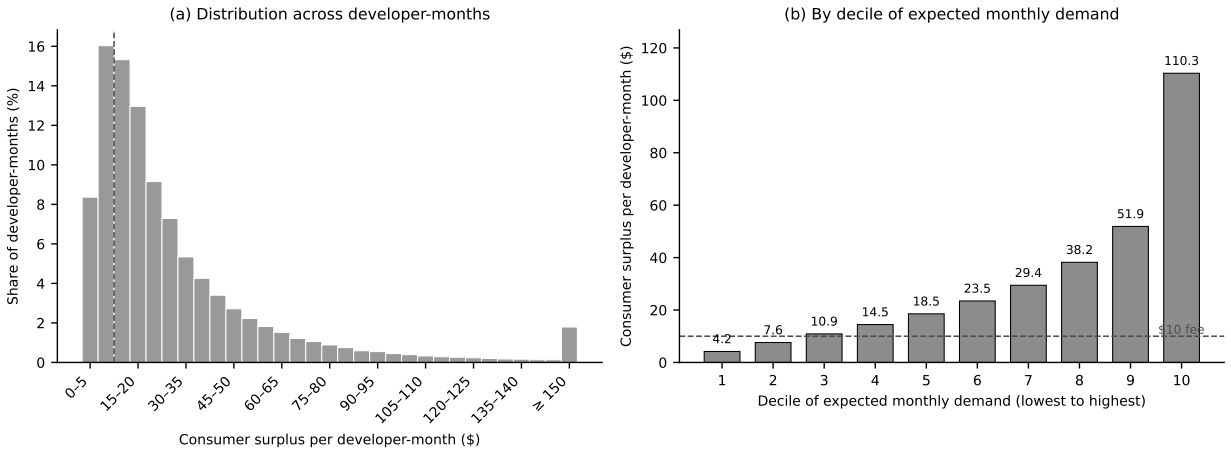
We measure consumer surplus as the realized monthly payoff from using Copilot, and we compute it for June to August 2026. For each simulated developer, we simulate her choices in each of these months at the menus and prices in place on each day. On each active day, her surplus is the value of the requests she sends to the model she chooses, given her demand that day, minus their time cost; on inactive days it is zero. We add these daily

Table 6: Price Elasticities and Diversion Ratios Between Models

	Anthropic				OpenAI										Summary		
Price raised ↓ / to →	Sonnet 4.6	Sonnet 5	Sonnet 4.5	Haiku 4.5	GPT-5.6 Terra	GPT-5.6 Luna	GPT-5.4	GPT-5.3-Codex	GPT-5.4 mini	GPT-5 mini	GPT-4.1	GPT-4o	GPT-4o mini	Other models	Same provider	Other providers	Outside
Panel A. Price Elasticities of Requests																	
Claude Sonnet 4.6	−1.37	0.43	1.06	0.13	0.28	0.05	0.29	0.22	0.05	0.04	0.05	0.03	0.02	–	–	–	–
Claude Sonnet 5	0.19	−1.38	0.12	0.03	0.07	0.02	0.06	0.05	0.03	0.02	0.03	0.01	0.02	–	–	–	–
Claude Sonnet 4.5	0.32	0.07	−1.99	0.29	0.10	0.10	0.19	0.20	0.05	0.20	0.14	0.04	0.03	–	–	–	–
Claude Haiku 4.5	0.00	0.00	0.04	−1.11	0.01	0.04	0.01	0.02	0.03	0.08	0.04	0.01	0.01	–	–	–	–
GPT-5.6 Terra	0.02	0.02	0.03	0.04	−1.41	0.11	0.12	0.08	0.01	0.02	0.05	0.01	0.01	–	–	–	–
GPT-5.6 Luna	0.00	−0.01	0.01	0.01	0.00	−0.48	0.00	0.01	0.01	0.03	0.04	0.01	0.01	–	–	–	–
GPT-5.4	0.06	0.03	0.14	0.08	0.21	0.06	−1.42	0.42	0.07	0.11	0.07	0.03	0.02	–	–	–	–
GPT-5.3-Codex	0.03	0.01	0.10	0.08	0.09	0.06	0.21	−1.53	0.07	0.14	0.06	0.02	0.02	–	–	–	–
GPT-5.4 mini	0.00	0.00	0.01	0.01	0.01	0.01	0.00	0.00	−0.70	0.02	0.01	0.00	0.00	–	–	–	–
GPT-5 mini	−0.02	−0.01	−0.01	0.03	0.00	0.04	−0.01	−0.01	0.02	−0.14	0.06	0.02	0.01	–	–	–	–
GPT-4.1	−0.01	−0.01	0.00	0.01	−0.01	0.02	−0.01	0.00	0.01	0.03	−0.10	0.00	0.03	–	–	–	–
GPT-4o	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.23	0.00	–	–	–	–
GPT-4o mini	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.01	–	–	–	–
Panel B. Diversion Ratios																	
Claude Sonnet 4.6	–	0.11	0.45	0.03	0.04	0.01	0.11	0.06	0.01	0.03	0.03	0.00	0.02	0.01	0.58	0.33	0.09
Claude Sonnet 5	0.40	–	0.15	0.02	0.03	0.01	0.07	0.04	0.01	0.05	0.06	0.00	0.06	0.02	0.57	0.35	0.08
Claude Sonnet 4.5	0.28	0.02	–	0.07	0.02	0.02	0.08	0.07	0.01	0.19	0.12	0.01	0.03	0.01	0.37	0.54	0.09
Claude Haiku 4.5	0.01	0.00	0.07	–	0.01	0.03	0.02	0.03	0.02	0.29	0.12	0.00	0.03	0.02	0.08	0.57	0.34
GPT-5.6 Terra	0.08	0.02	0.06	0.03	–	0.10	0.20	0.10	0.01	0.08	0.15	0.01	0.06	0.02	0.71	0.20	0.09
GPT-5.6 Luna	−0.03	−0.01	0.02	0.02	0.01	–	0.00	0.02	0.01	0.23	0.28	0.01	0.06	−0.01	0.60	0.00	0.40
GPT-5.4	0.09	0.01	0.11	0.03	0.06	0.02	–	0.23	0.02	0.18	0.10	0.01	0.04	0.01	0.65	0.25	0.10
GPT-5.3-Codex	0.04	0.01	0.10	0.04	0.03	0.03	0.18	–	0.02	0.27	0.10	0.01	0.04	0.01	0.67	0.20	0.13
GPT-5.4 mini	0.00	0.00	0.03	0.03	0.01	0.01	0.00	0.01	–	0.21	0.04	0.00	0.03	0.06	0.32	0.11	0.57
GPT-5 mini	−0.12	−0.03	−0.03	0.05	0.00	0.06	−0.04	−0.02	0.02	–	0.37	0.01	0.06	0.00	0.45	−0.12	0.67
GPT-4.1	−0.08	−0.03	−0.02	0.01	−0.01	0.07	−0.03	0.00	0.01	0.29	–	0.00	0.35	0.00	0.67	−0.11	0.43
GPT-4o	−0.02	−0.01	0.00	0.00	−0.01	0.01	0.01	−0.01	0.01	0.07	0.06	–	0.02	0.00	0.16	−0.03	0.87
GPT-4o mini	−0.04	−0.01	−0.08	0.05	−0.01	0.05	−0.02	−0.03	0.00	0.05	0.37	0.02	–	−0.02	0.44	−0.09	0.65

Notes: Each row raises the price of one model, simulated for every developer from June to August 2026. Within each provider, models are ordered by SWE-bench Verified score, highest first. Panel A reports the arc elasticity of the column model's requests with respect to the row model's price; own-price elasticities are on the diagonal. Panel B reports the share of the row model's lost requests gained by the column model. "Other models" sums the Google, xAI, Moonshot and Microsoft models. "Same provider" sums diversion to the other models of the row model's provider, "Other providers" to all models of other providers, including "Other models", and "Outside" is the share of lost requests that are not sent; these three sum to one. Negative entries arise because all models draw on one allowance.

Figure 7: Consumer Surplus



Notes: Panel (a) plots the distribution of consumer surplus across simulated developer-months from June to August 2026, at the menus and prices in place in each month; the dashed line marks the $10 monthly fee. Panel (b) plots mean consumer surplus per developer-month by decile of the developer's expected monthly demand, $\pi_i \exp(\mu_i/\beta)$, which combines how often she is active and how much she uses Copilot on an active day; the dashed line marks the $10 monthly fee. Consumer surplus is the realized monthly payoff from using Copilot, in dollars, net of the time cost of requests and of overage payments, averaged over ten draws of the daily shocks for each developer-month.

surpluses over the month and subtract any overage she pays. Surplus is therefore measured in dollars relative to not using Copilot, before the $10 monthly fee. Myopic developers choose as if the allowance were free until it runs out, but we evaluate their surplus at the payoff they actually receive (Section 5.3). We average each developer-month's surplus over ten draws of the daily activity and demand shocks, and we use simulated developers who are active in every month, as in the estimation sample.

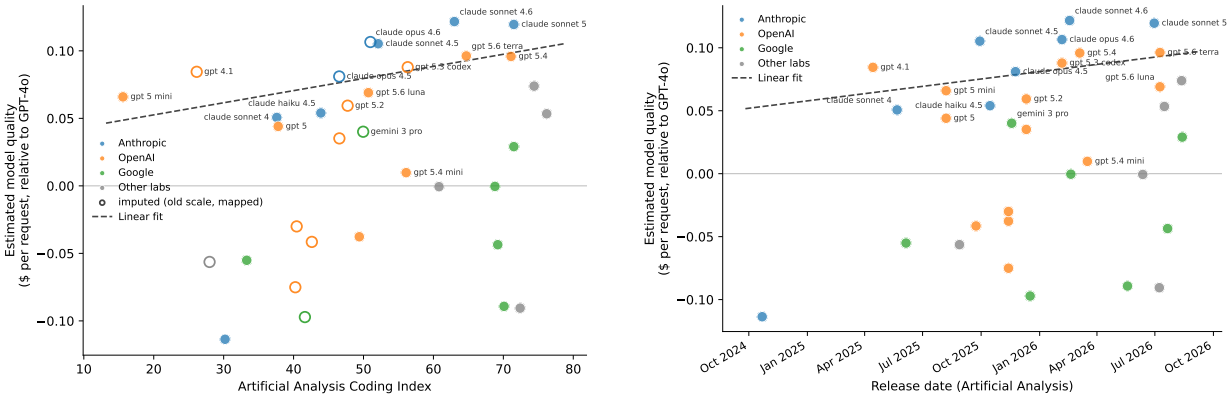
Panel (a) of Figure 7 shows the distribution of consumer surplus across developer-months. The average developer-month yields $30.9 of surplus and the median $18.8, against a monthly fee of $10, so 76% of developer-months yield more surplus than the fee.[20] The distribution is skewed: a developer-month at the 90th percentile yields $63.7, 3.4 times the median, while one at the 10th percentile yields $5.5. The skew reflects differences in how intensively developers use AI and how much it is worth in their work.

To quantify this heterogeneity, panel (b) relates surplus to how intensively developers use AI. Because a developer's usage in a given month also reflects the luck of that month's demand shocks, we sort developers by their expected monthly demand rather than by realized usage. This combines how often a developer is active, π_i, with how much she uses Copilot on an active day, $\exp(\mu_i/\beta)$, into the index $\pi_i \exp(\mu_i/\beta)$, which is fixed for

[20] This surplus covers chat and agent requests only, so it understates the value of the subscription, which also includes autocomplete.

Figure 8: Estimated Model Quality, Coding Ability and Release Date

(a) Artificial Analysis Coding Index **(b)** Release date



Notes: This figure plots the estimated quality of each model on the Copilot Pro menu against its Artificial Analysis Coding Index (Panel a) and its release date (Panel b), with markers by provider (Anthropic, OpenAI, Google and other providers). Quality is the model fixed effect δ_m expressed as the additional amount per request, in dollars at overage prices, that a developer with average tastes and match values would pay relative to GPT-4o. Hollow markers in Panel a are models whose index is imputed. The dashed line is the request-share-weighted linear fit across the models shown. GPT-4o and GPT-4o mini are omitted. Table OA-13 reports the estimates and their standard errors.

each developer, and we group developers into deciles of this index. Surplus rises steeply with expected demand. Developers in the top decile gain $110.3 a month on average, 26 times those in the bottom decile, and account for 36% of total surplus. Only developers from the 3rd decile upward gain more than the monthly fee on average.

7.5 Willingness to Pay for Intelligence

How much are developers willing to pay for a more intelligent model? The literature has measured the economic value of AI models mainly through benchmarks, which score models on standardized tasks. Our model fixed effects provide a different measure: they quantify how much developers are willing to pay for each model, as revealed by their choices and usage. In this section we relate these estimates to observed characteristics of the models, to understand how well benchmarks and other characteristics capture what developers are willing to pay for.

Panel (a) of Figure 8 plots the estimated quality of each model against its Artificial Analysis Coding Index. The relationship is positive: developers are willing to pay more for models that score higher on the index. It is far from perfect, however. The index explains only 10% of the variation in estimated quality, weighting models by their share of requests. Benchmark scores therefore capture part of what developers value in a model, but they do not measure its economic value. The linear fit implies that one additional point on the coding index is associated with an increase of $0.0009 in a developer's willingness

to pay per request, at overage prices.

We next turn to the relationship between a model's release date and willingness to pay. Panel (b) shows that newer models are valued more: willingness to pay rises by \$0.023 per request for each year between release dates, and release date explains 4% of the variation in estimated quality. Within a provider, newer versions of a model are generally preferred, but older models from one provider are often valued as much as newer models from another.

Overall, benchmarks and model vintage capture only part of developers' willingness to pay. Much of what developers value in a model lies outside these measures, for example its speed, its style of writing code and how well it uses tools in an agentic workflow. This has two implications. First, a demand system in which models differ only in benchmark scores would miss most of the differences between models that developers act on. Second, a price index for AI that adjusts only for benchmark performance would understate the quality improvements that developers value.

8 Counterfactuals

Our estimates describe how developers choose among models and how much they value each of them. In this section we use the estimated model to ask two questions. First, how did the consumer surplus that LLMs create for developers change as models entered the menu, and how much of it does each model and each provider generate? Second, how would that surplus change if the industry integrated vertically, with each provider selling its models only through its own plan? We answer both questions with the same welfare measure and under the same assumptions, which we describe next.

Unless stated otherwise, our base level of consumer surplus is the one a developer attains under the factual menu, prices and tariff. For each counterfactual we ask how much money would have to be given to or taken from a developer for her to attain the same utility as in the base environment, that is, her compensating variation. Because utility is quasilinear in money, compensating variation equals, up to sign, the change in the developer's realized monthly dollar payoff: the value of the requests she makes, net of the time cost of making them and any overage she pays. We report this payoff before the \$10 fee, which is the same in every counterfactual.[21]

Throughout, we make a number of assumptions consistent with a short-run analysis. We use the model to simulate developers' choices of model and quantity under each

[21] With quasilinear utility, compensating and equivalent variation coincide. We compute consumer surplus for the same panel of estimation-sample developers, who send at least one request in every month, and for myopic and overage-averse developers we evaluate the choices they make with the utility they experience, so that their mistakes in managing the allowance count as losses.

alternative menu. We hold preferences and behavioral types fixed and, except in the vertical-integration experiment of Section 8.2, keep each developer on her Copilot Pro plan.

8.1 Consumer Surplus from LLMs

Section 7.4 measured the consumer surplus developers obtain from the menu. Here we ask how it changes with the menu: how it grew as new models arrived over the sample, and how much of it is lost when a model or a provider is removed.

Consumer surplus over time. We first measure how consumer surplus changed over our sample. New frontier models arrived every few weeks between November 2025 and August 2026: Copilot added 22 models to the Pro menu over the period, among them new generations from each of the three main providers, and retired 11 of the 17 models available in November (Section 2). These changes raise two questions. How much did consumer surplus grow over the sample? And how much of that growth came from better models, rather than from changes in the prices at which developers could use them?

For each month from November 2025 to August 2026, we simulate the choices of the same panel of estimation-sample developers, with their estimated tastes, match values and behavioral types, under the Pro tariff in force after the billing change. We compute their consumer surplus per 30 days: the realized monthly payoff from coding with AI, net of the time cost of requests and of overage charges, averaged over 10 draws of the daily demand shocks. We report it before the $10 fee, which is common to all menus.[22] In each month we simulate the menu of that month at that month's prices; to isolate the value of new models, the menu also keeps the models that had been retired by then.[23]

Panel (a) of Figure 9 plots the resulting consumer surplus, indexed to November 2025. Developers gained substantially. For the average developer, surplus rises in almost every month and is 7% higher by August 2026 than in November, an increase from $31.0 to $33.2 per developer-month. The gain is concentrated among the heaviest users. The remaining lines of panel (a) follow fixed groups of developers around the 10th, 50th, 90th and 99th percentiles of expected monthly demand, the ranking of Figure 7. Developers around the 99th percentile gain about 31% by August, against 4% at the 90th percentile and 2 to 3% at the median and the 10th percentile.

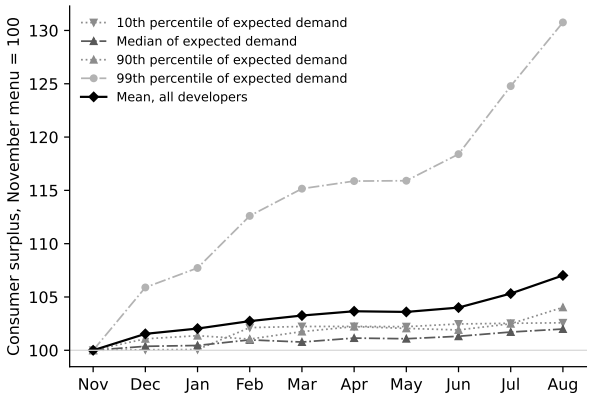
Where does this increase come from? There are two sources. The first is quality: new models with higher intelligence enter the menu. The second is prices, which change in

[22] We hold the tariff at its post-June form in every month so that the comparison isolates the menu and its prices from the change in billing.

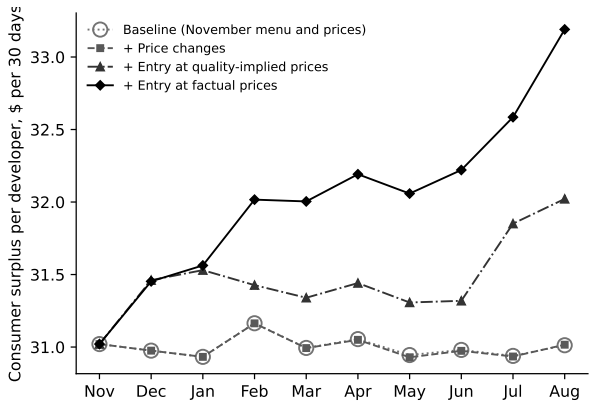
[23] Models that left the menu are kept at their last price, so that the change in surplus reflects entry alone.

Figure 9: Consumer Surplus over Time

(a) With entrants, by developer group



(b) Decomposition, mean across developers



Notes: This figure shows consumer surplus per developer-month from November 2025 to August 2026, computed for the same panel of estimation-sample developers (at least one request in every month) under the Pro tariff in force after the billing change ($15 of included usage, dollar-denominated overage), holding preferences, match values and behavioral types at their estimated values. Consumer surplus is the realized monthly dollar payoff, net of the time cost of requests and of overage and before the $10 fee, per 30 days, averaged over 10 draws of the daily demand shocks. "Baseline" is the November 2025 menu at November prices. "Price changes" moves the prices of the November models to their current values. "Entry at quality-implied prices" adds every model that had entered the Pro menu by that month, priced as the November relation between price and quality would price a model of its quality and provider (a regression of log price per request on the Artificial Analysis Intelligence Index with provider and month effects over all Pro menus, $R^2 = 0.74$); models that left the menu are kept, so the gap to "Price changes" is the value of entry alone. "Entry at factual prices" prices the entrants at the prices Copilot set. Panel (a) plots this last series relative to the November menu in the same month (November menu = 100), for all developers (mean) and for fixed groups of developers around the 10th, 50th, 90th and 99th percentiles of expected monthly demand, the ranking of Figure 7; panel (b) plots the means of all four series in dollars. Per-request prices hold each model's token mix fixed at a reference month so that only list prices vary. Table OA-14 reports the numbers.

two ways in this industry. The prices of the models already on the menu can change, as when a provider cuts the price of an existing model. And new models can change the price of intelligence itself: a model that enters at a lower price per unit of intelligence than the existing menu raises surplus beyond what its quality alone would.

To separate these sources, we simulate three further menus in each month. The first is the November menu at November prices, which serves as the baseline. The second keeps the November menu but moves its prices to their values in that month. The third adds the models that had entered by that month, but prices each entrant as the relation between price and quality on Copilot's menu would price a model of its quality and provider.[24] The menu of the month then prices the entrants at the prices Copilot set. Successive differences between the four menus give the contribution of price changes on the November menu, of

[24] We regress the log price per request on the Artificial Analysis Intelligence Index with provider and month fixed effects across all models on the Pro menu in all months of the sample (slope 0.12 log points per index point, $R^2 = 0.74$) and price each entrant at the fitted value for November. Per-request prices hold each model's tokens per request fixed at a reference month, so that only per-token prices vary.

entry at quality-implied prices, and of the pricing of the entrants relative to that relation.[25]

Panel (b) plots mean consumer surplus under the four menus, and Table OA-14 reports the numbers. Price changes of existing models play no role: the dotted and the dashed line coincide, and the difference between them is at most $0.02 in any month, because the per-token prices of the models already on the menu barely moved over the sample. This is how prices move in this industry: the price of a model rarely changes once it is released; it is new models that lower the price per unit of intelligence (Demirer et al., 2025).

The quality of the new models, valued at the existing price per unit of intelligence, is the gap between the dash-dotted and the dashed line: between $0.3 and $0.6 per developer-month from December to June and $1.0 (46% of the total gain) by August, after the summer entrants arrived. The pricing of the new models is the gap between the solid and the dash-dotted line, a further $1.2 (54%). This gap opens in February, with the first large wave of entrants, and broadly widens thereafter, because the new models were priced below what their quality and provider implied: the median entrant was priced at 74% of its quality-implied price. The baseline, the dotted line, is flat at about $31.0.[26] All of the $2.2 gain by August therefore comes from new models, roughly half from their quality and half from their lower price per unit of intelligence.

The value of each model and each provider. We next ask how much consumer surplus each model and each provider generates for developers. We measure it as the surplus developers lose when a model, or all models of a provider, is removed from the menu.

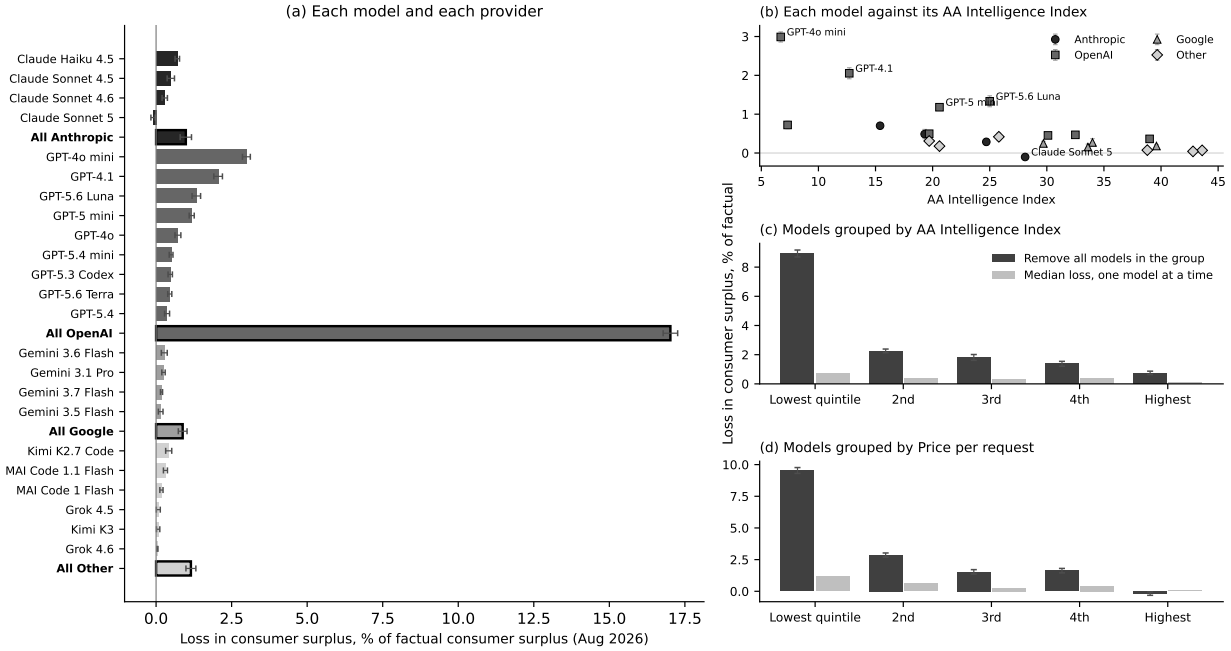
We perform these experiments on the menu of August 2026, the last month of our sample. For each model on that menu, we remove it and use the model to compute the choices developers would make: on each day, a developer who would have used the removed model moves to her next-best option, which may be another model or fewer requests. We do not renormalize tastes, so the remaining models keep their estimated quality and each developer keeps her provider tastes and match values. We then repeat the exercise for each provider, removing all of its models at once. In each case we compare consumer surplus with and without the removed models for the same developers, under the factual tariff and prices and the same demand shocks.

Figure 10 shows that no single model is essential. Panel (a) reports the loss from removing each model, in percent of consumer surplus. Removing any one model costs developers at most 3.0%, and for 19 of the 23 models less than 1%; the median loss is 0.36%.

[25] The split between the last two terms depends on the relation used to price the entrants; their sum, the value of entry at factual prices, does not.

[26] The small month-to-month movement of the baseline reflects the number of days in the month, since the included allowance is per calendar month, and simulation noise.

Figure 10: The Value of Each Model



Notes: Panel (a) reports the loss in consumer surplus, in percent of consumer surplus under the factual menu, from removing one model from the Copilot Pro menu of August 2026, without renormalizing tastes, for the same panel of estimation-sample developers under the factual tariff and prices; bars are grouped and shaded by provider (Other is Grok, Kimi and MAI), and the bold bar at the end of each group reports the loss from removing all of that provider's models at once. Panel (b) plots each model's loss against its Artificial Analysis Intelligence Index, with markers by provider as in panel (a) and labels on the models whose loss exceeds 1% or is negative. Panels (c) and (d) group the 23 models into quintiles of the Intelligence Index and of the price per request; the dark bar is the loss from removing every model in the quintile at once, the light bar the median loss across the single-model removals in it. Error bars are paired developer-bootstrap standard errors. Table OA-15 reports the losses in dollars and the destinations of displaced requests.

Developers move to a close substitute, usually another model of the same provider, and lose little. The most valuable models are the cheapest: the loss is largest for GPT-4o mini, the cheapest model on the menu, followed by GPT-4.1 (2.1%), while removing any of the frontier models costs well under 1%. Cheap models matter most because developers use them to stretch the included allowance.

Providers are a different matter. Removing all of OpenAI's models at once lowers consumer surplus by 17.0%, far more than the sum of the losses from removing each of its models alone (10.1%), mostly because OpenAI offers several cheap models (e.g., GPT 5.6 Luna and GPT 5.4 mini) that developers were substituting between in the one-model-at-a-time counterfactuals. The other providers are worth much less as a group: surplus falls by 1.0% without Anthropic, by 0.9% without Google and by 1.2% without the other providers. In dollars, the OpenAI and Anthropic losses are $5.58 and $0.32 per developer-month.

The gap reflects prices as much as quality. OpenAI has many cheap models on the

menu, and the developers who lose them turn to more expensive ones: without OpenAI, 82% of the displaced requests go to Anthropic models, 13% to Google and the other providers, and 4.5% are not made at all. Without Anthropic, 98% of the displaced requests go to OpenAI models at lower prices, so the loss is small. OpenAI's value to developers therefore comes from the number and the prices of its models rather than from their quality. Whether developers could do without several providers in one product, by choosing the single provider that suits them best, is the question of vertical integration we study in Section 8.2.

Panels (b) to (d) make the point by quality and by price. Panel (b) plots each model's loss against its Intelligence Index: the loss falls with the index, and the most capable models are among the least costly to remove. Panels (c) and (d) group the models into quintiles of the index and of the price per request and remove each quintile at once. Removing the cheapest quintile costs 9.5% of consumer surplus and removing the most expensive costs nothing (−0.2%); by the index, the least capable quintile costs 8.9% and the most capable 0.8%. In almost every quintile, the loss from removing the whole group far exceeds the median loss from removing one of its members (9.5% against 1.18% in the cheapest price quintile), the same substitution within a group that we see within a provider.

8.2 Vertical Integration

The surplus we measured above arises under today's industry structure, in which a single application, Copilot, sells models from several providers in a single product. This structure may not last: providers increasingly sell their models through coding tools of their own, such as Anthropic's Claude Code and OpenAI's Codex.[27] Hence, we now explore how consumer surplus would change if each provider sold its models only through its own plan, so that no developer could combine models from different providers.

In this experiment, each provider offers a plan that exclusively contains its own models, on the terms of Copilot Pro after the billing change: a $10 fee, a $15 usage allowance at list prices, and the per-request prices observed between June and August 2026. Each developer subscribes to the plan that yields her the highest expected consumer surplus over the three months and makes her daily choices within it; she cannot opt out. We assume that each provider's models have the same quality and prices in its own plan as in Copilot and that developers keep their tastes and match values, so the counterfactual changes only which models a developer can combine.

Table 7 reports the results. Panel A shows how developers sort across providers: 62%

[27] In August 2026, after SpaceX, the parent of the provider xAI, acquired the coding tool Cursor, OpenAI announced that it would withdraw its models from Cursor as of November 12, 2026.

Table 7: Full Vertical Integration: One Plan per Provider

	Developers choosing the plan of				All
	Anthropic	OpenAI	Google	Other providers	developers
Panel A. Who Chooses Each Provider's Plan					
Share of developers (%)	35.5	62.0	1.9	0.7	100
Factual share of their requests to the chosen provider (%)	81.2	81.9	57.8	63.4	80.8
Panel B. Change in Consumer Surplus ($ per Developer-Month)					
All developers in the group	-0.23 (0.01)	0.85 (0.04)	3.40 (0.39)	0.54 (0.75)	0.52 (0.03)
Forward-looking developers	-0.27 (0.01)	-0.43 (0.01)	-6.45 (0.67)	-11.21 (1.58)	-0.45 (0.01)
Myopic developers	-0.04 (0.02)	3.14 (0.10)	8.10 (0.39)	6.49 (0.54)	2.70 (0.07)
Light users	-0.02 (0.00)	0.07 (0.01)	-1.21 (0.35)	-1.39 (0.77)	0.04 (0.01)
Medium users	-0.13 (0.00)	0.13 (0.03)	-2.35 (0.48)	-2.49 (0.60)	-0.04 (0.02)
Heavy users	-0.66 (0.03)	2.14 (0.10)	4.49 (0.44)	1.08 (0.88)	1.54 (0.07)
Change in consumer surplus (%)	-1.2 (0.0)	2.4 (0.1)	5.7 (0.8)	0.7 (1.0)	1.7 (0.1)
Panel C. Change in Usage (%)					
Requests	-9.5 (0.2)	5.5 (0.1)	9.2 (1.2)	7.1 (1.4)	2.7 (0.1)
List-price spend	0.9 (0.1)	-17.3 (0.2)	-5.0 (0.5)	-8.4 (0.6)	-10.2 (0.1)

Notes: In the counterfactual, each provider sells its models only through its own plan, on the terms of Copilot Pro after the billing change (a $10 fee, an allowance of $15 at list prices, and the prices per request of June to August 2026), and each developer subscribes to the plan that maximizes her expected consumer surplus over the three months, computed as the mean over five simulated draws of the daily shocks; outcomes are averages over five further draws, each paired with the factual menu under the same draws. Columns group developers by the plan they choose; the last column reports all developers. Panel A reports the share of developers choosing each plan and the factual share of their requests sent to that provider. Panel B reports changes in consumer surplus relative to the factual multi-provider menu, in dollars per developer-month, overall, for forward-looking and for myopic developers (Section 5.3), by tercile of factual list-price spend, and in percent. Panel C reports percent changes in requests and in list-price spend. Standard errors from a paired developer bootstrap in parentheses. Developers are the simulated population of Section 7.4.

choose OpenAI's plan, 35% Anthropic's, and only 2% and 1% Google's and the other providers', whose plans contain few models. Developers mostly stay with the provider they already use most: 85% pick the plan of their most-used provider.

Panel B shows that forward-looking developers lose and myopic developers gain. Forward-looking developers, 69% of the population, lose $0.45 per month (1.4% of their surplus). The loss is small because little is displaced and each displaced request loses little. Only 15% of their requests went to the providers they give up, and each of these moves to the best remaining model at a cost of $0.035, against an average surplus of $0.39 per request. The best model within one provider's plan is nearly as good as the best model on the whole menu. The loss rises with usage, from $0.11 per month for light forward-looking users to $1.07 for heavy ones, who send the most requests to a second provider.

Myopic developers gain $2.70 per month on average (overage-averse gain $4.53). On the factual menu these developers use expensive models early in the month, reach the allowance and then cut back or stop. A plan limited to a single provider's models takes the expensive models of the other providers off the menu: on it the overage-averse send 26% more requests at 19% lower list-price spend and do not exhaust the allowance as early. The restriction therefore acts as a commitment device for developers who do not plan for

the allowance, and because they are 31% of the population, average surplus rises by $0.52 per month even though 59% of developers are worse off.

Panel C shows the effect on usage. Developers on Anthropic's plan lose OpenAI's cheap models for routine tasks and send 9% fewer requests. Developers on OpenAI's plan replace Anthropic's frontier models with cheaper ones and send 6% more requests at 17% lower list-price spend. Overall, requests rise by 3% and list-price spend falls by 10%.

9 Conclusion

How do developers use LLMs, and how do they respond to their prices? How differentiated are these models? How much consumer surplus do they create for developers, and how has it changed as new models arrived? And how would that surplus change if each provider sold its models only through its own tool? We study these questions in the context of GitHub Copilot, a coding tool that sells the models of several providers under one subscription with a monthly allowance, using developers' daily choices of model and quantity and a price experiment.

The usage data show how developers choose among these models and how much they use them. On a given day a developer usually works with a single model, but over a month she uses several, often from different providers, and her use varies greatly from one day to the next. New models are adopted quickly by developers who already use their provider and slowly by others. Developers also differ in their sophistication. Many do not plan for the allowance: they use it as if it were free, and many of them stop right after it runs out rather than pay for overage. Others are forward-looking and respond to the shadow price of the allowance even before they reach it. We find no sign of learning or habit formation: developers drawn to a model by a temporary price cut return to their previous models within weeks of its end.

Guided by this evidence, we estimate demand for LLMs from developers' daily choices of model and quantity within a monthly subscription. The monthly allowance makes the usage problem dynamic: as a developer uses up her allowance, the shadow price of each request rises, and it rises more for expensive models than for cheap ones. Changes in the menu and in prices over the sample identify how developers trade off quality, price and their tastes for particular providers and models.

Our estimates suggest that LLMs are differentiated, but more by model than by provider. Developers respond to a frontier model's price increase by switching models, usually within the same provider, and to a cheap model's price increase by sending fewer requests. Benchmark scores are correlated with the quality developers attach to models, but much of it remains unexplained by them. The consumer surplus these models create

is large and grew over our sample, mostly through the entry of new models, which lowered the price per unit of intelligence, and mostly for the heaviest users. No single model is essential, because developers find close substitutes. Overall, a market in which each provider sold its models only through its own plan would leave consumer surplus almost unchanged, so access to every provider in one tool is worth little to developers.

Taken together, our paper provides early evidence on the market structure of LLMs as it takes shape. While the magnitudes we estimate are specific to our sample and period, the way developers use these models, switching easily between them, relying on the cheapest ones and valuing each new entrant, is likely to outlast them. These patterns inform questions that will recur as the market matures: how much differentiation there is between providers, what multi-provider applications are worth, and who gains when providers integrate vertically.

References

Bergemann, D., A. Bonatti, and A. Smolin (2026). Menu Pricing of Large Language Models. *arXiv Preprint arXiv:2502.07736*.

Berry, S., J. Levinsohn, and A. Pakes (1995). Automobile Prices in Market Equilibrium. *Econometrica 63*(4), 841–890.

Berry, S., J. Levinsohn, and A. Pakes (2004). Differentiated Products Demand Systems from a Combination of Micro and Macro Data: The New Car Market. *Journal of Political Economy 112*(1), 68–105.

Berry, S. T. and P. A. Haile (2021). Foundations of Demand Estimation. In K. Ho, A. Hortaçsu, and A. Lizzeri (Eds.), *Handbook of Industrial Organization*, Volume 4, pp. 1–62. Elsevier.

Brynjolfsson, E., A. Collis, F. Eggers, S. Kazinnik, and D. Nguyen (2026). What is Generative AI Worth? *SSRN Working Paper, No. 6569938*.

Caoui, E. H. (2026). Estimating Consumer Preferences for LLMs: Evidence from LMArena. *Working Paper, University of Toronto*.

Cartis, C., J. Fiala, B. Marteau, and L. Roberts (2019). Improving the Flexibility and Robustness of Model-Based Derivative-Free Optimization Solvers. *ACM Transactions on Mathematical Software 45*(3), 32:1–32:41.

Chen, F. and J. Stratton (2026). Artificial Intelligence in the Firm: Bottlenecks in Software Production. Working paper.

Cui, Z., M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz (2026). The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. *Management Science*. Forthcoming.

Demirer, M., A. Fradkin, N. Tadelis, and S. Peng (2025). The Emerging Market for Intelligence: Pricing, Supply, and Demand for LLMs. *NBER Working Paper, No. 34608*.

Demirer, M., L. Musolff, and L. Yang (2026). Writing Code vs. Shipping Code: Productivity Effects Across Generations of AI Coding Tools. *NBER Working Paper, No. 35275*.

Dubé, J.-P. (2004). Multiple Discreteness and Product Differentiation: Demand for Carbonated Soft Drinks. *Marketing Science 23*(1), 66–81.

Egan, M. L., D. Russel, C. Shi, and V. Yao (2026). Competition in Generative AI: Evidence from Consumer Demand. *NBER Working Paper, No. 35953*.

Eizenberg, A. (2014). Upstream Innovation and Product Variety in the U.S. Home PC Market. *Review of Economic Studies 81*(3), 1003–1045.

Ershov, D., J.-W. Laliberté, M. Marcoux, and S. Orr (2025). Estimating Complementarity with Large Choice Sets: An Application to Mergers. *RAND Journal of Economics 56*(4), 689–707.

Gentzkow, M. (2007). Valuing New Goods in a Model with Complementarity: Online Newspapers. *American Economic Review 97*(3), 713–744.

Goolsbee, A. and A. Petrin (2004). The Consumer Gains from Direct Broadcast Satellites and the Competition with Cable TV. *Econometrica 72*(2), 351–381.

Grubb, M. D. and M. Osborne (2015). Cellular Service Demand: Biased Beliefs, Learning, and Bill Shock. *American Economic Review 105*(1), 234–271.

Handa, K., A. Tamkin, M. McCain, S. Huang, E. Durmus, S. Heck, J. Mueller, J. Hong, S. Ritchie, T. Belonax, K. K. Troy, D. Amodei, J. Kaplan, J. Clark, and D. Ganguli (2025). Which Economic Tasks are Performed with AI? Evidence from Millions of Claude Conversations. *arXiv Preprint arXiv:2503.04761*.

Hausman, J. A. (1996). Valuation of New Goods under Perfect and Imperfect Competition. In T. F. Bresnahan and R. J. Gordon (Eds.), *The Economics of New Goods*, pp. 207–248. University of Chicago Press for NBER.

Hendel, I. (1999). Estimating Multiple-Discrete Choice Models: An Application to Computerization Returns. *Review of Economic Studies 66*(2), 423–446.

Iaria, A. and A. Wang (2020). Identification and Estimation of Demand for Bundles. *CEPR Discussion Paper, No. 14363*.

Jimenez, C. E., J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *International Conference on Learning Representations (ICLR)*.

Lambrecht, A., K. Seim, and B. Skiera (2007). Does Uncertainty Matter? Consumer Behavior under Three-Part Tariffs. *Marketing Science 26*(5), 698–710.

McFadden, D. (1989). A Method of Simulated Moments for Estimation of Discrete Response Models without Numerical Integration. *Econometrica 57*(5), 995–1026.

Nevo, A. (2000). A Practitioner's Guide to Estimation of Random-Coefficients Logit Models of Demand. *Journal of Economics & Management Strategy 9*(4), 513–548.

Nevo, A. (2001). Measuring Market Power in the Ready-to-Eat Cereal Industry. *Econometrica 69*(2), 307–342.

Nevo, A., J. L. Turner, and J. W. Williams (2016). Usage-Based Pricing and Demand for Residential Broadband. *Econometrica 84*(2), 411–443.

Pakes, A. and D. Pollard (1989). Simulation and the Asymptotics of Optimization Estimators. *Econometrica 57*(5), 1027–1057.

Petrin, A. (2002). Quantifying the Benefits of New Products: The Case of the Minivan. *Journal of Political Economy 110*(4), 705–729.

Sarkar, S. K. (2026). AI Agents and Higher-Order Work. *SSRN Working Paper, No. 5713646*.

Stanford HAI (2025). The 2025 AI Index Report. *Stanford Institute for Human-Centered Artificial Intelligence*.

Thomassen, Ø., H. Smith, S. Seiler, and P. Schiraldi (2017). Multi-Category Competition and Market Power: A Model of Supermarket Pricing. *American Economic Review 107*(8), 2308–2351.

Trajtenberg, M. (1989). The Welfare Analysis of Product Innovations, with an Application to Computed Tomography Scanners. *Journal of Political Economy 97*(2), 444–479.

Product Differentiation and Consumer Surplus in the Market for LLMs: Evidence from Software Developers

Mert Demirer Leon Musolff Max Schnidman

Appendix - For Online Publication

Contents

A Data Appendix . . . . . **OA - 2**
A.1 Raw Telemetry OA - 2
A.2 Monthly Cleaning OA - 3
A.3 Balanced Panel OA - 3
A.4 Menus and Regimes OA - 4
A.5 Prices and the Attribution of Spend OA - 5
A.6 Budget State and Spend Outside Choices OA - 6
A.7 Model Characteristics OA - 7

B Model Solution . . . . . **OA - 9**
B.1 Notation OA - 9
B.2 The Developer's Problem with an Overage State OA - 9
B.3 The Overage State OA - 10
B.4 Menu Changes OA - 11
B.5 Numerical Solution OA - 11
B.6 Additional Model Assumptions OA - 11
B.7 Proofs OA - 12

C Estimation Details . . . . . **OA - 14**
C.1 Simulation of the Moments OA - 14
C.2 Moment Definitions OA - 14
C.3 Fixed and Pinned Parameters OA - 15
C.4 Optimization OA - 16
C.5 Standard Errors OA - 16

D Additional Tables and Figures . . . . . **OA - 18**
D.1 Additional Tables for Section 2 OA - 18
D.2 Additional Exhibits for Section 4 OA - 20
D.3 Spending after the Allowance Is Used Up OA - 21
D.4 Additional Tables for Section 5 OA - 21
D.5 Additional Tables for Section 7 OA - 21
D.6 Additional Tables for Section 8 OA - 22

Table OA-1: Variables in the Usage Data

Field	Definition	Recorded in
Developer identifier	Hashed account identifier	All months
Date	Calendar day of the request (UTC)	All months
Plan	Copilot Pro or Pro+	All months
Annual flag	Subscriber pays annually	All months
Model label	Model that served the request	All months
Selection method	Chosen by the developer, by auto mode, or unknown (code review)	All months
Initiation type	Initiated by the user or by the system	All months
Request count	Number of requests in the row	All months
Billed cost	Points charged (before June) or dollars (from June)	All months
Linked backend spend	Dollar spend linked from backend billing records	Jun–Aug 2026
Tokens	Input and output tokens	Jun–Aug 2026
Editor integration	Editor through which the request was sent	All months

Notes: This table lists the fields of GitHub's Copilot telemetry that we use. Each row of the raw data records the requests one developer sent to one model on one day through one selection method and initiation type. Linked backend spend and token counts are recorded only under token-based billing.

A Data Appendix

This appendix describes the data underlying our empirical analysis. We first describe the raw usage records and the fields we use, then the monthly cleaning and the balanced panel, and finally how we construct menus, prices, the budget state and model characteristics.

A.1 Raw Telemetry

The usage data are GitHub's internal Copilot telemetry for a large subsample of individual Pro and Pro+ subscribers, delivered as one export per calendar month from November 2025 to August 2026. Each monthly export contains between 161 and 332 million rows, and each row records the requests one developer sent to one model on one day through one selection method and initiation type. Table OA-1 defines every field we use. The size of the subsample is confidential, so the developer counts in this appendix do not reveal the number of Copilot subscribers.

Not every row is a model choice. Code review requests and other rows that cannot be attributed to a model the developer selected count toward the budget state, monthly totals and overage, because they draw down the same allowance, but they do not enter the choice data. A day on which a developer has only such rows is treated as inactive.

The January 2026 export contains 8,415 records, carrying 116,245 requests, whose request counts are corrupted in a systematic way. We repair them by reinterpreting the stored values, and the repaired counts match the billed cost of the same rows.

We map raw model labels to a universe of 48 models, of which 39 are on the Copilot Pro

Table OA-2: Monthly Sample Cleaning

	Dropped				Kept, by plan		More than ten
Month	Mixed plan	Annual	No request	Kept	Pro	Pro+	auto requests
November 2025	14.9	20.6	1.3	63.3	58.4	4.9	2.7
December 2025	15.0	20.1	1.4	63.4	57.6	5.8	4.4
January 2026	15.0	20.0	1.3	63.6	57.1	6.6	6.6
February 2026	15.5	19.7	1.2	63.6	56.0	7.6	10.9
March 2026	18.8	18.9	1.5	60.8	52.1	8.7	14.2
April 2026	23.3	16.6	1.4	58.6	48.6	10.0	17.8
May 2026	10.6	18.4	1.7	69.4	55.1	14.2	23.3
June 2026	21.6	17.1	2.2	59.2	47.6	11.6	18.6
July 2026	21.0	16.1	3.0	59.9	47.5	12.4	20.2
August 2026	19.0	14.6	2.7	63.7	50.2	13.6	21.8

Notes: This table reports, for each month and as a percentage of the developers with any Copilot Pro or Pro+ record in that month, the developers dropped because they appear on more than one plan in the month, have an annual record, or have no user-initiated record, the developers kept and their split between Pro and Pro+, and the kept developers with more than ten auto-mode requests in the month. The columns Mixed plan, Annual, No request and Kept sum to 100. New sign-ups were paused from April 20 to June 17, 2026.

menu at some point. The 0.10% of explicitly selected requests that go to labels outside this universe, mostly rare or experimental models, count toward daily totals and the budget state but not as model choices.

A.2 Monthly Cleaning

In each month we keep user-initiated records of developers on a single individual plan. We drop, in every month, any developer with an annual record, because annual subscribers stayed on request-based billing after the billing change, with different prices in points. We also drop developers without any user-initiated record in the month; after the billing change these are developers whose only usage is system-initiated. Table OA-2 reports the share of developers dropped at each step. Between 4 and 34% of the developers kept in a month (between 2.7 and 23.3% of all developers with a record, last column of Table OA-2) send more than ten auto-mode requests; the balanced panel below excludes them.

GitHub paused new Pro and Pro+ sign-ups from April 20 to June 17, 2026. The monthly population therefore has no new entrants in May and early June, which explains part of the movement in the counts in Table OA-2.

A.3 Balanced Panel

A developer is present in a month if she makes at least one user-initiated request, code review included. We require presence in all ten months.

We then keep developers who are on the same plan in every month, who send at most ten auto-mode requests in any month, and who never use Copilot through JetBrains editors

Table OA-3: Estimation Sample and Monthly Pro Population

	Estimation sample	Monthly Pro population
Share of days active	0.33	0.28
Requests per active day	8.9	13.3
Share of months reaching the allowance	0.12	0.18
Anthropic share of requests	(withheld)	(withheld)
Auto-mode share of requests	0.01	0.12
Share using more than one model	0.73	0.72
Developers	(withheld)	(withheld)

Notes: This table compares developers in the estimation sample with all developers on Copilot Pro in the same months after monthly cleaning (Appendix A.2). Statistics are means over developer-months. The population column includes developers who are not present in every month. Some numbers have been censored as they have not yet been cleared for public disclosure.

(Section 3.2). In auto mode Copilot routes the request to a model of its own choosing; our model has no such routing, so we restrict attention to developers for whom auto mode is negligible. We drop the few auto-mode requests these developers do send and use only explicitly chosen requests. Finally, we keep Copilot Pro developers only: Pro has several times more subscribers than Pro+, so that the sample remains large enough to estimate the model after these restrictions, and the two plans differ in their allowance.

The estimation sample is more persistent but less intensive than the monthly Pro population. Table OA-3 shows that, compared with all monthly Pro developers in the same months, panel developers are active on more days of the month but send fewer requests per active day and reach the allowance less often; by construction they send almost no auto-mode requests, against about one request in eight in the population. The share using more than one model in a month is the same in both groups.

A.4 Menus and Regimes

Before the billing change we infer each day's menu from the data. A model is on the menu on a given day if at least the larger of ten developers and 0.03% of that day's users select it explicitly, and we fill gaps of up to three days. A new regime starts whenever the set of models or their prices in points change.

A model's price in points is the modal charge per explicitly selected request. At least 84% of each model's requests in a regime carry exactly the modal charge, and the remainder reflect the auto-mode discount and rounding.

After the billing change we use dated menu lists, which we check against archived snapshots of GitHub's supported-models documentation.

Before the billing change, for several models the retirement dates detected in the data lag the documented dates by a few days; we use the data dates, because they record what

Table OA-4: Menu Regimes, Copilot Pro

Month	Days	Models	Change that starts the regime
Nov 2025	1–10	12	Sample start
	11–12	11	Claude Sonnet 3.5 exits
	13–17	14	GPT-5.1, GPT-5.1 Codex and GPT-5.1 Codex mini enter
	18–23	15	Gemini 3 Pro enters
	24–30	16	Claude Opus 4.5 enters
Dec 2025	1–3	16	Month start
	4–5	17	GPT-5.1 Codex Max enters
	6–10	17	Claude Opus 4.5 1 → 3 points
	11–16	18	GPT-5.2 enters
	17–31	19	Gemini 3 Flash enters
Jan 2026	1–13	19	Month start
	14–31	20	GPT-5.2 Codex enters
Feb 2026	1–4	20	Month start
	5–10	21	Claude Opus 4.6 enters
	11–15	22	GPT-5.3 Codex enters
	16–16	22	Grok Code Fast 1 0 → 0.25 points
	17–18	23	Claude Sonnet 4.6 enters
	19–28	22	Gemini 3.1 Pro enters; GPT-5 and GPT-5 Codex exit
Mar 2026	1–5	22	Month start
	6–16	23	GPT-5.4 enters
	17–26	24	GPT-5.4 mini enters
	27–31	23	Gemini 3 Pro exits
Apr 2026	1–1	23	Month start
	2–5	21	GPT-5.1 Codex and GPT-5.1 Codex mini exit
	6–15	20	GPT-5.1 Codex Max exits
	16–20	19	GPT-5.1 exits
	21–30	17	Claude Opus 4.5 and Claude Opus 4.6 exit
May 2026	1–6	17	Month start
	7–15	16	Claude Sonnet 4 exits
	16–18	15	Grok Code Fast 1 exits
	19–20	16	Gemini 3.5 Flash enters
	21–31	17	Claude Opus 4.6 enters
Jun 2026	1–30	13	Token-based billing starts; Claude Sonnet 5 and MAI-Code-1-Flash enter; Claude Opus 4.6, GPT-4.1, GPT-4o, GPT-4o mini, GPT-5.2 and GPT-5.2 Codex exit
Jul 2026	1–8	17	Month start; GPT-4.1, GPT-4o, GPT-4o mini and Kimi K2.7 Code enter
	9–20	19	GPT-5.6 Luna and GPT-5.6 Terra enter
	21–27	20	Gemini 3.6 Flash enters
	28–31	21	Grok 4.5 enters
Aug 2026	1–5	19	Month start; Gemini 2.5 Pro and Gemini 3 Flash exit
	6–10	20	Kimi K3 enters
	11–12	21	MAI-Code-1.1-Flash enters
	13–13	22	Gemini 3.7 Flash enters
	14–31	23	Grok 4.6 enters

Notes: This table lists the menu regimes of Copilot Pro, one row per regime. A regime is a spell of days with an unchanged set of models and prices; a new regime starts when a model enters or exits or a model's price changes, and at the start of each month. Before the billing change regimes are detected from the data (Appendix A.4); after the billing change they follow GitHub's dated menu lists. Days are the first and last day of the regime; Models is the number of models on the menu.

developers could actually select.

A.5 Prices and the Attribution of Spend

Before the billing change a model's price is its charge in points per request, the modal charge described above. After the billing change the price of a model in a regime is its

Table OA-5: Menu and Prices by Month

		Models by provider					Price per request		
Month	Models	Anthropic	OpenAI	Google	Other	Included	Min	Median	Max
November 2025	17	5	9	2	1	5	0.33	1	1
December 2025	19	4	11	3	1	5	0.33	1	3
January 2026	20	4	12	3	1	5	0.33	1	3
February 2026	24	6	13	4	1	4	0.25	1	3
March 2026	24	6	13	4	1	4	0.25	1	3
April 2026	23	6	13	3	1	4	0.25	1	3
May 2026	19	5	9	4	1	4	0.25	1	14
June 2026	13	4	4	4	1	0	\$0.019	\$0.185	\$0.608
July 2026	21	4	9	5	3	0	\$0.003	\$0.142	\$0.417
August 2026	23	4	9	4	6	0	\$0.002	\$0.145	\$0.421

Notes: This table reports, for each month, the number of models on the Copilot Pro menu at any time in the month, by provider, the number of included (zero-price) models, and the minimum, median and maximum price per request across the paid models. A model's price in a month is its price in the regime that covers most of the month's days before the billing change, and the day-weighted average of its regime prices afterwards. Prices are points per request through May 2026 and dollars per request after the billing change.

attributed spend divided by the user-initiated requests sent to it, computed over the Pro developers of the estimation sample who selected it explicitly. Attributed spend includes the backend requests that a user-initiated request triggers. Before June we attribute these backend requests to the user interaction ourselves, from the timing and model of the records; from June onward GitHub's export attributes them upstream, to the interaction that caused them, and we use that attribution. The two methods differ somewhat on the one month we observe under both, and we do not adjust for the difference. Spend that cannot be attributed to any interaction, and code review spend, counts toward the budget state but not toward any model's price.

A.6 Budget State and Spend Outside Choices

The budget state is a developer's cumulative spend from the start of the calendar month to the start of the day. It resets on the first of each month at 00:00 UTC regardless of the subscription billing date, as GitHub's allowance does.

After the billing change, 33, 36 and 40% of panel spend in June, July and August comes from requests that are not explicit choices. Code review accounts for 39% of this spend and backend requests that GitHub's export cannot attribute to any interaction for 60%.

Counting all spend, 25.5% of panel developers reach the \$15 allowance in June, against 22.5% when code review is excluded.

A.7 Model Characteristics

We use two measures of model capability. The first is the Coding Index of Artificial Analysis, which also provides each model's provider, release date and list prices (Section 3.1). The second is the SWE-bench Verified score. We take the highest score reported for each model under any scaffold. For 13 models without a reported score we use the score of a closely related model, and for 7 models we rely only on third-party evaluations. Table OA-6 lists every model with its score and source.

Table OA-6: Model Universe

Model	Provider	SWE-bench Verified	Source	On the Pro menu: First day	On the Pro menu: Last day	Points per request (Nov–May)	Price per request ($): Jun	Price per request ($): Jul	Price per request ($): Aug
Anthropic									
Claude Fable 5	Anthropic	95.0	Third-party	–	–	–	–	–	–
Claude Haiku 4.5	Anthropic	73.3	Official	Nov 1, 2025	Aug 31, 2026	0.33	$0.108	$0.116	$0.089
Claude Opus 4.5	Anthropic	80.9	Official	Nov 24, 2025	Apr 20, 2026	1/3	–	–	–
Claude Opus 4.6	Anthropic	81.4	Official	Feb 5, 2026	May 31, 2026	3	–	–	–
Claude Opus 4.6 (1M)	Anthropic	81.4	Related model	–	–	–	–	–	–
Claude Opus 4.7	Anthropic	87.6	Official	–	–	–	–	–	–
Claude Opus 4.8	Anthropic	88.6	Official	–	–	–	–	–	–
Claude Opus 4.8 (fast)	Anthropic	88.6	Related model	–	–	–	–	–	–
Claude Opus 5	Anthropic	96.0	Official	–	–	–	–	–	–
Claude Sonnet 3.5	Anthropic	49.0	Official	Nov 1, 2025	Nov 10, 2025	1	–	–	–
Claude Sonnet 4	Anthropic	72.7	Official	Nov 1, 2025	May 6, 2026	1	–	–	–
Claude Sonnet 4.5	Anthropic	77.2	Official	Nov 1, 2025	Aug 31, 2026	1	$0.227	$0.224	$0.223
Claude Sonnet 4.6	Anthropic	80.2	Official	Feb 17, 2026	Aug 31, 2026	1	$0.336	$0.356	$0.301
Claude Sonnet 5	Anthropic	79.6	Third-party	Jun 1, 2026	Aug 31, 2026	–	$0.608	$0.417	$0.350
OpenAI									
GPT-4.1	OpenAI	54.6	Official	Nov 1, 2025	Aug 31, 2026	0	–	$0.011	$0.011
GPT-4o	OpenAI	33.2	Official	Nov 1, 2025	Aug 31, 2026	0	–	$0.028	$0.027
GPT-4o mini	OpenAI	8.7	Official	Nov 1, 2025	Aug 31, 2026	0	–	$0.003	$0.002
GPT-5	OpenAI	72.8	Official	Nov 1, 2025	Feb 18, 2026	1	–	–	–
GPT-5 Codex	OpenAI	74.5	Official	Nov 1, 2025	Feb 18, 2026	1	–	–	–
GPT-5 mini	OpenAI	71.0	Official	Nov 1, 2025	Aug 31, 2026	0	$0.019	$0.020	$0.021
GPT-5.1	OpenAI	76.3	Official	Nov 13, 2025	Apr 15, 2026	1	–	–	–
GPT-5.1 Codex	OpenAI	73.7	Official	Nov 13, 2025	Apr 1, 2026	1	–	–	–
GPT-5.1 Codex Max	OpenAI	77.9	Official	Dec 4, 2025	Apr 5, 2026	1	–	–	–
GPT-5.1 Codex mini	OpenAI	71.3	Official	Nov 13, 2025	Apr 1, 2026	0.33	–	–	–
GPT-5.2	OpenAI	80.0	Official	Dec 11, 2025	May 31, 2026	1	–	–	–
GPT-5.2 Codex	OpenAI	80.0	Related model	Jan 14, 2026	May 31, 2026	1	–	–	–
GPT-5.3 Codex	OpenAI	78.0	Third-party	Feb 11, 2026	Aug 31, 2026	1	$0.185	$0.174	$0.180
GPT-5.4	OpenAI	78.2	Third-party	Mar 6, 2026	Aug 31, 2026	1	$0.220	$0.222	$0.210
GPT-5.4 mini	OpenAI	73.0	Third-party	Mar 17, 2026	Aug 31, 2026	0.33	$0.068	$0.061	$0.059
GPT-5.4 nano	OpenAI	73.0	Related model	–	–	–	–	–	–
GPT-5.5	OpenAI	82.6	Third-party	–	–	–	–	–	–
GPT-5.6 Luna	OpenAI	71.0	Related model	Jul 9, 2026	Aug 31, 2026	–	–	$0.098	$0.031
GPT-5.6 Sol	OpenAI	82.6	Related model	–	–	–	–	–	–
GPT-5.6 Terra	OpenAI	78.2	Related model	Jul 9, 2026	Aug 31, 2026	–	–	$0.265	$0.193
Google									
Gemini 2.5 Pro	Google	63.8	Official	Nov 1, 2025	Jul 31, 2026	1	$0.059	$0.055	–
Gemini 3 Flash	Google	78.0	Official	Dec 17, 2025	Jul 31, 2026	0.33	$0.048	$0.046	–
Gemini 3 Pro	Google	76.2	Official	Nov 18, 2025	Mar 26, 2026	1	–	–	–
Gemini 3.1 Pro	Google	80.6	Official	Feb 19, 2026	Aug 31, 2026	1	$0.201	$0.165	$0.188
Gemini 3.5 Flash	Google	79.3	Third-party	May 19, 2026	Aug 31, 2026	14	$0.272	$0.228	$0.181
Gemini 3.6 Flash	Google	79.3	Related model	Jul 21, 2026	Aug 31, 2026	–	–	$0.142	$0.137
Gemini 3.7 Flash	Google	79.3	Related model	Aug 13, 2026	Aug 31, 2026	–	–	–	$0.133
xAI									
Grok 4.5	xAI	70.8	Related model	Jul 28, 2026	Aug 31, 2026	–	–	$0.352	$0.323
Grok 4.6	xAI	70.8	Related model	Aug 14, 2026	Aug 31, 2026	–	–	–	$0.421
Grok Code Fast 1	xAI	70.8	Official	Nov 1, 2025	May 15, 2026	0/0.25	–	–	–
Moonshot									
Kimi K2.7 Code	Moonshot	67.5	Related model	Jul 1, 2026	Aug 31, 2026	–	–	$0.156	$0.145
Kimi K3	Moonshot	67.5	Related model	Aug 6, 2026	Aug 31, 2026	–	–	–	$0.349
Microsoft AI									
MAI-Code-1-Flash	Microsoft AI	71.6	Official	Jun 1, 2026	Aug 31, 2026	–	$0.091	$0.058	$0.077
MAI-Code-1.1-Flash	Microsoft AI	72.6	Official	Aug 11, 2026	Aug 31, 2026	–	–	–	$0.019

Notes: This table lists the models in the universe over which model quality is normalized (Section 5.4), including models that are on the Pro+ menu but never on the Pro menu. SWE-bench Verified is the highest score reported for the model under any scaffold. Source is Official for scores reported by the provider or the benchmark maintainers, Third-party for scores from independent evaluations only, and Related model for models without a reported score, which take the score of a closely related model (the SWE table in the replication package names it). First and last day, points per request and prices per request are defined as in Table OA-9. A dash indicates that the model is not on the Pro menu in that period. Two values of points per request separated by a slash record a price change, in chronological order.

B Model Solution

This appendix states the developer's problem with an overage state and a general continuation weight, describes how we solve and simulate it, discusses secondary assumptions, and proves the results in Section 5.

B.1 Notation

Table OA-7 collects the notation used in Sections 5 to 8. Throughout, "requests" always means model calls and "points" always means requests times points per request, so that a model's points per request α_m converts requests into draws on the allowance. Before the billing change points are GitHub's premium requests, and after it they are dollars of list-price spend.

B.2 The Developer's Problem with an Overage State

The state in Section 5 contains the day of the month and the remaining allowance. Here we add an overage flag. The flag switches on the first day in a month on which the developer pays for overage, and from then on the remaining allowance is zero and every point costs the overage price p, \$0.04 per point before the billing change and one dollar per dollar of list-price spend after it.

On a day with remaining allowance, the developer compares three options for each model: the best request count in the free region, where requests are bounded by the smaller of the remaining allowance divided by α_m and her satiation point; the best request count in the paid region, which uses up the allowance and enters overage; and not using the model. She then chooses the model and option with the highest decision value.

Models that use no points, the included models before the billing change, are always in the free region. A developer who chooses one of them therefore uses it at her satiation point, where marginal utility equals the time cost κ, regardless of her remaining allowance.

With a general continuation weight $\rho \geq 0$, the developer maximizes a decision value equal to flow utility plus ρ times the change in continuation value. The expected value recursion, in contrast, uses experienced values under her own future policy, so a myopic developer ($\rho = 0$) still has a well-defined value function that we use for welfare.

We implement the overage-averse type's constraint as a prohibitive cost F_i of entering the overage state. The cost is large enough that no simulated developer ever pays it, so the implementation reproduces the constraint in Section 5.3 exactly while keeping one solver for all types.

Table OA-7: Model Notation

Symbol	Definition
Indices	
i, m, k	Developer; model; model
t, T	Day of the month; days in the month
$r(t)$	Regime of day t: block of days with a fixed menu and prices
$\mathcal{M}_r, \mathcal{U}$	Menu in regime r; fixed universe of models
Tariff and state	
$\bar{C}$	Monthly allowance, in points
α_m	Points per request to model m
p	Overage price per point ($)
R_t	Points remaining at the start of day t
q	Points used on a day
Shocks and demand	
π_i	Probability of being active on a day
v_t, z_t	Daily demand shock; its standard normal draw
μ_i, σ_i	Mean and SD of log demand shock
c	Requests on a day
Utility	
γ_{im}	Quality of model m for developer i
β_i, B_i	Curvature, an affine function of a beta draw B_i on a bounded interval
κ_i	Time cost per request ($)
$\tau_m = \kappa_i + \lambda\alpha_m$	Full marginal cost of a request
Quality index and tastes	
δ_m	Model fixed effect
A_m, SWE_m	Anthropic indicator; SWE-bench Verified score
b_i^A, b_i^S	Anthropic and SWE-bench tastes, variances v_A, v_S
$\varepsilon_{im} = s_i G_{im}$	Persistent match value; G_{im} standard Gumbel
$s_i = \bar{s}\exp(g\, z_i^{\mu})$	Match scale, rising with demand level
Types	
ρ_i	Continuation weight: forward-looking or myopic
F_i	Cost of entering overage; prohibitive for overage-averse types, zero otherwise
p_ρ, p_F	Probability forward-looking; probability overage-averse if myopic
Dynamics	
λ	Shadow price of a point
$V_{it}(v, R)$	Value of an active day
$EV_{it}(R)$	Expected value of entering day t with R points
Distribution parameters	
m_x, v_x	Location and variance of the distribution of x
$m_\pi, c_\pi; m_B, c_B$	Mean and concentration of beta distributions

Notes: Points are what GitHub calls premium requests before the billing change and dollars of list-price spend afterwards. All money amounts are in dollars per developer. Distribution parameters follow the convention in the last block: for a normal or log-normal variable x, m_x and v_x are the location and variance of x or $\log x$.

B.3 The Overage State

Once a developer is in the overage state, her daily problem is static. On each active day she picks the model and number of requests that maximize flow utility net of overage

payments, and the value of the rest of the month is the sum of these expected daily values over the remaining days.

We solve for the difference between the overage-state value and the value at zero remaining allowance through its own backward recursion rather than as the difference of two separately computed values. This avoids cancellation error when the two values are close, and the difference is identically zero when entering overage is costless.

B.4 Menu Changes

Menus and prices change within months. We solve each regime as if it lasted until the end of the month and carry the remaining allowance and the overage flag across regime boundaries, so developers do not anticipate menu changes. Regimes with identical menus and prices share value functions up to a shift in the horizon, which we exploit to reduce computation.

B.5 Numerical Solution

We discretize the remaining allowance on a uniform grid of 100 points, refined near zero when entering overage is costly. We integrate over the truncated daily demand shock with Gauss–Legendre quadrature.

We store value functions as piecewise-linear functions of the remaining allowance. The decision-relevant marginal values come from integrated envelope slopes rather than from differencing the stored levels: the marginal value of a point is the overage price when the developer pays, the slope of the continuation value when her choice is interior, and her marginal utility per point at the kink where the allowance runs out.

With several models, the continuation value need not be concave in the remaining allowance. We therefore maximize exactly over the free region, which is piecewise concave, by comparing the optima within each segment, the kinks and the endpoints. Before doing so we drop any model that is dominated in both quality and price, which Lemma 2 below shows is never chosen.

With no cost of entering overage, the marginal value of a point lies between zero and the overage price at every state, which we use as a check on the solution. Solving and simulating one developer type for all ten months takes 0.27 core-seconds.

B.6 Additional Model Assumptions

Months are independent. The allowance resets on the first of each month and unused allowance does not carry over, so the only link between months in the model is the developer's fixed type. The model therefore rules out learning about new models, habit

formation and shifting work across months, and all persistence in behavior comes from fixed types and match values.

Activity is independent across days given the developer's activity probability, although activity is persistent in the data (Section 4). This matters little for monthly totals, but it can affect when within the month developers reach the allowance.

After the billing change, points per request equal the mean list-price spend per request. This ignores variation in tokens per request across requests to the same model, and it is measured on all Pro developers rather than on the estimation sample.

In the data, spending outside chosen models, such as code review, also draws down the allowance after the billing change (Section A.6). How the simulated allowance treats this spending determines how quickly simulated developers reach the allowance relative to the data.

We normalize quality by a softmax over a fixed universe of 48 models and set GPT-4o's fixed effect to zero, so menu changes leave quality unchanged. Because the universe is fixed for the paper, removing a provider in Section 8 does not mechanically raise the quality of the remaining models. Adding models to the universe, in contrast, would require re-estimation.

The solver allows any nonnegative continuation weight. The baseline restricts the weight to zero or one, following the support-point estimates of Nevo et al. (2016, Supplement S.1.4).

B.7 Proofs

Daily demand. With flow utility $u(c) = v(\gamma c)^{1-\beta}/(1-\beta) - \kappa c$ and a full marginal cost $\tau = \kappa + \lambda\alpha$ per request, where λ is the overage price when paying and the shadow price of a point in the free region, the first-order condition $v\gamma^{1-\beta}c^{-\beta} = \tau$ gives

$$c^* = \gamma^{1/\beta-1}\left(\frac{v}{\tau}\right)^{1/\beta}, \qquad \frac{\partial \ln c^*}{\partial \ln \tau} = -\frac{1}{\beta}. \tag{14}$$

The maximized daily value is $u(c^*) - \lambda\alpha c^* = \frac{\beta}{1-\beta}\tau c^*$, which is proportional to $(\gamma/\tau)^{(1-\beta)/\beta} v^{1/\beta}$.

Proof of the choice rule, Equation (4), at a shadow price λ that is the same for all models. Because $\beta \in (0,1)$, the maximized daily value is increasing in γ_m/τ_m for every value of the demand shock v. The best model therefore maximizes $\ln\gamma_m - \ln(\kappa + \lambda\alpha_m)$, and the ranking does not depend on the shock. □

Lemma 1. With no cost of entering overage, the shadow price of a point lies between zero and the overage price, equals the overage price on any day the developer pays for

overage, and is zero inside the allowance for a myopic developer. *Proof.* An additional point can always be bought at the overage price, so its value cannot exceed p, and free disposal bounds it below by zero. On a day with overage the marginal point is bought at p. A myopic developer places no weight on the continuation value, so the allowance is free to her until it runs out. □

Lemma 2. A model with weakly lower quality and weakly higher points per request than another model on the menu, with at least one inequality strict, is never chosen. *Proof.* The dominated model has a weakly lower γ_m and a weakly higher τ_m at every shadow price, so by Equation (4) it is never strictly preferred at a common shadow price; when the shadow price differs across models, the other model used with the same requests gives at least the same flow utility, uses no more points and pays no more overage, so the dominated model is again never strictly preferred, and ties occur with probability zero. □

Remark 1. Rescaling a developer's quality vector by a constant k is equivalent to shifting her log demand by $\frac{1-\beta}{\beta} \ln k$ for every model, as Equation (14) shows. The softmax normalization therefore leaves model rankings at a given shadow price unchanged.

Remark 2. Conditional on tastes, the scale of match values and the shadow price, the cross-section of model choices is a logit in log quality minus log full marginal cost, with scale equal to the match scale. Because match values are persistent, this logit describes the distribution of choices across developers, not day-to-day randomness for a given developer.

C Estimation Details

This appendix describes how we simulate the moments, defines each moment precisely, lists the parameters we fix, and describes the optimizer and the computation of standard errors.

C.1 Simulation of the Moments

We simulate 100,000 developers. Each simulated developer receives a fixed sequence of uniform draws: productivity, the scale of her daily demand shock, her activity probability, her provider and SWE-bench tastes, and one match value per model. Quantile transforms map these draws into parameters on bounded domains, so a change in a parameter moves every simulated developer smoothly.

Daily shocks are drawn in fixed monthly streams. The same simulated developer therefore faces the same shocks at every parameter value, and the objective varies only through the parameters, as is standard in simulated method of moments (McFadden, 1989; Pakes and Pollard, 1989).

We solve and simulate each developer under each of the three types: forward-looking, myopic and overage-averse. The three simulated paths enter every moment with weights equal to the type probabilities, which keeps the objective smooth in these probabilities instead of assigning each simulated developer a single type.

A simulated path enters the moments only if it has requests in every month, which mirrors the presence requirement of the balanced panel (Section 3.2). At the estimates, 64.7% of simulated developer-type paths satisfy it.

We treat the allowance as exhausted when cumulative spend reaches it up to a small numerical tolerance. Developers who stop exactly at the allowance are therefore classified as crossing it, as they are in the data.

C.2 Moment Definitions

The activity moments in Panel A of Table 3 are computed for each developer-month and averaged over the months before the billing change. A developer's mean log daily requests and the within-developer standard deviation of log daily requests are taken over her active days in the month, among developer-months with at least one active day; the share of inactive days is the share of days in the month on which she sends no request, over all developer-months.

Budget states are defined by cumulative spend at the start of the day. Days early in the allowance begin below half of it, and overage days begin at or above it. The price-response

moment is the mean, over developer-months with at least one active overage day and at least one active day early in the allowance, of the difference between the developer's mean log requests on her active overage days and on her active days early in the allowance. The May-to-June moment is the mean, over developers with at least one active overage day in both May and June 2026, of the change from May to June in the developer's mean log requests on her active overage days.

The cap-crossing moments are ratios of sums over developer-months in which the allowance runs out between day 4 and three days before the end of the month. They compare spend and activity on the day after the crossing with the day before. Restricting the crossing day keeps a full day on each side within the month.

Persistence uses each developer's most-used model in a month, defined as the strict argmax of her monthly requests. The persistence moment is the share of developers whose most-used model in May is still on the June menu and remains their most-used model in June, among developers with a strict most-used model in both months. The second persistence moment restricts this share to developers who send requests in overage in both May and June.

The launch moment uses the three Anthropic models that enter the Pro menu isolated from other menu changes: Claude Opus 4.5, Claude Opus 4.6 and Claude Sonnet 4.6. For each launch, a developer's previous model is her most-used model in the menu regime before the launch, and she adopts the new model if it is her most-used model in the regime in which it enters; developers with a tie or without requests in either regime are excluded. The moment is the adoption rate of the entering Anthropic model among developers whose previous model is not an Anthropic model, relative to the overall adoption rate, averaged over the three launches with weights equal to their adoption rates. SWE proximity compares the squared distance in SWE-bench score between adopters' previous model and the entrant with the same distance across all developers, pooled in the same way over all launches isolated from other menu changes.

Model shares are each model's share of choice requests on days on which it is on the Pro menu, averaged over those days.

C.3 Fixed and Pinned Parameters

In estimation we restrict curvature to lie between 0.5 and 0.95, writing $\beta_i = 0.5 + 0.45\, B_i$ with B_i drawn from a beta distribution whose mean, m_B, we estimate. We fix three kinds of parameters, listed in Table OA-8. The first are parameters that the moments do not separately identify, namely the dispersion of the time cost and of curvature. The second are normalizations: mean provider tastes, which the model fixed effects absorb, and the

Table OA-8: Parameters Fixed in Estimation

Parameter	Symbol	Value	Reason
Variance of log time cost	v_κ	0	Not separately identified
Concentration of curvature draw	c_B	10,000	Not separately identified
Variance of latent match scale	$v_{\bar{s}}$	0	Not separately identified
Mean Anthropic and SWE-bench tastes	$\bar{b}^A, \bar{b}^S$	0	Absorbed by model fixed effects
Fixed effect of GPT-4o	$\delta_{\text{GPT-4o}}$	0	Normalization
Fixed effects of the 9 models never on the Pro menu	δ_m	Start values	Enter only the normalization over $\mathcal{U}$
Correlation of demand level and activity	$\varrho_{\mu\pi}$	0	Restricted in baseline
Cost of entering overage, overage-averse type	$\bar{F}$	10^6	Implements overage aversion

Notes: This table lists the parameters held fixed in the baseline estimation and the reason for each. "Not separately identified" parameters are fixed at values that make the corresponding distribution degenerate. Fixed effects of models never offered on the Pro menu enter only the normalization of quality over the model universe $\mathcal{U}$ (Section 5.4).

GPT-4o fixed effect, set to zero. The third are devices, such as the cost of entering overage that implements overage aversion.

The fixed effects of models in the universe that are never offered on the Pro plan stay at their starting values. They do not change any developer's ranking of the models on the menu; through the normalization of quality, they affect only the level of her requests.

C.4 Optimization

We minimize the objective with a derivative-free trust-region Gauss–Newton method for least-squares problems (Cartis et al., 2019) in its mode for noisy objectives. We impose a box on each parameter, set the initial trust-region radius proportional to the width of the box, and allow a budget of 6,000 evaluations.

The search ends when the evaluation budget is exhausted rather than on a convergence test. The best point attains an objective of 0.0016.

We distribute the computation across parallel workers, each solving and simulating a block of developers. A full estimation takes 37 hours of wall time on a fleet of up to 199 workers with 16 cores each, about 119 thousand allocated core-hours; the diagnostics after the fit, the perturbation bank behind the fit tables and the standard-error grid, take another 34 hours of wall time and 66 thousand core-hours.

C.5 Standard Errors

The variance of the estimator combines sampling error and simulation error. We estimate the covariance of the data moments from 200 bootstrap draws over developers, keeping all

of a developer's days together in each draw, and the covariance of the simulated moments from 100 bootstrap draws over simulated developers at the estimates.

We compute the Jacobian of the simulated moments by a two-point finite difference with step $h = 10^{-3}$ in each parameter, holding the simulation draws fixed.

D Additional Tables and Figures

D.1 Additional Tables for Section 2

This appendix reports the exhibit behind Figure 1. Table OA-9 lists every model on the Copilot Pro menu with its SWE-bench Verified score, the first and last days it is on the menu, its price in points before the billing change and its price per request afterwards.

Table OA-9: Models on the Copilot Pro Menu

Model	SWE-bench	On the Pro menu		Points per	Price per request ($)		
		First day	Last day	request (Nov–May)	June	July	August
Anthropic							
Claude Haiku 4.5	73.3	Nov 1, 2025	Aug 31, 2026	0.33	$0.108	$0.116	$0.089
Claude Opus 4.5	80.9	Nov 24, 2025	Apr 20, 2026	1/3	–	–	–
Claude Opus 4.6	81.4	Feb 5, 2026	May 31, 2026	3	–	–	–
Claude Sonnet 3.5	49.0	Nov 1, 2025	Nov 10, 2025	1	–	–	–
Claude Sonnet 4	72.7	Nov 1, 2025	May 6, 2026	1	–	–	–
Claude Sonnet 4.5	77.2	Nov 1, 2025	Aug 31, 2026	1	$0.227	$0.224	$0.223
Claude Sonnet 4.6	80.2	Feb 17, 2026	Aug 31, 2026	1	$0.336	$0.356	$0.301
Claude Sonnet 5	79.6	Jun 1, 2026	Aug 31, 2026	–	$0.608	$0.417	$0.350
OpenAI							
GPT-4.1	54.6	Nov 1, 2025	Aug 31, 2026	0	–	$0.011	$0.011
GPT-4o	33.2	Nov 1, 2025	Aug 31, 2026	0	–	$0.028	$0.027
GPT-4o mini	8.7	Nov 1, 2025	Aug 31, 2026	0	–	$0.003	$0.002
GPT-5	72.8	Nov 1, 2025	Feb 18, 2026	1	–	–	–
GPT-5 Codex	74.5	Nov 1, 2025	Feb 18, 2026	1	–	–	–
GPT-5 mini	71.0	Nov 1, 2025	Aug 31, 2026	0	$0.019	$0.020	$0.021
GPT-5.1	76.3	Nov 13, 2025	Apr 15, 2026	1	–	–	–
GPT-5.1 Codex	73.7	Nov 13, 2025	Apr 1, 2026	1	–	–	–
GPT-5.1 Codex Max	77.9	Dec 4, 2025	Apr 5, 2026	1	–	–	–
GPT-5.1 Codex mini	71.3	Nov 13, 2025	Apr 1, 2026	0.33	–	–	–
GPT-5.2	80.0	Dec 11, 2025	May 31, 2026	1	–	–	–
GPT-5.2 Codex	80.0	Jan 14, 2026	May 31, 2026	1	–	–	–
GPT-5.3 Codex	78.0	Feb 11, 2026	Aug 31, 2026	1	$0.185	$0.174	$0.180
GPT-5.4	78.2	Mar 6, 2026	Aug 31, 2026	1	$0.220	$0.222	$0.210
GPT-5.4 mini	73.0	Mar 17, 2026	Aug 31, 2026	0.33	$0.068	$0.061	$0.059
GPT-5.6 Luna	71.0	Jul 9, 2026	Aug 31, 2026	–	–	$0.098	$0.031
GPT-5.6 Terra	78.2	Jul 9, 2026	Aug 31, 2026	–	–	$0.265	$0.193
Google							
Gemini 2.5 Pro	63.8	Nov 1, 2025	Jul 31, 2026	1	$0.059	$0.055	–
Gemini 3 Flash	78.0	Dec 17, 2025	Jul 31, 2026	0.33	$0.048	$0.046	–
Gemini 3 Pro	76.2	Nov 18, 2025	Mar 26, 2026	1	–	–	–
Gemini 3.1 Pro	80.6	Feb 19, 2026	Aug 31, 2026	1	$0.201	$0.165	$0.188
Gemini 3.5 Flash	79.3	May 19, 2026	Aug 31, 2026	14	$0.272	$0.228	$0.181
Gemini 3.6 Flash	79.3	Jul 21, 2026	Aug 31, 2026	–	–	$0.142	$0.137
Gemini 3.7 Flash	79.3	Aug 13, 2026	Aug 31, 2026	–	–	–	$0.133
xAI							
Grok 4.5	70.8	Jul 28, 2026	Aug 31, 2026	–	–	$0.352	$0.323
Grok 4.6	70.8	Aug 14, 2026	Aug 31, 2026	–	–	–	$0.421
Grok Code Fast 1	70.8	Nov 1, 2025	May 15, 2026	0/0.25	–	–	–
Moonshot							
Kimi K2.7 Code	67.5	Jul 1, 2026	Aug 31, 2026	–	–	$0.156	$0.145
Kimi K3	67.5	Aug 6, 2026	Aug 31, 2026	–	–	–	$0.349
Microsoft AI							
MAI-Code-1-Flash	71.6	Jun 1, 2026	Aug 31, 2026	–	$0.091	$0.058	$0.077
MAI-Code-1.1-Flash	72.6	Aug 11, 2026	Aug 31, 2026	–	–	–	$0.019

Notes: This table lists the models that appear on the Copilot Pro menu between November 2025 and August 2026, grouped by provider. SWE-bench is the model's score on SWE-bench Verified (Appendix A.7). First and last day are the first and last days on which the model is on the Pro menu within our sample. Points per request is the number of points one request uses under request-based billing (November 2025 to May 2026); zero marks an included model. Prices per request under token-based billing (June to August 2026) are the spend attributed to the model divided by its explicitly chosen requests (Appendix A.5). A dash indicates that the model is not on the Pro menu in that period. Two values of points per request separated by a slash record a price change, in chronological order.

Table OA-10: Use of Several Models and Providers

	Developer-days	Developer-months
	Panel A. Number of Models Used	
One	62.6	25.1
Two	27.4	25.6
Three	7.4	19.1
Four or more	2.6	30.1
Mean	1.51	2.95
	Panel B. Number of Providers Used	
One	76.5	44.0
Two	22.0	45.6
Three or more	1.5	10.4
Mean	1.25	1.68
	Panel C. Concentration and Persistence	
Share of requests to the developer's top model	88.5	76.6
Share of requests to the developer's top provider	93.7	87.8
Same top model as in the previous period	73.7	60.6
Same top provider as in the previous period	85.6	81.8
Observations	–	–

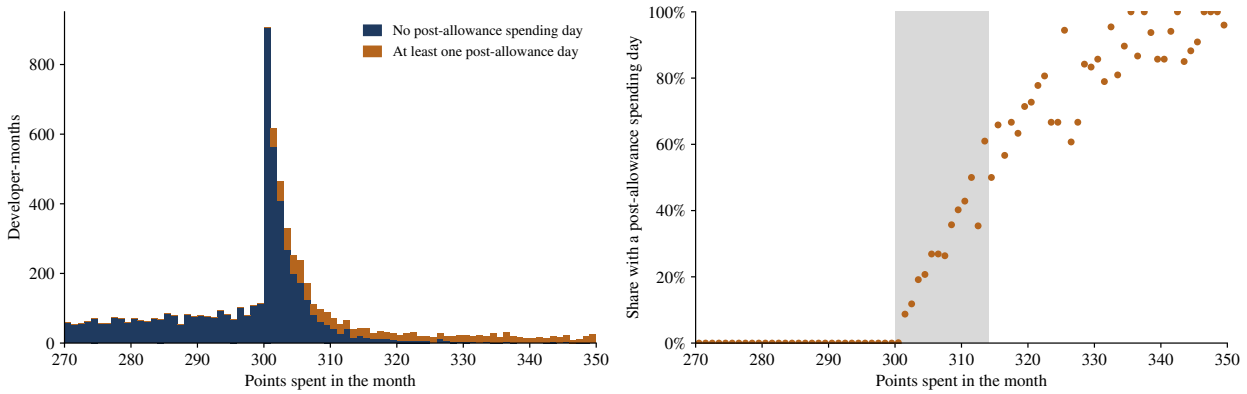
Notes: Panels A and B report the distribution of the number of models and of providers a developer uses, among developer-days (first column) and developer-months (second column) with at least two explicitly chosen requests in the estimation sample, counting explicitly chosen requests only; entries are percent except the Mean rows. Panel C reports the average share of a developer's requests that go to her top model and to her top provider within the day or month, and the share of developers whose top model (provider) is the same as on her previous active day (first column) or in the previous month (second column).

D.2 Additional Exhibits for Section 4

This appendix reports additional exhibits for Section 4.

Table OA-10 reports the distributions plotted in Figure 2, together with the concentration of requests on a developer's top model and provider and the persistence of the top model and provider.

Figure OA-1: Spending after the Allowance Is Used Up



Notes: This figure covers developer-months in the estimation sample between November 2025 and May 2026, when the allowance is 300 points, that end between 270 and 350 points. A post-allowance spending day is a day with positive spending that begins with at least 300 points already used. The left panel plots the number of developer-months by points spent in the month, split by whether the month has such a day. The right panel plots the share of developer-months that do; the shaded area marks 300 to 314 points, the range a developer can reach with one last request. The data do not order requests within a day, so requests made on the day the allowance runs out, after it runs out, are not counted.

D.3 Spending after the Allowance Is Used Up

Figure OA-1 checks that the mass of developer-months just above the allowance in Figure 5 reflects one last request rather than further spending once the allowance is used up. If developers stop once they are over the allowance, months that end just above 300 points should rarely contain a day that begins with the allowance already used, and this share should rise through the overage region. This is what the figure shows. About one in ten developer-months just above 300 points has such a day, the share reaches about one half at 314 points, and it approaches one well beyond that range.

D.4 Additional Tables for Section 5

Table OA-11 summarizes the three behavioral types of Section 5.3. For each type it gives the continuation weight, whether overage is feasible, and the price of a request before, at and after the point where the allowance runs out, under request-based billing before the billing change and under token-based billing after it.

D.5 Additional Tables for Section 7

This appendix reports two exhibits behind Sections 6 and 7. Table OA-12 reports each targeted moment in the data and in the model, with the parameters it mainly identifies (Section 6.2). Table OA-13 reports the estimated quality of every model on the Pro menu, plotted in Figure 8, and the willingness to pay for SWE-bench performance (Section 7.5).

Table OA-11: Behavioral Types

				Price of a request: points (before the billing change)			Price of a request: token-based billing (after the billing change)		
	Continuation weight	Overage feasible	Share	Inside allowance	At allowance	After allowance	Inside allowance	At allowance	After allowance
Forward-looking ($\rho_i = 1$)	Full	Yes	69.2 (6.5)	$\lambda_{it}\alpha_m$	$p\,\alpha_m$	$p\,\alpha_m$	$\lambda_{it}\alpha_m$	$p\,\alpha_m$	$p\,\alpha_m$
Myopic ($\rho_i = 0$)	None	Yes	11.7 (6.8)	Free	$p\,\alpha_m$	$p\,\alpha_m$	Free	$p\,\alpha_m$	$p\,\alpha_m$
Overage-averse	None	No	19.2 (7.5)	Free	Included only	Included only	Free	Unavailable	Unavailable

Notes: This table summarizes the three behavioral types of Section 5.3. The continuation weight ρ_i multiplies the change in the continuation value in the daily problem. Shares are the estimated type probabilities, p_ρ, $(1-p_\rho)(1-p_F)$ and $(1-p_\rho)p_F$, with standard errors in parentheses. Prices are per request: λ_{it} is the shadow price of a point, p the overage price per point and α_m the points per request of model m. "Free" means a zero shadow price; "Included only" means premium models are unavailable but models that use no points remain; "Unavailable" means no model can be used until the allowance resets.

D.6 Additional Tables for Section 8

This appendix reports the numbers behind Figure 9. Table OA-14 reports consumer surplus by month from November 2025 to August 2026 under the four menus of Figure 9, and the contributions of entry and of the pricing of the entrants.

Table OA-15 reports the full results of removing models and providers: losses in consumer surplus for each model and each provider and the destinations of displaced requests.

Table OA-12: Targeted Moments: Fit

Moment	Main parameters	Deviation	Data SE	Deviation / SE
	Panel A. Activity and Usage Volume			
Mean of developers' mean log daily requests (%)	m_μ	-0.72	0.58	-1.2
SD of developers' mean log daily requests (%)	v_μ	-0.4	0.4	-1.1
Mean of within-developer SD of log daily requests (%)	m_σ	-0.7	0.2	-3.1
SD of within-developer SD of log daily requests (%)	v_σ	-0.3	0.4	-0.8
Mean share of inactive days (pp)	m_π	0.11	0.17	0.6
SD of share of inactive days (%)	c_π	-0.4	0.4	-1.2
	Panel B. Price Response and Behavioral Types			
Log requests, overage minus early-allowance days (%)	m_κ, m_B	0.04	1.57	0.0
Change in log requests on overage days, May–June (%)	m_κ, m_B	0.10	8.42	0.0
Spend, day after vs. day before reaching the allowance (%)	p_ρ, p_F	2.1	3.8	0.6
Activity, day after vs. day before reaching the allowance (%)	p_F	-0.1	1.8	-0.0
	Panel C. Persistence and Launches			
Share keeping most-used model, May–June (pp)	$\bar{s}$, g	0.21	0.53	0.4
Same, among developers in overage in both months (pp)	$\bar{s}$, g	-1.45	3.24	-0.4
Anthropic launch adoption by other providers' users, relative (%)	v_A	-0.6	6.0	-0.1
SWE-bench proximity of new-model adopters, relative (%)	v_S	1.2	7.7	0.2
	Panel D. Model Request Shares (39 Moments)			
Mean absolute deviation of model shares (pp)	δ_m	0.26		
Largest absolute deviation of any model share (pp)	δ_m	0.60		

Notes: This table reports the 14 behavioral moments and a summary of the 39 model request shares used in estimation. "Main parameters" lists the parameters each moment mainly identifies (Section 6.3). We report deviations rather than levels, which are confidential. Deviation is the simulated moment at the estimates minus the data moment, in the unit given in each row (percentage points for shares and percent otherwise, with 100 times the difference in logs for moments measured in logs); Data SE is the bootstrap standard error of the data moment from 200 draws over developers, in the same unit. Data moments are computed on the balanced Pro panel; behavioral moments average November 2025 to May 2026 with equal weights, and the May-to-June rows compare the two months around the June 2026 billing change. Moment definitions are in Appendix C.2.

Table OA-13: Estimated Model Quality and Willingness to Pay for Intelligence

Panel A. Model Quality

Model	SWE-bench	Months on menu	Price per request: Pre (points)	Price per request: Post ($)	Quality δ_m	(SE)	Premium over GPT-4o ($)
Anthropic							
Claude Sonnet 4.6	80.2	7	1.00	0.331	0.486	(0.039)	0.122
Claude Sonnet 5	79.6	2	–	0.384	0.479	(0.038)	0.120
Claude Opus 4.6	81.4	4	3.00	–	0.437	(0.033)	0.107
Claude Sonnet 4.5	77.2	10	1.00	0.225	0.433	(0.032)	0.105
Claude Opus 4.5	80.9	6	2.84	–	0.348	(0.032)	0.081
Claude Haiku 4.5	73.3	10	0.33	0.104	0.245	(0.033)	0.054
Claude Sonnet 4	72.7	7	1.00	–	0.231	(0.034)	0.051
Claude Sonnet 3.5	49.0	1	1.00	–	-0.877	(0.328)	-0.114
OpenAI							
GPT-5.6 Terra	78.2	2	–	0.224	0.402	(0.035)	0.096
GPT-5.4	78.2	6	1.00	0.218	0.401	(0.035)	0.096
GPT-5.3-Codex	78.0	7	1.00	0.179	0.373	(0.028)	0.088
GPT-4.1	54.6	9	0.00	0.011	0.361	(0.034)	0.084
GPT-5.6 Luna	71.0	2	–	0.059	0.304	(0.032)	0.069
GPT-5 mini	71.0	10	0.00	0.020	0.292	(0.034)	0.066
GPT-5.2	80.0	6	1.00	–	0.266	(0.034)	0.059
GPT-4o mini	8.7	9	0.00	0.002	0.229	(0.046)	0.050
GPT-5	72.8	4	1.00	–	0.204	(0.034)	0.044
GPT-5.2-Codex	80.0	5	1.00	–	0.166	(0.034)	0.035
GPT-5.4 mini	73.0	6	0.33	0.062	0.049	(0.055)	0.010
GPT-4o	33.2	9	0.00	0.027	0.000	—	0.000
GPT-5.1-Codex-Max	77.9	5	1.00	–	-0.162	(0.090)	-0.029
GPT-5.1-Codex	73.7	6	1.00	–	-0.168	(0.096)	-0.030
GPT-5.1	76.3	6	1.00	–	-0.215	(0.095)	-0.038
GPT-5-Codex	74.5	4	1.00	–	-0.240	(0.102)	-0.041
GPT-5.1-Codex mini	71.3	6	0.33	–	-0.488	(0.143)	-0.075
Google							
Gemini 3 Pro	76.2	5	1.00	–	0.187	(0.034)	0.040
Gemini 3.7 Flash	79.3	1	–	0.133	0.139	(0.061)	0.029
Gemini 3.1 Pro	80.6	7	1.00	0.184	-0.002	(0.079)	-0.000
Gemini 3.6 Flash	79.3	2	–	0.138	-0.254	(0.186)	-0.044
Gemini 2.5 Pro	63.8	9	1.00	0.057	-0.333	(0.116)	-0.055
Gemini 3.5 Flash	79.3	4	14.00	0.226	-0.614	(0.363)	-0.089
Gemini 3 Flash	78.0	8	0.33	0.047	-0.692	(0.218)	-0.097
Other providers							
Grok 4.6 (xAI)	70.8	1	–	0.421	0.322	(0.031)	0.074
Kimi K3 (Moonshot)	67.5	1	–	0.349	0.243	(0.052)	0.053
Kimi K2.7 Code (Moonshot)	67.5	2	–	0.150	-0.003	(0.078)	-0.001
MAI Code 1.1 Flash (Microsoft)	72.6	1	–	0.019	-0.145	(0.163)	-0.026
Grok Code Fast 1 (xAI)	70.8	7	0.11	–	-0.342	(0.156)	-0.056
Grok 4.5 (xAI)	70.8	2	–	0.326	-0.626	(0.462)	-0.091
MAI Code 1 Flash (Microsoft)	71.6	3	–	0.075	-0.794	(0.225)	-0.107
SD of δ_m within providers					0.337		
SD of provider means of δ_m					0.191		

Panel B. Willingness to Pay for Intelligence

	(1) Across models	(2) Within providers
SWE-bench Verified score (points)	0.0020	0.0015
	(0.0011)	(0.0008)
Provider indicators	No	Yes
R^2	0.05	0.46
Correlation of δ_m and SWE-bench	0.22	
Models	39	39
Value of one SWE-bench point		
Reduction in full cost of a request (%)	0.20	0.15
Per developer-month ($), mean	0.052	0.039
Per developer-month ($), median	0.031	0.022
Per developer-month ($), 10th percentile	-0.070	-0.083
Per developer-month ($), 90th percentile	0.203	0.185

Notes: Panel A: one row per model offered on the Pro menu, grouped by provider and sorted by estimated quality within provider. Prices before the billing change are points per request and prices after the billing change the average list price of a request in dollars; blank if not on the menu. δ_m is the model fixed effect, a log price equivalent relative to GPT-4o. The premium over GPT-4o is the additional amount per request a developer with average tastes and match values would pay at overage prices. Panel B projects δ_m on the SWE-bench Verified score across all models (1) and within providers (2), weighting by request share, and converts the slope into the value of one point using each simulated developer's usage and SWE-bench taste. Standard errors in parentheses.

Table OA-14: Consumer Surplus over Time

	Consumer surplus per developer-month ($)				Contribution ($)	
	Baseline	+ Price changes	+ Entry at quality-implied prices	+ Entry at factual prices	Entry	Entrant pricing
Nov.	31.02	31.02	31.02	31.02	–	–
Dec.	30.98	30.98	31.46	31.45	0.49 (0.06)	-0.01 (0.02)
Jan.	30.93	30.93	31.53	31.56	0.60 (0.06)	0.03 (0.01)
Feb.	31.16	31.16	31.43	32.02	0.26 (0.08)	0.59 (0.02)
Mar.	30.99	30.99	31.34	32.00	0.35 (0.08)	0.66 (0.02)
Apr.	31.05	31.05	31.44	32.19	0.39 (0.09)	0.75 (0.02)
May	30.94	30.93	31.31	32.06	0.38 (0.09)	0.75 (0.02)
Jun.	30.98	30.97	31.32	32.22	0.34 (0.09)	0.90 (0.02)
Jul.	30.94	30.93	31.85	32.59	0.92 (0.11)	0.73 (0.03)
Aug.	31.01	31.02	32.02	33.19	1.01 (0.11)	1.17 (0.03)

Notes: Each entry is consumer surplus per developer-month (per 30 days), in dollars: the realized monthly payoff net of the time cost of requests and of overage, before the $10 fee, for the same panel of estimation-sample developers as in Figure 9. Every month uses the token-based tariff in force from June 2026, and preferences, match values and behavioral types are held at their estimated values. "Baseline" is the November 2025 menu at November prices. "+ Price changes" moves the prices of the November models to their values in that month. "+ Entry at quality-implied prices" adds every model that had entered the Pro menu by that month, priced as the November relation between price and the Artificial Analysis Intelligence Index would price it, and keeps the models that left the menu. "+ Entry at factual prices" prices the entrants at the prices Copilot set. "Entry" is the third column minus the second, and "Entrant pricing" is the fourth column minus the third; standard errors from a paired developer bootstrap in parentheses.

Table OA-15: The Value of Each Model and Each Provider: Full Results

	Loss in consumer surplus		Displaced requests go to (%)			
	$ per dev.-month	%	Anthropic	OpenAI	Google and Other	Not made
			Panel A. Removing One Model			
GPT-4o mini	0.98 (0.04)	2.99 (0.13)	22.0	74.1	1.3	2.7
GPT-4.1	0.67 (0.05)	2.05 (0.14)	17.2	77.7	2.6	2.6
GPT-5.6 Luna	0.44 (0.05)	1.33 (0.14)	21.1	71.2	4.9	2.7
GPT-5 mini	0.39 (0.03)	1.18 (0.08)	17.7	74.9	4.4	3.0
GPT-4o	0.24 (0.03)	0.72 (0.10)	16.4	67.4	11.9	4.3
Claude Haiku 4.5	0.23 (0.02)	0.70 (0.07)	28.4	62.6	7.1	1.9
GPT-5.4 mini	0.16 (0.02)	0.50 (0.06)	23.9	60.5	12.2	3.3
Claude Sonnet 4.5	0.16 (0.04)	0.49 (0.12)	57.0	40.1	2.3	0.7
GPT-5.3 Codex	0.15 (0.02)	0.47 (0.07)	32.1	62.6	4.3	0.9
GPT-5.6 Terra	0.15 (0.02)	0.45 (0.07)	28.3	68.7	2.3	0.7
Kimi K2.7 Code	0.14 (0.03)	0.42 (0.10)	19.1	62.5	17.6	0.8
GPT-5.4	0.12 (0.03)	0.36 (0.08)	29.4	67.4	2.6	0.6
MAI Code 1.1 Flash	0.10 (0.02)	0.31 (0.07)	14.2	71.6	12.1	2.1
Claude Sonnet 4.6	0.09 (0.03)	0.29 (0.09)	79.9	19.0	0.6	0.5
Gemini 3.6 Flash	0.09 (0.03)	0.27 (0.10)	24.3	60.3	14.0	1.4
Gemini 3.1 Pro	0.08 (0.02)	0.25 (0.06)	23.7	64.1	11.8	0.4
Gemini 3.7 Flash	0.06 (0.01)	0.18 (0.03)	25.0	64.2	10.4	0.4
MAI Code 1 Flash	0.06 (0.02)	0.18 (0.05)	18.7	57.0	22.9	1.4
Gemini 3.5 Flash	0.05 (0.03)	0.15 (0.08)	26.1	44.9	28.6	0.4
Grok 4.5	0.02 (0.02)	0.07 (0.06)	13.3	56.1	30.6	0.0
Kimi K3	0.02 (0.02)	0.07 (0.05)	27.7	64.3	7.8	0.1
Grok 4.6	0.01 (0.01)	0.04 (0.02)	23.1	68.2	8.7	0.0
Claude Sonnet 5	-0.03 (0.02)	-0.10 (0.07)	74.0	24.5	1.5	0.1
			Panel B. Removing One Provider			
Anthropic	0.32 (0.06)	0.98 (0.19)	0.0	97.8	2.2	0.1
OpenAI	5.58 (0.10)	17.02 (0.24)	82.2	0.0	13.3	4.5
Google	0.29 (0.05)	0.88 (0.14)	25.6	64.8	9.0	0.6
Other	0.38 (0.05)	1.15 (0.16)	22.7	69.7	7.1	0.5

Notes: Counterfactual menus remove the listed model (Panel A) or all of a provider's models (Panel B) from the Copilot Pro menu of August 2026, without renormalizing tastes; losses are per developer-month for the same panel of estimation-sample developers under the factual tariff and prices, with paired developer-bootstrap standard errors in parentheses. Panel A lists every model on the August 2026 Pro menu in decreasing order of the loss in consumer surplus. Losses are in dollars per developer-month and in percent of factual consumer surplus; destination shares divide displaced requests among the remaining providers and requests not made.