Self-preferencing, Quality Provision, and Welfare in Mobile Application Markets*

Xuan Teng†
University of Munich

September 2023

Abstract

Platforms often display their products ahead of third-party products in search. Is this due to consumers preferring platform-owned products or platforms engaging in self-preferencing by biasing search towards their own products? What are the welfare implications? I develop a structural model of mobile application markets to identify self-preferencing and quantify its welfare effects, taking into account third-party developers' quality adjustment. A new dataset on app downloads, prices, characteristics, and search rankings is used to estimate the model. Estimates indicate self-preferencing. Simulations show higher consumer welfare and third-party profits without self-preferencing.

Keywords: competition policy, platform design, consumer search, endogenous product choice.
JEL Classification: D12, D83, L13, L86.

*This paper is based on various chapters of my dissertation. I am indebted to my advisors, Ying Fan, Zach Brown, Jagadeesh Sivadasan, and Justin Huang for their continual guidance, support, and encouragement. I also thank Chenyu Yang, Benjamin Leyden, José L. Moraga, Ao Wang, and the participants at various seminars for helpful comments and discussions. I gratefully acknowledge the financial support from the Department of Economics Research Grant at the University of Michigan in obtaining the data needed for this research project. Support by Deutsche Forschungsgemeinschaft through CRC TRR 190 (project number 280092119) is gratefully acknowledged. All errors are mine.

†xuan.teng@econ.lmu.de. Department of Economics, University of Munich, Akademiestr. 1/III, Munich 80799, Germany.

1 Introduction

In many digital platforms, search algorithms shape the competition landscape due to the tension between numerous products and limited consumer attention. Thus, when platforms play a dual role of first-party seller and intermediary between third-party sellers and consumers, people worry about biased recommendations in favor of platform-owned products (Feasey and Krämer, 2019). In other words, platforms may be "self-preferencing" in search results (Crémer, de Montjoye and Schweitzer, 2019). In fact, in many important markets, the press has reported that platform-owned products dominate top positions.[1] In light of the potential damage to competition and consumer welfare, regulation against self-preferencing is under discussion in the US and Europe.[2]

Two questions are at the core of dealing with self-preferencing. First, do platforms use self-preferencing? Despite widespread media coverage of first-party dominance, platforms deny engaging in self-preferencing. Instead, they argue that platform-owned products receive higher rankings because consumers prefer them over third-party products. Second, what is the welfare effect of self-preferencing? The resulting competitive advantage enjoyed by platform-owned products reduces both pre-innovation and post-innovation rents of existing third-party producers (Aghion et al., 2005). Hence, its equilibrium impacts on innovation and welfare remain ambiguous.

To address these questions, I develop a structural model for the mobile application markets on the US Apple App Store. This model incorporates consumer search, potential self-preferencing, and quality-upgrading competition among developers. I estimate the model with a newly compiled dataset from multiple data sources, covering market-level information on consumer search and downloads, installation prices, product characteristics, and search rankings of popular apps on the US Apple App Store between April 2018 and February 2020. To deal with endogeneity concerns on installation price, update frequency, average rating, and search ranking, I exploit exogenous choice set variations to construct instrumental variables. After detecting self-preferencing in estimation, I simulate counterfactuals without self-preferencing to quantify the welfare effects.

[1] Example markets include restaurant search engines, e-commerce, and mobile applications. See the stories covered by (1) Dougherty, Conor. 2017. “Inside Yelp’s Six-Year Grudge Against Google.” The New York Times, July 1, (2) Mattioli, Dana. 2019. "Amazon Changed Search Algorithm in Ways That Boost Its Own Products" The Wall Street Journal, Sept. 16, and (3) Mickle, Tripp. 2019. "Apple Dominates App Store Search Results, Thwarting Competitors", The Wall Street Journal, July 23, respectively.

[2] For example, American Choice and Innovation Online Act in the US and Digital Markets Act in Europe.

I motivate the structural model with empirical evidence from an unexpected search algorithm change in Apple App Store in July 2019. Specifically, Apple twisted a feature in the search algorithm so that fewer of its own apps appear in top search results.[3] Applying a difference-in-differences (DiD) approach, I find that the algorithm change leads to higher rankings, increased downloads, and more updates for third-party apps in categories with Apple's apps, compared to third-party apps in categories without Apple's apps. However, I find no significant effects on other app characteristics, such as installation price and average rating. These findings drive my focus on developers' choice of update frequency as the primary strategic response to search algorithm changes in the structural model, taking installation price as given.

The structural estimation results confirm two primary factors contributing to first-party dominance in search results: consumer preference and platform self-preferencing. First, demand estimation reveals a preference for Apple's apps over third-party apps, indicating higher unobserved quality of Apple's apps on average. Second, search ranking estimation shows that Apple's ownership significantly increases the probability of receiving top rankings, conditional on the revealed app quality and other observed ranking shifters. Thus, while iPhone users prefer Apple's apps, their preference is not strong enough to fully explain the prominence of platform-owned products on the Apple App Store.

Based on the estimated model parameters, I conduct simulations that eliminate self-preferencing on the Apple App Store during June and July of 2019. The simulation results show that, without self-preferencing, the average update frequency increases by 1.86 percent in an average market. However, this effect varies significantly across apps. While some apps experience an increase in update frequency by up to 65.61 percent update frequency, others see a decrease of up to 21.42 percent. A correlation analysis reveals that the decrease in update frequency is associated with the business-stealing effect of boosted-up third parties on other third-party products, as well as the cannibalization concerns of multiple-app developers.

As for the welfare effect, I find that consumer surplus increases by 0.28 percent, and third-party profits increase by 0.66 percent without self-preferencing. In equilibrium, the search ranking and downloads of an average third-party app rise up by 1.22 percent and 1.95 percent, respectively. While the downloads of an average Apple's app decrease by 22.56 percent, total downloads increase by 1.47 percent due to the much smaller number

[3] Nicas, Jack and Collins, Keith. 2019. "How Apple's Apps Topped Rivals in the App Store It Controls". The New York Times, Sept. 9.

of first-party apps than third-party apps. Therefore, while self-preferencing is detrimental to consumer welfare, third-party innovation and profits, the magnitude of these effects is limited within the empirical context of this study. In scenarios where consumers' preference for platform-owned products is weaker, and thus self-preferencing is stronger given similar observed first-party dominance, the welfare effect of self-preferencing is likely to be larger.

This article contributes to three branches of literature. First, it belongs to the broad literature on information frictions and competition, dated back to Stigler (1961) and Diamond (1971).[4] Many papers in the literature study the effects of search costs on price and its dispersion. Examples include Hortaçsu and Syverson (2004), Brown (2017), Brown (2019), Dinerstein et al. (2018), and Salz (2020).[5] An emerging group of literature studies the effects of search costs on quality. The theoretical papers in this group find ambiguous relationships between search cost and product quality (Wolinsky, 2005, Fishman and Levy, 2015, and Moraga-González and Sun, 2023). Empirically, Ershov (2020) shows that reduced discovery costs for game apps lead to lower entrant quality. However, empirical studies on the quality provision of existing products are rare. This article adds to the literature by showing that the quality effect largely contributes to the welfare effect of self-preferencing.

A handful of recent papers study the issue of self-preferencing. Empirically, Chen and Tsai (2019) and Farronato, Fradkin and MacKay (2023) show evidence of self-preferencing on Amazon using product-level data and micro-level consumer search data, respectively. To quantify the welfare effects of self-preferencing on Amazon, Lam (2021) and Lee and Musolff (2021) take the structural approach and incorporate directed consumer search and firm entry, respectively. Complementing these papers, I look at a large marketplace for digital products and naturally incorporate quality upgrading. Furthermore, Lam (2021) and Lee and Musolff (2021) quantify the welfare effects based on

[4] See Goldfarb and Tucker (2019) for a review on the papers studying the effects of reduced search costs in digital economies.

[5] Closely related to this paper, Dinerstein et al. (2018) studies the trade-off between promoting price competition and displaying the most desired product to consumers on eBay. They estimate a parameter governing the emphasis on lower prices in the search design and simulate the effects of a redesign with an increase in the parameter. In similar spirits, I estimate a parameter governing the emphasis on platform ownership and simulate the effects of reducing the parameter to zero to study the effect of self-preferencing. However, while I do not have browsing data as available in Dinerstein et al. (2018), I build on the method in Moraga-González, Sándor and Wildenbeest (2023*a*) to estimate search cost distributions and consumer preference with market-level data.

Amazon's discoverability advantage that is unconditional on consumers' unobserved preferences for Amazon's products. Both studies find that such first-party advantage improves consumer welfare at least in the short run. In contrast, this article finds self-preferencing conditional on consumers' preference for platform-owned products and its negative effects on consumer welfare.[6]

Second, this article builds on the literature on endogenous product choice (Crawford, 2012).[7] Papers in the literature model product characteristics as either discrete product choices (e.g., Draganska, Mazzeo and Seim, 2009) or continuous characteristic choices (e.g., Fan, 2013). This article adapts the framework to product characteristics featuring corner solutions, like update frequency, modeling it as outcomes of two-stage discrete-continuous choices. Furthermore, many papers in the literature study markets without information frictions. I extend the framework to a frictional context where consumers incur search costs to visit products.

Finally, this article is also related to the emerging literature on mobile applications. Examples include Ghose and Han (2014), Leyden (2019), Ershov (2020), Allon et al. (2021), Singh, Hosanagar and Nevo (2021), and Janssen et al. (2021). This article complements existing papers by examining the welfare effects and contributing factors of first-party dominance, which is a salient but understudied feature of the mobile application industry.

The rest of the paper is organized as follows. Section 2 describes the data. Section 3 provides background on the search algorithm change in the U.S. market on the Apple App Store and presents descriptive evidence. Section 4 describes the structural model of demand, search ranking, and update competition in mobile application markets. Section 5 describes the estimation procedure and presents the estimation results. Section 6 presents counterfactual simulations. Section 7 concludes.

[6]There is a group of theoretical papers studying self-preferencing or "own-content bias" (De Corniere and Taylor, 2019, Hagiu, Teh and Wright, 2022, Zennyo, 2022), who find mixed welfare implications of self-preferencing under different market conditions.

[7]Examples include Mazzeo (2002), Seim (2006), Augereau, Greenstein and Rysman (2006), Draganska, Mazzeo and Seim (2009), Chu (2010), Crawford and Yurukoglu (2012), Fan (2013), Eizenberg (2014), Berry, Eizenberg and Waldfogel (2016), Wollmann (2018), Crawford, Shcherbakov and Shum (2019), Hristakeva (2022), Fan and Yang (2020), and Fan and Yang (2022).

2 Data

For this study, I compile a new dataset from various sources covering the US Apple App Store's mobile application markets in 38 categories from April 2018 to February 2020.[8] The dataset contains aggregated information on consumer search and purchase. Regarding consumer purchase, I observe downloads, installation prices, and total revenues from installations and in-app purchases. One download represents the initial installation by a unique consumer. However, due to the confidential nature of actual download and revenue data, I obtain estimated figures from AppTweak.[9] To calculate market shares, I then obtain the number of US iPhone users from Comscore. Regarding consumer search, I observe the ratio of app downloads over the number of consumers who encounter the app during search, commonly known as the conversion rate. Although the conversion rate data is only available at type/category/month level, with type being free or paid apps, the data is uniquely informative of consumer choice among searched apps.

The dataset contains two search ranking variables, aggregated from various search results information. Given an app and a keyword, I observe the position of the app in the search result of the keyword.[10] For each app, I focus on the relevant keywords where the app shows up in the top 50 positions.[11] Then, as the main search ranking variable, I calculate the app's weighted average position across these relevant keywords. The weight is an Apple-constructed search volume index, reflecting how many consumers search for the keyword. When I do not observe any keyword where the app shows up in the top-50 positions, I use an indicator to document the case.[12]

The dataset also comprises app characteristics, including ratings, age, file size, in-app purchase availability, the number of screenshots, description length, and update frequency. Note that updates differ in content and importance, with major updates often surpassing bug-fix updates in improving apps and typically entailing longer release notes. Thus, I assign higher weights to updates with longer release notes and calculate the

[8] Table D.1 lists the 22 non-game categories and 16 game categories in the sample.

[9] Figure E.1 shows the fitness of AppTweak's estimated downloads based on their confidential data on actual downloads.

[10] The search results on Apple's App Store are non-personalized, alleviating the misspecification concerns for demand estimation when personalized search results are typically unobservable.

[11] Figure E.2 plots residual downloads against the granular position data observed at app/keyword/day level. It shows a fast decline of downloads as position increases from 1 to 50. In the data, total downloads across the top-50 apps account for 60% total downloads across the top-500 apps in search results on average.

[12] Specifically, I find the 60 keywords that are mostly used in each category and check whether a given app in a given category shows up in the top-50 positions in any of the 60 keywords.

weighted number of updates in a given month. Recognizing the heterogeneity across categories, I normalize the weights within each category by comparing release notes of app versions in the same category.

I find popular independent apps based on top charts and annual downloads in 2019. Then, I combine them with Apple's apps to construct the sample. In the end, there are 47,977 app/month pairs in the sample, 17.5% of which operate in multiple categories, leading to 56,570 observations at app/category/month level. Appendix A provides a detailed explanation of the sample selection process, data sources, and variable definitions. Table 1 reports the summary statistics of the main variables used in the empirical analysis.

Table 1: Summary Statistics

Variable	Mean	Median	SD	Min	Max	Obs
Downloads (million)	0.06	0.01	0.23	0	7.00	56,570[a]
Revenues (million)	0.37	0.01	1.81	0	56.51	56,570
Conversion Rates (%)	0.05	0.04	0.06	0.00	0.45	1,337[b]
Type (paid app?)	0.49	0.00	0.50	0	1	3,110[c]
Apple	0.01	0.00	0.08	0	1	3,110
Offer In-app Purchase?	0.67	1.00	0.47	0	1	3,110
Installation Price|Paid ($)	4.15	2.99	4.69	0	99.99	23,884[a]
Top 50 in Search Results?	0.62	1.00	0.49	0	1	56,570
Search Ranking	14.51	12.52	14.63	0	50	56,570
Update Frequency	0.68	0.25	1.00	0	11	56,570
Average Rating	4.40	4.57	0.55	1	5	56,570
Age (month)	51.16	49.00	32.66	1	140	56,570
File Size (MB)	225.69	115.20	411.01	0.73	4096	56,570
#Screenshots	5.54	5.00	1.96	0	10	56,570
Description Length (character)	2212.50	2180	1035.02	0	3998	56,570
Market Share (%)	0.06	0.01	0.21	0	6.36	56,570
#iPhone users (million)	105.90	107.58	5.31	98.55	117.72	23[d]

[a]These observations are at app/category/month level.
[b]These observations are at type/category/month level.
[c]These observations are at app level.
[d]These observations are at month level.

3 Descriptive Evidence

3.1 Search Algorithm Change on Apple App Store

In July 2019, Apple launched a search algorithm change on Apple App Store that reduced the dominance of Apple's apps in search results. Specifically, the algorithm change

"tweaked a feature of the app store search engine that sometimes grouped apps by maker" so that "Apple apps would no longer look as if they were receiving special treatment".[13] There was no official report on why Apple changed the search algorithm. Given the timing, the algorithm change is likely due to increasing antitrust challenges against Apple; however, most of the challenges were about commission instead of self-preferencing. For example, in May 2019, the Supreme Court voted 5 to 4 to allow a antitrust lawsuit brought by Apple App Store customers against Apple regarding Apple using monopoly power to raise the prices of iPhone apps.[14] Furthermore, this algorithm change was firstly reported in September of 2019 by New York Times, two months later than the launch. Therefore, I argue that the algorithm change was unanticipated by independent developers and consumers.[15]

Figure 1 compares the search rankings of Apple's apps and independent apps and covey multiple messages. First, on average, Apple's apps enjoy higher search rankings than independent apps between October 2018 and February 2020. Second, the search ranking of Apple's apps become sharply lower after the search algorithm change in July 2019. Third, although the new search algorithm tweaked a feature that groups apps by makers, multiple-app developers do not see a different pattern than single-app developers on average.[16] The figure confirms that the algorithm change reduced the dominance of Apple's apps. To focus on the effect of the algorithm change, I examine the sample period between June and November of 2019 in a difference-in-differences (DiD) analysis.

3.2 Difference-in-Differences Analysis

The algorithm change provides a quasi-natural experiment to study the effect of platform-owned products' dominance in search results on third-party products. To that end, I use independent apps competing with Apple's apps in the categories as the treatment group and independent apps in categories that do not contain Apple's apps as the control group.[17]

[13] Nicas, Jack and Collins, Keith. 2019. "How Apple's Apps Topped Rivals in the App Store It Controls". The New York Times, Sept. 9.

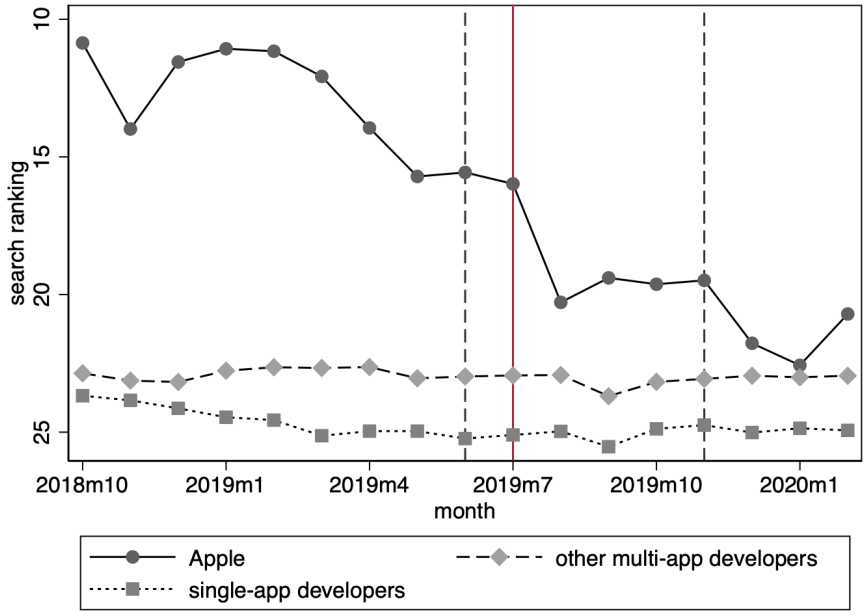
[14] Liptak, Adam and Nicas, Jack. 2019. "Supreme Court Allows Antitrust Lawsuit Against Apple to Proceed". The New York Times, May. 13.

[15] Even if independent developers and consumers expect a similar search algorithm change to come; however, I argue that they would not know exactly when the search algorithm change would come.

[16] In the data, 16.83% developer/month pairs possess more than one app.

[17] There are 16 non-game categories containing Apple's apps while 22 non-game and game categories do not. Table D.6 shows the summary statistics on observations in categories with and without Apple's apps, before and after the search algorithm change.

Figure 1: Average Search Ranking of Apple's Apps around July 2019



Following the literature, I use the two-way fixed effects specification:

$$y_{jgt} = \beta(AppleCompetitor_{jg} \times Post_t) + \lambda_{jg} + \lambda_t + \nu_{jgt} \tag{1}$$

where $AppleCompetitior_{jg}$ indicates whether independent app j is in a category g with Apple's Apps; $Post_t$ indicates if month t is after July 2019. I include app-category fixed effects, λ_{jg}, to capture time-invariant confounders and month-fixed effects, λ_t, to control for time-varying factors. I consider a variety of outcome variables, y_{jgt}, including search ranking, downloads, conversion rate, update frequency, price, average rating, and file size. When the outcome variable is conversion rate, j denotes app type rather than app, in order to align with the observation level. The coefficient on the interaction term, β, captures the average treatment effect of the search algorithm change.

Table 2 presents the estimated average treatment effects from Equation 1. After the search algorithm change that reduces the dominance of Apple's apps, there are significant increases in independent app's search ranking (3.6%), downloads (22.1%), and update frequency (2.1%) of independent apps in categories that contain Apple's apps, compared to independent apps in categories that do not; while the other outcome variables of interest

Table 2: Effects of the Search Algorithm Change on Independent Apps: Difference-in-Differences Estimates

Outcome Variable	ATE	SE	Obs	Adj. R^2	FE	Mean Level
log(Search Ranking)	−0.04	0.01	11,642	0.86	A	24.17
log(Downloads)	0.22	0.02	20,423	0.95	A	0.06
log(Conversion Rates)	0.09	0.07	330	0.95	B	0.07
log(1+Update Frequency)	0.02	0.01	20,423	0.62	A	0.63
log(1+Price) ($\times$10)	0.01	0.04	20,423	0.98	A	1.91
log(Avg.Rating) ($\times$10)	0.02	0.02	20,423	0.94	A	4.37
log(File Size) ($\times$10)	−0.07	0.04	20,423	0.99	A	216.30

Notes: ATE is the estimate of β in Equation (1) for the outcome variable on the row. SE are robust standard errors. Search ranking is observed for apps that have ranked in top-50 search results of any popular keyword in a given category and month. Conversion rates are observed at type/category/month level. The other outcome variables are observed at app/category/month level. FE: (A) app/category-fixed effects, month-fixed effects; (B) type/category-fixed effects, month-fixed effects, where type indicates paid or free apps. For exposition, the last three outcome variables are multiplied by 10 during estimation. Mean levels are *not* in logarithms nor enlarged.

remain unaffected.[18] It implies that third-party developers' main strategic response to the search algorithm change is update frequency, rather than installation price, file size, or unobserved efforts to improve average ratings. The result motivates my focus on update frequency in the structural model.

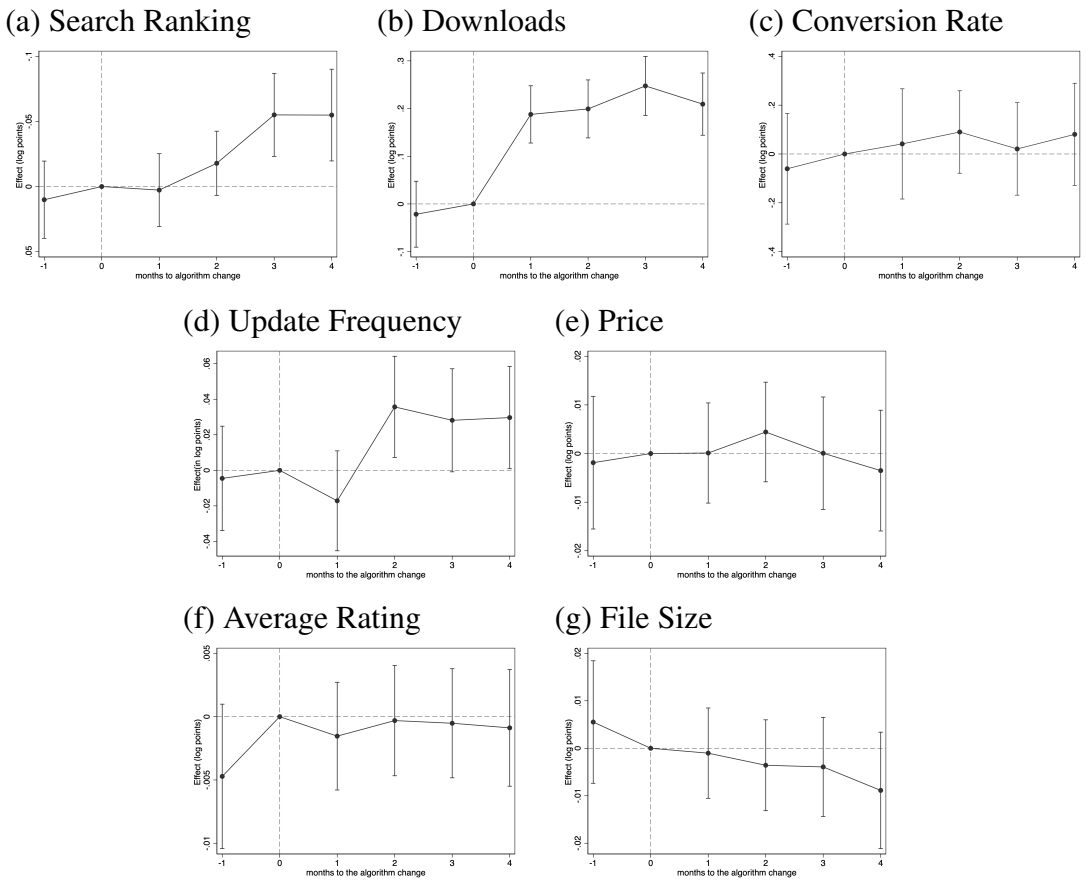
Figure 2 presents the pre-trends between the treatment and control groups.[19] In the period before the search algorithm change, there is no significant effect for independent apps in categories that contain Apple's apps relative to independent apps in categories that do not on any of the outcome variables of interests. This provides evidence that the independent apps competing with Apple's apps had similar trends in the preperiod as the independent apps that do not compete with Apple's apps, supporting the common trends assumption. Figure E.5 shows that the results are robust to multiple preperiods when examining half-month treatment effects.

Figure 2, panels (a) and (b), demonstrate that the downloads effect precedes the ranking effect, which needs further explanation. It indicates that the algorithm change initially drops the ranking of certain independent apps while boosting others. Furthermore, the dropped independent apps turn out to experience fewer download losses compared to the

[18] Within the sample period, developers may choose to reset ratings after updates. Appendix A.2 argues that such reset behaviors rarely happen in the data.

[19] The specification used for Figure 2 is $y_{jgt} = \sum_\tau \beta_\tau (AppleCompetitor_{jg} \times 1\{t = \tau\}) + \tilde{\lambda}_{jg} + \tilde{\lambda}_t + \tilde{\nu}_{jgt}$, where $\tau \in \{-1, 1, 2, 3, 4\}$. The interaction with July 2019 is omitted, because the search algorithm change is launched on July 22, near the end of the month.

Figure 2: Effect of the Search Algorithm Change on Independent Apps, by Month



Notes. The charts present point estimates for each month using the difference-in-differences specification as specified in Section 3.2. The omitted period is the month at the end of which the search algorithm change was launched. Error bars indicate 95% confidence interval using standard errors robust to heteroscedasticity.

download gains of the boosted independent apps. Later on, the average independent apps rise in search ranking due to the algorithm change.[20] Thus, the algorithm change goes beyond eliminating self-preferencing, if any; otherwise, there would be no decline in any independent app. Therefore, one should be cautious to interpret the algorithm change as evidence for self-preferencing and its effect.

The panel (c) of Figure 2 shows a zero conversion rate effect, indicating that the increased downloads are proportional to the increased consumer views on average. In similar spirits of the random rankings exploited in Ursu (2018), the result implies that the exogenous rise in search ranking does not cause consumers to perceive the boosted-up apps as better than before. Otherwise, conditional on seeing an app, consumers should be more likely to download the app after the algorithm change, leading to a positive conversion rate effect. It motivates me to assume that search ranking only affects demand through affecting search costs in the structural model.

Figure 2, panels (d) to (g), reflect the stickiness of installation price, average rating, and file size, in contrast with the sensitivity of update frequency with respect to search algorithms. Simiarly, Table D.4 compares the within-app variation of these app characteristics in the data, confirming that update frequency varies more within apps than how the other studied app characteristics do. Figure E.3 illustrates the stickiness of installation prices over time.[21] While my sample focuses on popular apps and thus is short of representative industry-level entry observations, Figure E.6 shows that there was no significant effect on the entry of independent apps due to the search algorithm change.

4 Model

4.1 Demand

To study self-preferencing in search, it is essential to incorporate the effect of search ranking on demand. To that end, I use a random-coefficient discrete choice model augmented with consumer search to describe app downloads. The model is based on Moraga-González, Sándor and Wildenbeest (2023*a*), and I tailor it to the mobile application mar-

[20] Figure E.5b shows that the ranking effect is robust to controlling for one-period lagged downloads, supporting the direct (though slow) ranking effect of the search algorithm change.

[21] In-app prices for purchase and subscription may be more sensitive to search algorithms; however, data on these prices are typically unavailable.

kets.

In the model, a market is a category/month pair. While it is likely that consumers download multiple new apps in a month, multiple downloads are less likely within categories. For example, in 18 months in the data, the total downloads of all apps are larger than the number of iPhone users. However, within all category/month pairs in the data, the total downloads are smaller than 25% of the number of iPhone users. Thus, I assume that a consumer may download apps from multiple categories in a month and remain agnostic about category choices, but within a category, the consumer downloads no more than one app in a month

In each market, consumers incur *search costs* to learn about the *indirect utilities* from downloading apps, and choose the app with the highest indirect utility among the searched apps.

Specifically, by downloading the app j in category g and month t, a consumer i receives the following indirect utility:

$$u_{ijgt} = \alpha p_{jt} + \boldsymbol{x}_{jt}\boldsymbol{\beta} + \tilde{\gamma}_i a_{jgt} + \xi_{jgt} + \varepsilon_{ijgt} \tag{2}$$

where p_{jt} is the installation price, $\boldsymbol{x}_{jt}$ is a high-dimensional vector of observable app features, a_{jgt} is $log(1 + \text{update frequency})$, and ε_{ijgt} is an idiosyncratic match value, i.i.d., and follows the Type I extreme value distribution.[22] The outside option is not downloading any app, which gives the indirect utility ε_{i0gt}.[23]

Previous updates may continuously contribute to future app quality (Leyden, 2019). The term ξ_{jgt} is a partially unobservable (to researchers) category/year-specific taste for app j. It follows an AR(1) process as below.

$$\xi_{jgt} = \rho\xi_{jgt-1} + \gamma a_{jgt} + \eta_{jgt}, \qquad \mathbb{E}[\xi_{jgt-1}\eta_{jgt}] = 0 \tag{3}$$

[22]The vector of app features, $\boldsymbol{x}_{jt}$, includes i) indicators of Apple ownership, paid installation, offering in-app-purchase, game apps, and whether the category g contains pre-installed apps; ii) the logarithms of age (in months), file size (in MB), one plus the length of description (in characters); iii) average rating, and month indicators to capture month-fixed effects omitting the first month. Note that $\boldsymbol{x}_{jt}$ does not contain a_{jgt}, as update frequency changes across categories within app/month pairs due to the category-specific weights explained in Section 2.

[23]The outside option lumps all cases where consumers do not download any app in a given category in a month, which are i) using no app; ii) using one or multiple previously downloaded apps; iii) using pre-installed apps. The likelihood of the second case may increase over time as a consumer develops the habit of using a previously downloaded app. Such dynamics in downloads are flexibly captured by the month-fixed effects in $(\boldsymbol{x}_{jt}\boldsymbol{\beta})$. Note that pre-installed apps serve as a part of the outside option; thus, they are not included in the estimation sample for the empirical model.

where η_{jgt} is an unobservable taste. I interpret γ as the mean static effect of updates on indirect utility. And $\tilde{\gamma}_i$ in Equation (2) is i.i.d. and follows the normal distribution $\mathcal{N}(0,\sigma)$, capturing consumers' temporary and heterogeneous tastes over updates.[24]

Two key assumptions come along with Equation (2). First, the equation does not include search ranking, meaning that search ranking does *not* affect product value – once searched by a consumer, an app's search ranking does not matter anymore for the consumer to download the app or not. This assumption is motivated by the insignificant effect of the search algorithm change on conversion rates, as discussed in Section 3.2. It helps with the separate identification of consumer tastes and search costs.[25] Second, following the literature, consumers are at a relatively late stage in search – before search, they know $(p_{jt}, \boldsymbol{x}_{jt}, \xi_{jgt})$.[26] Consumers only search to know the idiosyncratic match value ε_{ijgt}, by incurring a search cost c_{ijgt}.

Note that search cost c_{ijgt} may change across search rankings and across consumers. For example, it may include the cost of scrolling down along the search rankings to see the app, clicking on the app, and digesting the information on the app page. Meanwhile, for a given app, some consumers might know the match value from friends before search, and thus have a zero search cost, while others do not. To model the heterogeneity, I use the following cumulative distribution function for c_{ijgt}.

$$
\begin{aligned}
F^{c}_{jgt}(c|\mu_{jgt}) &= \frac{1-\exp(-\exp(-H_0^{-1}(c)-\mu_{jgt}))}{1-\exp(-\exp(-H_0^{-1}(c)))} \\
\mu_{jgt}(\boldsymbol{\lambda}) &= \log\left[1+\exp(\lambda_1 E_{jgt}+\lambda_2 \log(ranking_{jgt}))\right]
\end{aligned}
\tag{4}
$$

where μ_{jgt} is the location parameter of the distribution, shifted by two ranking variables: i) E_{jgt}, an indicator of appearing in the top-50 search results; ii) $ranking_{jgt}$, the average

[24] For example, while some consumers may like the availability of new features, others may dislike the inconvenience to adjust to the new features.

[25] In principle, without the assumption, consumer preference and search costs can be separately identified by matching the observed conversion rates with model predictions. However, the conversion rates are observed at a much coarser level (type/category/month level) compared to the main data (app/category/month level). Thus, I go with this simplification assumption. One main resulting caveat is that the constant search cost that does not change with search rankings is not disentangled from the constant term in indirect utility. See Moraga-González, Sándor and Wildenbeest (2023*b*) for a discussion on how the separate identification affects estimation and policy predictions in a simultaneous consumer search framework.

[26] As a weaker assumption, consumers have rational expectations of the observable app characteristics. Appendix A.3 shows descriptive patterns that are consistent with the weaker assumption. When consumers have imperfect information on observable app characteristics, the estimated demand parameters with respect to these characteristics are biased towards zero as if consumers do not care about these characteristics.

ranking conditional on $E_{jgt} = 1$.[27] To interpret the coefficients, note that the distribution has a mass point at 0, i.e., $F^c_{jgt}(0|\mu_{jgt}) = \exp(-\mu_{jgt})$. It shows that as μ_{jgt}, the *search cost parameter*, increases, fewer consumers have low search costs. Thus, if higher ranking leads to lower search costs, then one expects that $\lambda_1 < 0$ and $\lambda_2 > 0$.

Following Proposition 1 in Moraga-González, Sándor and Wildenbeest (2023*a*), when consumers search according to the optimal sequential search model in the spirit of Weitzman (1979), the distributional assumption in Equation (4) rationalizes a closed-form choice probability in the Berry, Levinsohn and Pakes (1995) (BLP) framework.[28] Specifically, the probability of consumer i downloading app j in category g and month t is given by

$$s_{ijgt}(\boldsymbol{\theta}^D, \tilde{\gamma}_i) = \frac{\exp(\delta_{jgt} + \tilde{\gamma}_i \tilde{a}_{jgt} - \mu_{jgt} + I_g \mu_{0gt})}{1 + \sum_{l \in \mathscr{J}_{gt}} \exp(\delta_{lgt} + \tilde{\gamma}_i \tilde{a}_{lgt} - \mu_{lgt} + I_g \mu_{0gt})} \tag{5}$$

where $\boldsymbol{\theta}^D = (\alpha, \boldsymbol{\beta}, \rho, \gamma, \sigma, \boldsymbol{\lambda})$, the relative mean utility $\delta_{jgt} = \alpha p_{jt} + \boldsymbol{x}_{jt}\boldsymbol{\beta} + \xi_{jgt}$, $\mathscr{J}_{gt}$ is the set of apps in the market, and I_g indicates categories with pre-installed apps.[29]

Then, I aggregate the consumer-level choice probability to market level and obtain the following model-implied market share (s_{jgt}) and downloads (Q_{jgt}) of app j in category g and month t:

$$\begin{aligned} s_{jgt}(\boldsymbol{\theta}^D, \sigma) &= \int s_{ijgt}(\boldsymbol{\theta}^D, \tilde{\gamma}_i) dF_{\tilde{\gamma}}(\tilde{\gamma}_i), \qquad \tilde{\gamma}_i \sim \mathscr{N}(0, \sigma). \\ Q_{jgt}(\boldsymbol{\theta}^D, \sigma) &= M_t \cdot s_{jgt}(\boldsymbol{\theta}^D, \sigma) \end{aligned} \tag{6}$$

where M_t is the number of iPhone users in month t.

[27] $H_0(r) = Euler\ Constant - r + \int_{\exp(-r)}^{\infty} \frac{\exp(-t)}{t} dt$.

[28] During the optimal sequential search, consumers visit apps in the descending order of reservation values, stop searching when the highest realized utility so far is above the reservation value of the next product to be searched, and choose the product with the highest realized utility. To clarify, the descending order of reservation values does not necessarily coincide with the ascending order of search rankings. This is realistically possible thanks to the fact that the variable $ranking_{jgt}$ is a weighted average search ranking across different keywords. For example, product A might have rankings 1 and 4 in two equally-weighted keywords respectively; while product B has rankings 9 and 1 in the same two keywords respectively. Thus, product A's average search ranking is 3, while product B's is 5. A consumer may have a higher reservation value for product B such that s/he firstly search for the second keyword and sees product B first, and find himself/herself unsatisfied. Then, s/he searches for the first keyword and sees product A next.

[29] The term $I_g \mu_{0gt}$ roots in the assumption that consumers may incur search costs to learn ε_{0gt} when there are pre-installed apps in the category, as pre-installed apps are lumped into outside options and they show up in search results. In reality, such search costs may happen when consumers do not know which apps are pre-installed, where are the pre-installed apps on the smartphone, or whether they have deleted the pre-installed apps. When a category does not have pre-installed apps, the outside option is either not using any app or using a previously-downloaded app; thus, consumers incur zero search cost to learn ε_{0gt}.

The demand estimation equation is given by

$$\tilde{\delta}_{jgt}(s_{gt};\sigma) = \underbrace{\alpha p_{jt} + x_{jt}\beta + \rho\xi_{jgt-1} + \gamma a_{jgt} + \eta_{jgt}}_{\textit{relative-mean utility}(\delta_{jgt})} - \underbrace{(\mu_{jgt}(\lambda) - I_g\mu_{0gt}(\lambda))}_{\textit{relative search cost parameter}} \tag{7}$$

where the variable $\tilde{\delta}_{jgt}$ is the output of the BLP inversion (Berry, Levinsohn and Pakes, 1995), which I interpret as a search-augmented relative mean utility. On the right-hand side is the traditional relative mean utility (δ_{jgt}) minus the relative search cost parameter between the choice j and the outside option, after replacing the ξ_{jgt} in δ_{jgt} with the AR(1) process in Equation (3). The estimation is based on the General-Methods-of-Moments (GMM) with iterated guesses of the non-linear parameters (σ,ρ,λ).[30] The moments are between the error term η_{jgt} and the instruments to be specified in Section 5.1.

Now I define app j's quality index, $\check{\delta}_{jgt}$ in the following equation:

$$\check{\delta}_{jgt} := \delta_{jgt} - \alpha p_{jt} - \beta^{paid} paid_j \tag{8}$$

where $paid_j$, as an entry in x_{jt}, indicates paid installation. Thus, the app quality index is the part of the relative mean utility that is independent of installation payment. Importantly, since x_{jt} includes the indicator of Apple ownership, $\check{\delta}_{jgt}$ captures consumers' preference for Apple's apps. It will serve as a key control variable in the following search ranking model to identify self-preferencing.

4.2 Search Ranking

Given that Apple's search ranking algorithm is proprietary, I apply a rank-ordered logistic regression model (Beggs, Cardell and Hausman, 1981) to approximate the algorithm. The model predicts the probability of a given ordering of products based on latent *ranking scores*.

Specifically, I model the ranking score of an app j in category g and month t as

$$score_{jgt} = \theta_{1tk(g)} Apple_j + P(\check{\delta}_{jgt}) \cdot \theta_2 + z^s_{jgt} \cdot \vartheta_{k(g)} + e_{jgt} \tag{9}$$

where θ_{1tk} is a flexible self-preferencing parameter that changes across months t and

[30] Appendix B.2 details the steps to transform Equation (7) into a linear equation given a guess of (σ,ρ,λ).

across category group k, $Apple_j$ indicates Apple ownership, $P(\check{\delta}_{jgt})$ is a third-order polynomial function of the app quality index $\check{\delta}_{jgt}$, z^s_{jgt} is a vector of observed search ranking shifters, and e_{jgt} is independently and identically drawn from the Type-I Extreme Value distribution.

Equation (9) gives the technical definition of self-preferencing in this paper: $\theta_{1tk} > 0$, i.e., a positive effect of platform ownership on search ranking, conditional on product quality and other non-discriminatory ranking shifters. Because the constructed app quality includes consumers' preference for platform-owned apps, the model can directly test a typical defense for platforms' dominance in top positions, namely "Our products are ranked higher because they are preferred by consumers". In particular, if consumers' preference is strong enough to justify Apple apps' higher ranking, then the coefficient on app quality should be significantly positive and the coefficient on Apple's ownership should be insignificantly different from zero.

Let $\boldsymbol{y}$ denote an ordering of products, where $\boldsymbol{y}(k)$ is the k-th product in the ordering. The conditional probability of $\boldsymbol{y}$ is given by

$$\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}^s] = p_{\boldsymbol{y}(1)} \cdot \frac{p_{\boldsymbol{y}(2)}}{1-p_{\boldsymbol{y}(1)}} \cdot \frac{p_{\boldsymbol{y}(3)}}{1-p_{\boldsymbol{y}(1)}-p_{\boldsymbol{y}(2)}} \cdot \cdots \cdot \frac{p_{\boldsymbol{y}(J-1)}}{p_{\boldsymbol{y}(J-1)}+p_{\boldsymbol{y}(J)}} \cdot \frac{p_{\boldsymbol{y}(J)}}{p_{\boldsymbol{y}(J)}} \tag{10}$$

where $\boldsymbol{x}^s = (\boldsymbol{Apple}, \check{\boldsymbol{\delta}}, \boldsymbol{z}^s)$, $p_j = \exp(\boldsymbol{x}^s_j \cdot \boldsymbol{\theta}^s) / (\sum_{l \in \mathscr{J}} \exp(\boldsymbol{x}^s_l \cdot \boldsymbol{\theta}^s))$, $\boldsymbol{\theta}^s = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\vartheta})$, and J is the number of products in the market. I estimate the model with Maximum-Likelihood-Estimation (MLE).[31]

4.3 Supply

I develop a supply model to describe how independent developers choose update frequency, taking other app features as given.[32] This model can predict how updates and thus app quality of independent apps change with search ranking algorithms.

The model is a static two-stage game of quality-upgrading competition among multiple-product firms (app developers). In the first stage, each firm chooses its *update portfolio* – the set of apps to be updated, by incurring a fixed cost of update that does not change with the content of update. In the second stage, after observing idiosyncratic shocks to

[31] Appendix B.3 provides the conditional log-likelihood function, as well as the list of variables in z^s_{jgt}.

[32] The event study on the search algorithm change in Section 3, along with the stickiness of installation price discussed in Section 2, motivates my focus on update frequency.

the marginal cost of update, the firm chooses how much to update for each app in the update portfolio. Altogether, the two stages determine the value of update frequency: zero or positive (first stage); if positive, how large (second stage).[33] Throughout the game, developers form beliefs on search rankings based on the previously specified search ranking model. In equilibrium, the beliefs are self-fulfilling.[34] I describe the two stages in backward order. For exposition, I omit the market index gt in this section.

Stage 2 - Update Frequency. App developers have three revenue sources: i) installation, ii) in-app purchase and subscription, and iii) in-app advertising. The first revenue source generates $p_j Q_j$. For the second revenue source, while I do not observe prices for in-app purchase and subscription, I fit a fixed effects model for the observed revenues, using downloads and update frequency as main revenue shifters. For the detailed specification of the revenue model, please see Appendix B.4. I estimate the revenue model separately from the supply model, obtain the model-fitted revenues $R(Q_j, a_j)$, and treat $R(\cdot)$ as known functions.

For the third source of revenue, I do not have data on in-app advertising. Thus, I estimate an in-app-advertising profit function together with the marginal cost of update. Specifically, I assume that the variable in-app-advertising profit is a simple quadratic function of downloads as below.

$$F(Q_j;\psi) = \psi_1 Q_j + \psi_2 Q_j^2 \tag{11}$$

And I specify the marginal cost of update at a_j in the following equation.

$$g'(a_j, \omega_j; \phi) = \phi_1 a_j + z_j^g \phi + \omega_j \tag{12}$$

[33] The timing assumption implies that the fixed cost of update serves a role of commitment: even if the second-stage marginal cost shock turns out to be so large that the ex-post total cost of update exceeds the benefit, the developer still upgrades the app to the level that balances the marginal cost and marginal benefit. For example, based on past experience, the developer calculates the fixed cost, expects an update to be profitable, and incurs some sunk costs (e.g. planning) as a commitment device. As the developer works on the update, unexpected bugs appear and require extra effort, i.e., a positive idiosyncratic shock to the marginal cost. Then, the developer might end up with a smaller and even ex-post unprofitable update than expected.

[34] Due to the continual effect of update on quality, developers may consider the impacts of current update on future installations. However, modeling the dynamic and competitive portfolio choice of multiple-product firms is out of the scope of this paper. Leyden (2019) and Allon et al. (2021) develop dynamic structural models for app updates, treating developers as single-product firms and abstracting away from market competition.

where the vector z_j^g contains app age and month-fixed effects, and ω_j is an idiosyncratic marginal cost shock. I assume that ω_j's are revealed at the beginning of the second stage. Since whether an app is updated or not is determined in the first stage, updated apps are not selected based on ω_j's.

Developers cannot perfectly predict rankings or downloads when choosing update frequency. Ex-post, given an ordering $\boldsymbol{y}$, the variable profit of an independent developer f is given by

$$\begin{aligned}\pi_f^I(\boldsymbol{y},\boldsymbol{a},\boldsymbol{\omega}_f) = \sum_{j\in\mathscr{J}_f} & 0.7p_jQ_j(\boldsymbol{y},\boldsymbol{a}) + 0.7R(Q_j(\boldsymbol{y},\boldsymbol{a}),a_j) \\ & + F(Q_j(\boldsymbol{y},\boldsymbol{a});\psi) - g(a_j,\omega_j;\phi)\end{aligned} \tag{13}$$

where 0.7 comes from the 30% commission rate charged by Apple, $\mathscr{J}_f$ is the set of apps owned by the developer f, the vector $\boldsymbol{\omega}_f = (\omega_j : j \in \mathscr{J}_f)$, and the function $g(a_j,\omega_j;\phi) = \frac{\phi_1}{2}a_j^2 + (z_j^g\phi + \omega_j)a_j$. I assume zero marginal distributional cost for mobile applications to serve additional consumers. The only uncertainty about downloads conditional on updates comes from search rankings.

Ex-ante, developers form beliefs on $\boldsymbol{y}$ based on the ranking probability in Equation (10). However, the original belief space, denoted as $\mathscr{B}$, is infeasibly large for computation because the number of possible orderings increases factorially with the number of products.[35] To deal with the computational challenge, I construct a heuristic belief space, denoted as $\mathscr{B}_{\boldsymbol{a}}$, by assuming that developers only consider some most likely orderings when choosing updates. In particular, they always consider the most likely ordering, i.e., the descending order of ranking scores, and only consider up to two sequential swaps of products in the most likely ordering.[36] By construction, depending on the number of products in the market, the heuristic belief space has at most 141 possible orderings,

[35] In the data, an average market has 65 products. Thus, it is infeasible to allow developers to consider all possible orderings of search rankings. It is likewise generally infeasible to closely approximate $\mathscr{B}$. For example, I test with a market consisting of 10 products in the data. To reach $\sum_{\boldsymbol{y}\in\mathscr{B}}\mathbb{P}[\boldsymbol{y}] \geq 0.2$, I find that one needs to evaluate at least 1% of the elements in $\mathscr{B}$, which corresponds to 36,288 orderings.

[36] For example, suppose the most likely ranking is $(1,2,3,4,5)$, one example for the first swap is $(2,1,3,4,5)$, built on which, one example for the second swap is $(2,1,4,3,5)$. Appendix B.4.2 provides the full list of swaps that the heuristic belief space contains and explains the mathematical motivation. Furthermore, the construction implies that the heuristic belief space may change with updates: as one app updates more, its ranking score may rise, and the most likely ordering may change accordingly, which in turn changes the heuristic belief space. I use the subscript $\boldsymbol{a}$ in $\mathscr{B}_{\boldsymbol{a}}$ to denote this relationship.

which turns out to capture 8.5 out of the top-10 most likely orderings on average.[37]

On the heuristic belief space $\mathscr{B}_{\boldsymbol{a}}$, a well-defined probability measure is the following conditional probability for any ordering $\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}$:

$$\tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] := \mathbb{P}[\boldsymbol{y}|\boldsymbol{a}] \Big/ \left(\sum_{\boldsymbol{y}' \in \mathscr{B}_{\boldsymbol{a}}} \mathbb{P}[\boldsymbol{y}'|\boldsymbol{a}] \right) \tag{14}$$

where $\mathbb{P}[\boldsymbol{y}|\boldsymbol{a}]$ is the ranking probability defined in Equation (10) with a highlight on $\boldsymbol{a}$ in $\boldsymbol{x}^s$. Then, the ex-ante variable profit of a developer f is given by

$$\pi_f^{II}(\boldsymbol{a}, \boldsymbol{\omega}_f) = \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \pi_f^{I}(\boldsymbol{y}, \boldsymbol{a}, \boldsymbol{\omega}_f) \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] \tag{15}$$

In equilibrium, the following necessary conditions hold true: marginal ex-ante variable profit of update equals zero, given other developers' decisions. Let D_j indicate whether app j is updated. Then the necessary conditions are given by,

$$MB_j(a_j, \psi) = \phi_1 a_j + \boldsymbol{z}_j^g \phi + \omega_j, \qquad \forall j \text{ s.t. } D_j = 1 \tag{16}$$

where the marginal benefit of update, $MB_j(a_j, \psi)$, is given by

$$\begin{aligned}
MB_j(a_j, \psi) &\equiv MB_j^{[0]} + MB_j^{[1]} \psi_1 + MB_j^{[2]} \psi_2, \\
MB_j^{[0]} &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} \left(0.7 p_l + 0.7 \frac{\partial R(Q_l, a_l)}{\partial Q_l} \right) \frac{\partial Q_l}{\partial a_j} + 0.7 \frac{\partial R(Q_l, a_l)}{\partial a_l} \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] \\
&\quad + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} (0.7 p_l Q_l + 0.7 R_l) \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j} \\
MB_j^{[1]} &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} \frac{\partial Q_l}{\partial a_j} \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} Q_l \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j} \\
MB_j^{[2]} &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} 2 Q_l \frac{\partial Q_l}{\partial a_j} \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} Q_l^2 \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j}
\end{aligned}$$

where $MB_j^{[0]}$ is the marginal benefit of update from the first two revenue sources, and the sum of $MB_j^{[1]} \psi_1$ and $MB_j^{[2]} \psi_2$ is the marginal benefit of update from in-app advertising. These marginal benefits consist of two parts: the direct effect of update on downloads

[37] In contrast, $|\mathscr{B}_{\boldsymbol{a}}| / |\mathscr{B}| < 0.03$ for markets with more than 5 products in the data.

while holding ranking probability fixed implied from the demand model (e.g., $\frac{\partial Q_l}{\partial a_j}$), and the indirect effect of the update on downloads through affecting ranking probabilities implied from the search ranking model (e.g., $\frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j}$).[38] I take Equation (16) to GMM with the instruments for update frequency detailed in Appendix B.1.

Stage 1 - Update Portfolio. In the first stage, developers do not observe ω_j's. Denote the second-stage equilibrium update frequency as $a^+(D,\boldsymbol{\omega}^+)$, where $D=(D_1,\cdots,D_J)$ and $\boldsymbol{\omega}^+=(\omega_j : D_j=1)$. The objective function of developer f in the first stage is given by,

$$\pi_f^{III}(\boldsymbol{D}) := \mathbb{E}_{\boldsymbol{\omega}^+}[\pi_f^{II}(\boldsymbol{a}^+(D,\boldsymbol{\omega}^+),\boldsymbol{\omega}_f^+)|\boldsymbol{D}] - \sum_{j\in\mathscr{J}_f} C_j D_j \tag{17}$$

where C_j is the fixed cost of update for app j.

In Nash Equilibrium, each developer chooses his/her update portfolio, denoted as $D_f=(D_1,\cdots,D_{J_f})$, that maximizes the objective function in Equation (17), given others' update portfolios D_{-f}. Following Fan and Yang (2020), the equilibrium condition implies no profitable deviation from the observed update decisions, which enables researchers to back out bounds on C_j without specifying any equilibrium selection rule for potential multiple equilibria. Specifically, when an app j is not updated, a necessary inequality is that the expected increase in ex-ante variable profits cannot offset the fixed costs, i.e., $\forall D_j=0$,

$$\begin{aligned} C_j \geq & \mathbb{E}_{\boldsymbol{\omega}^+}[\pi_{f(j)}^{II}(\boldsymbol{a}^+(1,\boldsymbol{D}_{-j},\boldsymbol{\omega}^+),\boldsymbol{\omega}_{f(j)}^+)|1,\boldsymbol{D}_{-j}] \\ & - \mathbb{E}_{\boldsymbol{\omega}^+}[\pi_{f(j)}^{II}(\boldsymbol{a}^+(0,\boldsymbol{D}_{-j},\boldsymbol{\omega}^+),\boldsymbol{\omega}_{f(j)}^+)|0,\boldsymbol{D}_{-j}] \end{aligned} \tag{18}$$

Meanwhile, when an app j is updated in category g and month t, a necessary inequality is that the expected increase in ex-ante variable profits can offset the fixed costs, i.e., $\forall D_{jgt}=1$,

$$\begin{aligned} C_j \leq & \mathbb{E}_{\boldsymbol{\omega}^+}[\pi_{f(j)}^{II}(\boldsymbol{a}^+(1,\boldsymbol{D}_{-j},\boldsymbol{\omega}^+),\boldsymbol{\omega}_{f(j)}^+)|1,\boldsymbol{D}_{-j}] \\ & - \mathbb{E}_{\boldsymbol{\omega}^+}[\pi_{f(j)}^{II}(\boldsymbol{a}^+(0,\boldsymbol{D}_{-j},\boldsymbol{\omega}^+),\boldsymbol{\omega}_{f(j)}^+)|0,\boldsymbol{D}_{-j}] \end{aligned} \tag{19}$$

During estimation, I use Equation (18) to yield the lower bounds ($\underline{C}_j$'s) on the fixed costs of update for apps that are not updated, and Equation (19) to obtain the upper bounds ($\bar{C}_j$'s) on the fixed costs of update for apps that are updated.

[38] I numerically compute $\frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j}$ based on the estimated search ranking model with a perturbation step of `1e-4`. For more technical details, see Appendix B.4.3.

5 Estimation

5.1 Estimation Procedure

I estimate the three pieces of the empirical model separately. I start with demand estimation in Equation (7). Then I carry the estimated app quality to the search ranking probability in Equation (10). Lastly, in the supply estimation, I treat the estimated demand function and ranking probability function as known functions, and take Equation (16) and inequalities (18) and (19) to data. The estimation sample consists of non-preinstalled Apple's apps and popular third-party apps on Apple's App Store, while pre-installed apps are lumped into outside options.

The identification of app demand parameters is similar to that in Berry, Levinsohn and Pakes (1995). However, unlike BLP, apart from endogenous price (p_{jt}), I consider three endogenous product characteristics: update frequency (a_{jgt}), average rating (x_{2jgt}), and search ranking ($E_{jgt}, ranking_{jgt}$). I therefore exploit two different sources of exogenous variation in the above endogenous variables.

First, I construct indirect markup shifters based on the endogenous characteristics of products in *other* markets that are owned by *other* developers in the same market. Given a developer, his/her products' characteristics in other markets serve as a proxy for the developer's productivity (Hausman, 1996, Nevo, 2000).[39] Thus, these indirect markup shifters capture "how productive are my competitors" as opposed to "how attractive are my competitors" in the classical BLP instruments. They indirectly affect the markup of a product by affecting the product's rivals' strategic decisions on price, update, and ratings.[40]

Second, I exploit two exogenous search ranking shifters: i) the unanticipated search algorithm change; ii) the match between app titles and keywords. For the latter, I assume that app title does not affect indirect utility. It shifts demand only by affecting ranking probability as described in Equation (10). For the other excluded instruments and first-stage regression results, please see Appendix B.1.

The identification of the self-preferencing parameters relies on consistent estimates of

[39] Table D.5 provides summary statistics on the source of variation for the Hausman IV and the indirect market shifters in the data.

[40] On top of the assumption for exclusion restrictions of the cost proxies, the exclusion of the indirect markup shifters additionally depends on a timing assumption that the unobserved demand shocks are realized after app entry, so that developers cannot perfectly foresee their competitors' η_{jgt} when entering the market.

app quality. Given that, categories with and without Apple's apps also help with identification. How much higher quality contributes to higher search ranking in categories without Apple's apps reveals the effects of app quality. Then, how much Apple ownership reverses this relationship between quality and search ranking identifies the extent of platform self-preferencing.

In the last step of estimation, evaluating inequalities (18) and (19) involves solving the second-stage game for each drawn vector of marginal cost shocks, as well as computing market shares for each possible ordering of products in the heuristic belief space. Following the literature, to alleviate the computational burden, I restrict the sample for computing the cost bounds with two steps. First, I focus on relevant markets, namely, the categories with Apple's apps during the difference-in-differences sample period, since only these categories may have a different counterfactual equilibrium than the status quo. Second, I focus on the top 5 developers in each category and only allow these top 5 developers to change their updates.[41] As a result, there are 506 upper bounds and 176 lower bounds to be computed.

5.2 Estimates of Demand

Table 3 reports the estimates for the parameters of the demand model. Regarding consumer preference, it shows that an average consumer significantly prefers apps with higher update frequency, Apple ownership, higher average rating, and lower installation price, among other app features. Quantitatively, the results indicate that an average consumer is willing to pay $5.2 more for downloading an Apple's app developed than a third-party counterpart. As a result, the estimated quality of Apple's apps is 2.25% higher than competing third-party apps on average. The quality premium may reflect better integration of Apple's apps into iPhones. Regarding search costs, it shows that consumers are significantly more likely to incur low search costs when searching for apps with higher search rankings in the top 50 search results.

To quantify the power of search ranking, Table 4 shows the semi-elasticities of demand with respect to price (Panel A) and search ranking (Panel B) for four popular apps in the entertainment category in July 2019. Panel A shows that a $1 increase in the installation price of an app leads to about 0.21% decrease in its demand. Panel B shows that a

[41] The category-specific top 5 developers are constructed based on total downloads of owned apps during the post-July difference-in-differences sample period.

Table 3: Estimates of the Demand Model

Variables	Parameter	Standard Error
Quality Coefficients		
log(1+Update Frequency)	0.166	0.084
Apple	1.133	0.447
Average Rating	0.512	0.175
log(Age) (month)	0.345	0.143
log(File Size) (MB)	0.423	0.055
#Screenshots	−0.042	0.015
log(1 + Description Length)	−0.041	0.147
1{Offer In-App-Purchase}	0.221	0.134
Game	−0.041	0.147
One-month Lagged Unobserved Quality	0.920	0.004
1{Category Contains Pre-installed Apps}	0.246	0.134
Constant	−13.806	1.030
Price	−0.216	0.048
Paid Installation	−1.719	0.240
Random Coefficients		
log(1+Update Frequency)	0.682	0.119
Search Cost Parameter Coefficients		
1{Top50 in Search Results}	−5.266	2.670
log(Search Ranking)|Top50	1.419	0.758
Month-FE	YES	
Average Estimated Quality	−8.261	(1.893)
Observations	52,959	

Notes: Standard deviation of estimated quality is in parenthesis.

10-position decline in the search ranking of an app leads to about 0.18% decrease in its demand. Dividing own-ranking semi-elasticity by own-price semi-elasticity returns an intuitive measure of *position effect*: a 10-position decline in search ranking is equivalent to a \$0.83 increase in price. The position effect is relatively small compared to those found in the other industries, reflecting that some consumers know their match values with these four apps before searching and therefore have zero search costs for these four apps.[42]

Table 4: (Semi-)Elasticities of Demand

	Netflix	TikTok	Hulu	Amazon Prime Video
		Panel A. Price Semielasticities		
Netflix	−0.205	0.008	0.004	0.003
TikTok	0.010	−0.209	0.004	0.002
Hulu	0.009	0.007	−0.212	0.002
Amazon Prime Video	0.008	0.006	0.003	−0.214
		Panel B. Ranking Semielasticities		
Netflix	−0.178	0.007	0.004	0.002
TikTok	0.008	−0.182	0.003	0.002
Hulu	0.008	0.006	−0.175	0.002
Amazon Prime Video	0.007	0.005	0.003	−0.175
		Panel C. Update Elasticities		
Netflix	1.395	−0.073	−0.038	−0.024
TikTok	−0.073	0.994	−0.026	−0.016
Hulu	−0.068	−0.048	0.954	−0.015
Amazon Prime Video	−0.043	−0.030	−0.016	0.601

Notes. Panel A reports the percentage change in the market share of the column-product with a \$1 increase in the row-product's installation price. Panel B reports the percentage change with a ten-position decline of the row-product's search ranking. Panel C reports percentage change in market share of the column-product with a 1 percent increase in the row-product's $\log(1+\text{update frequency})$.

The effect of update frequency on demand supports the hypothesized direct incentive for developers to update apps. Quantitatively, Panel C of Table 4 shows that a 1 percent increase in $\log(1+\text{update frequency})$ is associated with a 0.6 percent to 1.4 percent increase in market shares among the four Entertainment apps. Unsurprisingly, the own-update elasticities are larger than the cross-update elasticities.

[42] For example, in the online hotel industry, Ursu (2018) finds the effect of 1 position decline between \$0.55 and \$3.19, Chen and Yao (2017) find it to be \$0.21, Koulayev (2014) finds it ranging from \$2.93 to \$18.78.

5.3 Estimates of Self-Preferencing

Table 5 reports the estimation results of the search ranking model. It first confirms that higher quality leads to higher search ranking. Combined with the estimated preference for Apple's apps in Table 3, it implies that everything else equal, Apple's apps would receive higher rankings than third-party counterparts, even without platform self-preferencing. However, the estimated self-preferencing parameters are significantly positive in all categories with Apple's apps. It implies that consumers' preference for Apple's apps is not strong enough to justify the observed higher ranking of Apple's apps than third-party apps. Last but not least, the significantly positive coefficient on update frequency, as well as the quality coefficients, confirms the hypothesized indirect incentive for developers to update apps, namely, enhancing downloads through higher rankings.[43]

Table 5: Estimates of the Search Ranking Model

Variables	Parameter	Standard Error
Quality	0.126	0.013
Squared Quality ($\times$ 0.1)	-0.065	0.010
Squared Quality ($\times$ 0.01)	-0.046	0.005
Apple $\times$ Category Group 1	0.636	0.213
Apple $\times$ Category Group 2	1.981	0.184
Apple $\times$ Category Group 3	1.534	0.207
log(1+Update Frequency)	0.073	0.011
Apple $\times$ Months (Omit July 2019)	Yes	
Paid Installation $\times$ Category Groups	Yes	
Price$\times$ Category Groups	Yes	
Title Match $\times$ Category Groups	Yes	
Subtitle Match $\times$ Category Groups	Yes	
Lagged %5-star $\times$ Category Groups	Yes	
Lagged %4-star $\times$ Category Groups	Yes	
Lagged #Ratings $\times$ Category Groups	Yes	
Observations	52,959	
Average Latent Score	-0.393	
Pseudo R-sq	0.069	

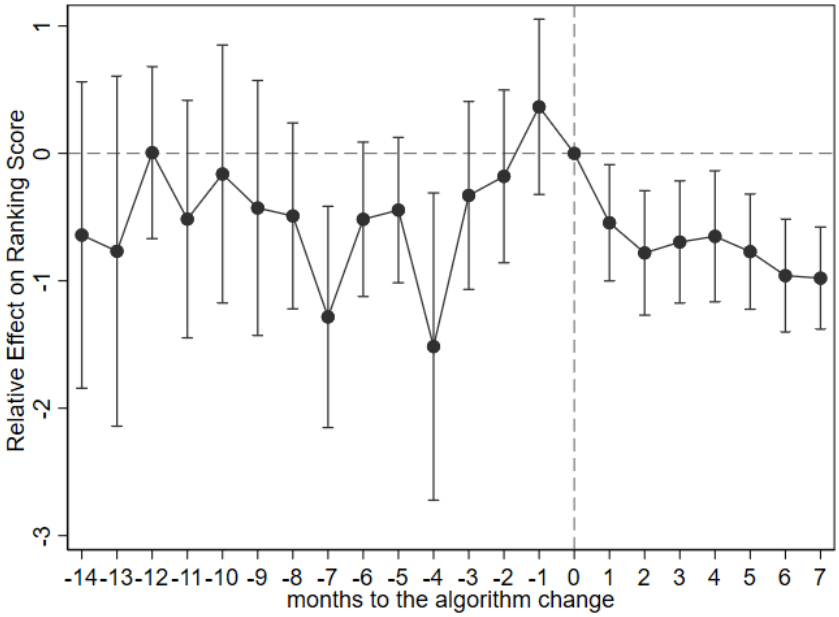
Notes: Category Groups are i) Game apps; ii) Non-Game categories without Apple's non-preinstalled apps; iii) Category Group 1/2/3: non-game categories with 1/3/7 Apple's non-preinstalled apps.

Figure 3 presents the coefficients on the interaction terms between Apple ownership

[43] Table D.11 reports estimates of the other parameters in the search ranking model. They are all qualitatively intuitive. Appendix B.3 reports the robustness of the estimates with respect to alternative specifications.

and month indicators, normalizing the month of the search algorithm change. As a cross-validation, it shows that the search ranking estimates reveal the search algorithm change in July 2019. Specifically, the self-preferencing parameters are estimated to be significantly lower after the search algorithm change.

Figure 3: Relative Self-preferencing Parameters across Months, Normalizing the Month of Algorithm Change



Notes. The figure presents point estimates of the effects of Apple ownership on ranking score in each month during the sample period, relative to July 2019. The bars indicate 95% confidence interval using standard errors clustered at the category-month level.

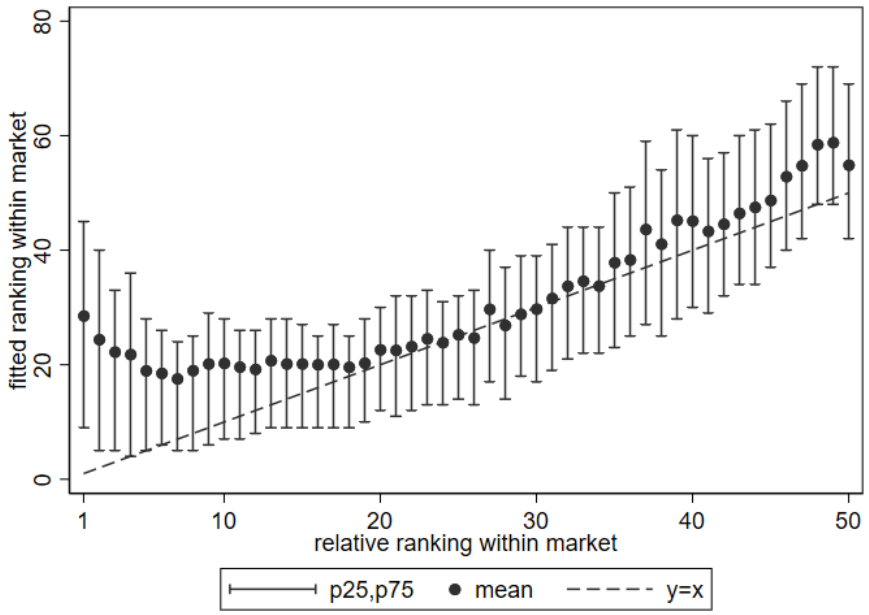
Figure 4 visualizes the fitness of the search ranking model. It plots the most-likely within-market ordering against the observed within-market ordering.[44] It shows that the model fits the data relatively well. For example, in most cases, the average fitted within-market orderings are close to the observed ones, and the intervals between the first and the third quartiles of the fitted orderings cover the observed ones.

5.4 Estimates of Supply

Table 6 reports the estimates of the supply model. In the second-stage supply model, Panel A shows that i) the in-app-advertising profits insignificantly increase with downloads; ii) the variable cost of update is convex with respect to update frequency; iii) the marginal

[44]For example, the search ranking of an app might be 33.5 in a market where there are 9 apps whose positions are strictly higher than 33.5. Then, the within-market ordering of this app is 10.

Figure 4: Fitness of the Search Ranking Model



Notes. The figure presents the most-likely within-market ordering (y-axis) against the observed within-market ordering (x-axis) across markets. The bars indicate the first and third quartiles of the predicted ranking.

cost of update is lower for non-game free apps with in-app purchase. Quantitatively, the results imply that an average app earns \$0.14 from in-app advertising with each new download. The implied average marginal cost of update is \$0.32 million. This is roughly equivalent to hiring 17.3 computer engineers in a month.[45] In the first-stage supply model, Panel B shows that the average upper bound is \$1.47 million, and the average lower bound is \$1.11 million.[46] The large update cost reflects the fact that the data focuses on popular apps and the average market size is as large as 105.9 million iPhone users per month.

Appendix C.2 compares structural and reduced-form estimates of the ATEs of the search algorithm change on update frequency, search ranking, and downloads. It shows that the structurally estimated ATEs have the same sign as those estimated from DiD, with smaller magnitudes. The discrepancy in magnitudes may reflect additional structural

[45] The figure comes from the following back-of-envelope calculation. The first-order Taylor Expansion of $\log(1+x)$ has $1/(1+x)$ as the coefficient on the first-order difference. The average update frequency among updated apps is 1.35. Then the cost of one additional release-note-augmented update is roughly $0.32/(1+1.35) = 0.14$ million dollars. The average salary for a computer engineer is about \$97,000 per year in California in 2020. Therefore, the cost is roughly equivalent to hiring 17.3 computer engineers in a month.

[46] Figure E.9 plots the density curves of the estimated upper and lower bounds of the fixed costs of update. It shows that the estimated upper bounds turn out to first-order stochastically dominate the estimated lower bounds.

Table 6: Estimates of the Supply Model

Variables	Parameter	Standard error
Panel A: Second-Stage Supply Model		
In-App-Advertising Profit		
Downloads	0.137	0.358
Squared downloads	−0.010	0.074
Marginal Cost of Updates		
log(1+Update Frequency)	3.008	0.108
Paid Installation	0.107	0.031
1{Offer In-App Purchase}	−0.216	0.034
Game	0.271	0.015
Constant	−2.127	0.098
Month-FE	YES	
Average Marginal In-App-Advertising Profit ($)	0.135	
Average Marginal Cost (million$)	0.322	
Observations	25,326	
Panel B: First-Stage Supply Model		
Fixed Costs of Updates (million$)	Observations	Mean
Upper bounds	506	1.469
Lower bounds	176	1.114

Notes: Marginal marginal profit IS with respect to downloads. Marginal update cost is with respect to log(1+Update Frequency).

changes in the search algorithm change other than self-preferencing.

6 Counterfactual Simulations

To quantify the welfare effect of self-preferencing, I simulate counterfactuals where the estimated self-preferencing parameters, $\{\theta_{1tk}\}$ in Equation (9), become zero. The difference between the counterfactual and the status-quo market outcomes captures the effect of self-preferencing in the studied empirical context. The simulations involve all categories with Apple's non-preinstalled apps and the two months with the largest estimated self-preferencing parameters.[47]

The following definition equation of consumer surplus summarizes how self-preferencing could affect welfare.

$$CS_{gt} := M_t \cdot \mathbb{E}_{(\varepsilon,c,\sigma,y,\omega)} \left[-\left(u_{ij(i)gt} - \sum_{l \in \mathscr{S}_{igt}} c_{ilgt}\right) / \alpha \right] \tag{20}$$

where $j(i)$ denotes the chosen app of consumer i, $\mathscr{S}_{i\cdot}$ denotes the set of searched apps of consumer i.[48] Holding product characteristics fixed, an alternative ordering of products changes the search cost of a given product l for a consumer i, c_{ilgt}, which changes the set of searched products by consumer i, $\mathscr{S}_{igt}$, as well as the chosen product of consumer i, $j(i)$. This leads to changes in the incurred *search costs* , $\sum_{l \in \mathscr{S}_{igt}} c_{ilgt}$, and the *choice quality*, $u_{ij(i)gt}$. Furthermore, eliminating self-preferencing may encourage independent apps to provide better apps through more updates. This in turn affects choice quality and search costs on the consumer side, as well as profits on the supply side. I first examine the effect on update frequency, then examine the effect on consumer surplus and developer profit.

[47] In particular, the categories are weather, utilities, productivity, music, entertainment, health, and shopping. The two months are June and July in 2019 as reported in Figure 3. For more details on the simulations, please see Appendix C.1.

[48] All expressions for expected consumer surplus are up to a constant. See Small and Rosen (1981). Recall that α is the price coefficient in Equation (2). For more details on the computation, please see Appendix C.3.

6.1 Effects of Self-Preferencing on Update Frequency

I take four main steps to find counterfactual update frequencies and app qualities. First, I take five simulation draws of the fixed costs of update from a range that is consistent with the identified bounds. Second, for each drawn vector of fixed costs, I compute the first-stage equilibrium update portfolios based on best-response iterations. Third, assuming that the shocks to the marginal cost of update remain the same between status-quo and counterfactual, I compute the counterfactual second-stage equilibrium update frequency with the backed-out ω_{jgt}'s whenever possible, given the counterfactual first-stage equilibrium update portfolios.[49] Finally, I feed the counterfactual update frequency into Equation (8) to calculate counterfactual app quality. I report the average market outcomes over the five draws of fixed costs of update. For more details on each step, please see Appendix C.

Table 7: Effects of Self-preferencing on Update Frequency and App Quality

		Status-quo	Shut-down	Percentage Change (%)			
	obs	mean	mean	mean	std	min	max
Market-Level Results							
Average Update Frequency	13	1.21	1.23	1.86	3.60	0	10.29
Average App Quality	13	−6.29	−6.29	0.01	0.02	0	0.05
Number of Updated Apps	13	6.08	6.06	−0.19	0.69	−2.5	0
App-Market-Level Results							
Update Frequency[a]	105	1.22	1.25	1.41	9.93	−21.42	65.61
App Quality	105	−6.34	−6.33	0.02	0.22	−0.58	1.64
Probability of Update[a]	105	0.75	0.75	−0.25	2.25	−20.00	0

Notes: Update frequency is the monthly number of updates weighted by the length of release notes. Each market is a category/month pair. For the status-quo case, there is the estimated self-preferencing. For the shut-down case, there is no self-preferencing. In cases[a], the percentage changes are conditional on 79 observations that upgrade in the status quo.

Table 7 shows the counterfactual simulation results for update frequency and app quality. The top panel shows that, at market level, eliminating self-preferencing increases update frequency by 1.86% and app quality by 0.01% on average. The small quality increase reflects the limited positive effects of updates on app quality. The bottom panel shows that, at product level, the average effects are similar. Sizeable heterogeneity exists

[49] Backed-out ω_{jgt}'s are not available when the app is not updated in the data but updated in the counterfactual. In such cases, I use the same draws of ω's as those for computing the bounds on fixed costs of update. This step ensures that the difference between the simulated updates and observed updates is not due to different ω's.

in update effects. The percentage change of update frequency has a standard deviation that is 1.9 or 7 times the mean at market or product level. The range of the percentage change of update frequency, as well as the extensive-margin effects captured by the probability of update and the number of updated apps, shows that some independent apps update less without platform self-preferencing while others update more.[50]

The following regression sheds light on the heterogeneous update effects (standard errors are in parenthesis).

$$\begin{aligned} 1\{\Delta UpdateFreq_{jgt} < 0\} = \quad & 0.18 \times 1\{\Delta Ranking^{NG}_{jgt} = 0\} + \quad 0.24 \times \#Apps_{f(j)gt} \\ & (0.08) \qquad\qquad\qquad\qquad\quad (0.01) \\ & + 0.01 \qquad + \varepsilon_{jgt} \\ & (0.04) \end{aligned}$$

where the left-hand side is the indicator of fewer updates without self-preferencing, the right-hand-side variables include an indicator of higher search ranking in non-game simulation without platform self-preferencing, the number of active apps owned by the app's developer, and a constant. The correlation results show that negative update effects are associated with indirectly affected apps whose rankings remain fixed without platform self-preferencing and those apps of multi-product developers. The first factor is consistent with a business-stealing effect of directly affected apps: indirectly affected apps lose business to directly affected apps that are boosted up by the elimination of self-preferencing. Eliminating self-preferencing shifts the residual demand curves of the indirectly affected apps leftward and discourage them from upgrading apps.[51] The second factor is consistent with cannibalization within firms: a multi-product developer might update an app less if the app receives a higher search ranking and cannibalize the revenues of other apps that s/he has.

6.2 Effects of Self-Preferencing on Welfare

Table 8 shows the effects of eliminating self-preferencing on search rankings, downloads, and welfare. The first row confirms that eliminating self-preferencing does *not* change average search rankings; it only re-allocates the positions. Therefore, eliminating self-preferencing is a differential change in search costs across different products instead of a

[50] Table D.19 reports the quantiles of update effects by the sign of update effects.

[51] To verify the business-stealing effect, I regress the indicator of lower downloads without self-preferencing in non-game simulation on the indicator of unchanged search ranking without self-preferencing and a constant. The coefficient on the indicator is 0.55 with a robust standard error of 0.08.

Table 8: Effects of Self-preferencing on Search Rankings, Installations and Welfare

	Variable	Status-quo	Shut-down	Mean Δ	Mean $\%\Delta$
(1)	Average Search Rankings	40.54	40.54	0.00	0.00
(2)	- Independent apps	41.24	40.74	-0.50	-1.22
(3)	- Apple's apps	15.63	32.71	17.08	173.30
(4)	Total Downloads (million)	6.29	6.45	0.16	1.47
(5)	- Independent apps	6.20	6.38	0.18	1.95
(6)	- Apple's apps	0.09	0.07	-0.02	-22.56
(7)	Consumer Surplus (million \$)	297.10	298.00	0.94	0.28
(8)	Search Costs (million \$)	25.22	25.16	-0.06	0.37
(9)	Choice Quality (million \$)	322.30	323.20	0.88	0.25
(10)	Third-party Profits (million \$)	48.72	49.05	0.34	0.66
Number of Markets			13		

Notes: For the status-quo case, there is identified self-preferencing. For the shut-down case, there is no self-preferencing. Third-party profits are subject to a constant that does not change with downloads. Reported values are average across markets and fixed cost draws.

uniform reduction in search costs and thus only affects equilibrium search costs endogenously. It makes the self-preferencing issue different from most studies in the literature on information friction and product competition.[52] Specifically, the second and third rows show that, in an average market, the elimination boosts up independent apps by 0.5 positions and decreases their search costs while lowering Apple's apps by 17.1 positions and increasing the search costs for Apple's apps. The smaller change in independent apps' search rankings reflects that there are much more independent apps than Apple's apps in the market.

The fourth row shows that eliminating self-preferencing increases total downloads by 1.5 percent in an average market. Considering platforms' revenues from independent apps and platform-owned apps, one can do a back-of-envelope calculation with the changes in downloads: independent apps' downloads increase by 0.18 million, contributing to 0.05 million increase in the commission revenue measured in the unit of downloads with the 30% commission rate. Such an increase in commission revenue is larger than the decreased downloads of 0.02 million for Apple's apps.[53] Therefore, it remains a mystery

[52] Examples for uniform changes in information frictions include the introduction of internet (Orlov, 2015, Ellison and Ellison, 2018), price transparency tools (Brown (2019)), intermediaries (Salz, 2020), and the influx of novice investors in the mutual fund industry (Hortaçsu and Syverson, 2004).

[53] Appendix Table D.21 shows that the platform revenues will increase by 0.16 million dollars with the elimination.

why Apple was self-preferencing, which is out of the scope of this paper.[54]

The last four rows show positive and modest welfare benefits from eliminating the estimated self-preferencing. In an average market, consumer surplus increase by 0.28%, and third-party profits increase by 0.66%. As a comparison, from the first three rows, one can calculate that the gap between the search rankings of independent apps and Apple's apps shrinks by 69% after the elimination.[55] Search costs turn out to account for a smaller portion of consumer surplus than choice quality. The elimination decreases the incurred search costs by 0.06 million dollars and increases choice quality by 0.88 million dollars.[56] The fact that search costs are low in the studied empirical context might lead to the small welfare effect of self-preferencing: consumers can still scroll down to their preferred products with self-preferencing.

Figure 5 illustrates the heterogeneity of welfare effects across categories and months. It shows that the most affected category is Entertainment, where consumer surplus increases by 1.1% to 1.2% and third-party profits increase by 2.0% to 2.3% after eliminating the estimated self-preferencing. This result is not surprising, given that Entertainment is also the category where Apple is historically good and competes intensely with third-party products. The figure also shows a positive correlation between welfare effects and the extent to which Apple's apps have benefited from self-preferencing in the market. For example, in the Entertainment category where the welfare effects are large, the three Apple's non-preinstalled apps' search rankings would decrease by 10.1 to 11.2 times in total without self-preferencing, which is also among the largest x-axis values across categories.

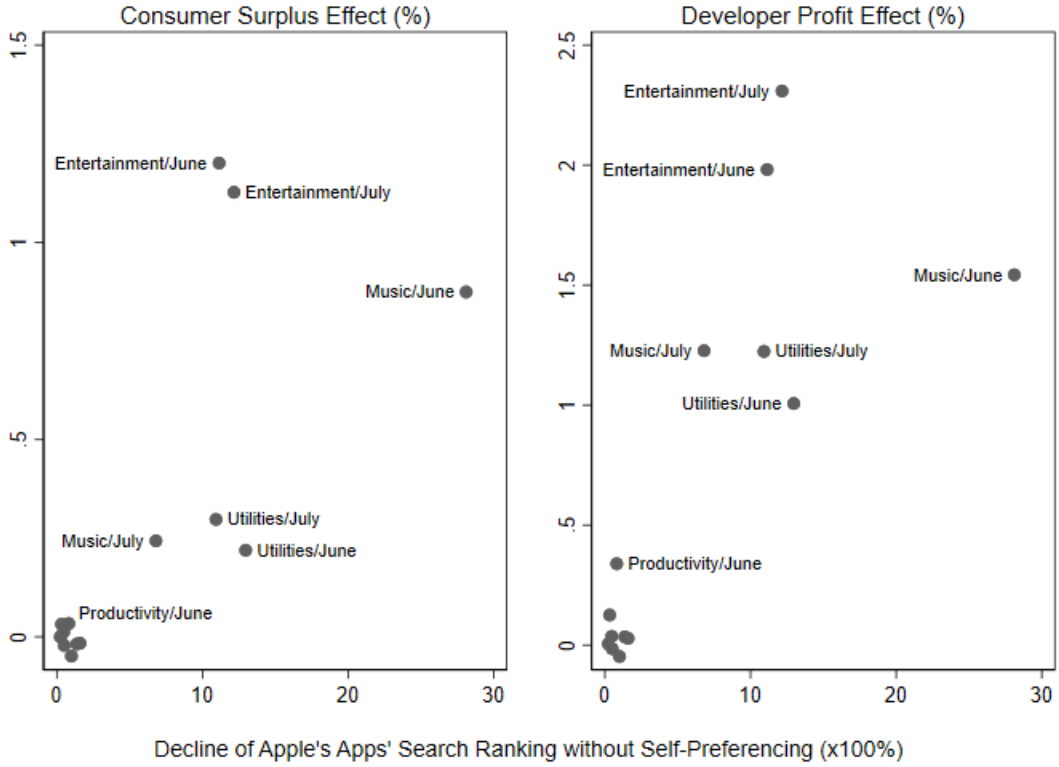
Table 9 compares the welfare effects with or without update adjustment. It shows that not incorporating independent apps' update adjustment will under-predict the gains in consumer surplus by 44% and over-predict the gains in third-party profits by 6.7%. While the average quality does not increase much due to update adjustment on average, as reported in Table 7, the results reflect that consumers disproportionately benefit from apps that update more and achieve higher quality after eliminating self-preferencing.

[54] I argue that self-preferencing might be beneficial for the platforms' search ads revenue, because self-preferencing increases the value of the sponsored positions for independent apps.

[55] The calculation is: 1-(40.74-32.71)/(41.24-15.63)=0.69.

[56] As a comparison to other markets, in mortgage markets, Allen, Clark and Houde (2019) finds that 50% of consumer surplus gain from reduced search frictions is associated with reduced search costs, while 22% gain is associated with inefficient matching.

Figure 5: Heterogenous Welfare Effects across Categories and Months



Notes. The $x-axis$ variable is $\sum_{j\in\mathcal{J}_{Apple}}(pos_j^0 - pos_j^1)/pos_j^1$, where $pos_j^{1/0}$ is the search ranking with/without platform self-preferencing, holding update frequency fixed; and $\mathcal{J}_{Apple}$ collects non-preinstalled apps owned by Apple. The vertical axes are percentage changes of consumer surplus (left) and third-party profits (right). The unlabeled category/month pairs are Productivity/July, Health/June, Health/July, Shopping/June, Shopping/July, Weather/June, and Weather/July. They are too close to be labeled in the figure. The reported welfare effects on Music/July allow all observed independent developers to change update frequency while holding update portfolios fixed.

Table 9: Welfare Effects With and Without Update Adjustment

Update Adjustment	Yes	No	Difference (%)
Mean ΔConsumer Surplus (million $)	0.94	0.53	44.01
Mean ΔThird-Party Profits (million $)	0.34	0.36	−6.65

Notes: The first column is from the third column in Table 8. The second column is from simulations of eliminating self-preferencing while holding update frequencies fixed at observed values.

7 Conclusion

This paper provides a structural model of consumer search and quality-upgrading competition, in order to deal with potential platform self-preferencing in search ranking. I apply the model to the U.S. Apple's App Store data. I find positive effects of platform ownership on search ranking, conditional on app quality, pointing to platform self-preferencing. In counterfactual simulations without the estimated self-preferencing, I find positive and modest innovation effects and welfare effects of eliminating self-preferencing in the studied empirical context. For example, in an average market, update frequency increases by 1.86%, consumer surplus increases by 0.28%, and third-party profits increase by 0.66%, after the elimination. I also find heterogeneous effects across different independent apps and categories. For example, the most affected category is Entertainment. When motivating and estimating the structural model, I exploit an unexpected search algorithm change that dropped some Apple's apps from top search results.

It remains a mystery why Apple was self-preferencing its own products in search results. The counterfactual simulations show that the lost commission revenue from independent apps due to self-preferencing is larger than the increased revenues from Apple's apps. Answering this question is an interesting topic for future research. One potential reason is to increase search-ads revenue with self-preferencing, because self-preferencing increases the value of sponsored positions in search results for independent apps.

The modest welfare gains from eliminating self-preferencing support regulations against self-preferencing on the one hand, but also imply that self-preferencing might not be the most beneficial margin for antitrust policies in the studied empirical context. The structural model can be applied in other contexts where search ranking algorithms may affect suppliers' incentives and total welfare.

References

Aghion, Philippe, Nick Bloom, Richard Blundell, Rachel Griffith, and Peter Howitt. 2005. "Competition and innovation: An inverted-U relationship." *The quarterly journal of economics*, 120(2): 701–728.

Allen, Jason, Robert Clark, and Jean-François Houde. 2019. "Search frictions and market power in negotiated-price markets." *Journal of Political Economy*, 127(4): 1550–1598.

Allon, Gad, Georgios Askalidis, Randall Berry, Nicole Immorlica, Ken Moon, and Amandeep Singh. 2021. "When to be agile: Ratings and version updates in mobile apps." *Management Science*.

Augereau, Angelique, Shane Greenstein, and Marc Rysman. 2006. "Coordination versus differentiation in a standards war: 56k modems." *The RAND Journal of Economics*, 37(4): 887–909.

Beggs, Steven, Scott Cardell, and Jerry Hausman. 1981. "Assessing the potential demand for electric cars." *Journal of econometrics*, 17(1): 1–19.

Berry, Steven, Alon Eizenberg, and Joel Waldfogel. 2016. "Optimal product variety in radio markets." *The RAND Journal of Economics*, 47(3): 463–497.

Berry, Steven, and Philip Haile. 2016. "Identification in differentiated products markets." *Annual review of Economics*, 8: 27–52.

Berry, Steven, James Levinsohn, and Ariel Pakes. 1995. "Automobile prices in market equilibrium." *Econometrica: Journal of the Econometric Society*, 841–890.

Brown, Zach Y. 2017. "An empirical model of price transparency and markups in health care." *Manuscript*.

Brown, Zach Y. 2019. "Equilibrium effects of health care price information." *Review of Economics and Statistics*, 101(4): 699–712.

Chen, Nan, and Hsin-Tien Tsai. 2019. "Steering via Algorithmic Recommendations." *RAND Journal of Economics, Forthcoming*.

Chen, Yuxin, and Song Yao. 2017. "Sequential search with refinement: Model and application with click-stream data." *Management Science*, 63(12): 4345–4365.

Chu, Chenghuan Sean. 2010. "The effect of satellite entry on cable television prices and

product quality." *The RAND Journal of Economics*, 41(4): 730–764.

Crawford, Gregory S. 2012. "Endogenous product choice: A progress report." *International Journal of Industrial Organization*, 30(3): 315–320.

Crawford, Gregory S, and Ali Yurukoglu. 2012. "The welfare effects of bundling in multichannel television markets." *American Economic Review*, 102(2): 643–85.

Crawford, Gregory S, Oleksandr Shcherbakov, and Matthew Shum. 2019. "Quality overprovision in cable television markets." *American Economic Review*, 109(3): 956–95.

Crémer, Jacques, Yves-Alexandre de Montjoye, and Heike Schweitzer. 2019. "Competition Policy for the Digital Era, final report presented to the European Commission."

De Corniere, Alexandre, and Greg Taylor. 2019. "A model of biased intermediation." *The RAND Journal of Economics*, 50(4): 854–882.

Diamond, Peter A. 1971. "A model of price adjustment." *Journal of economic theory*, 3(2): 156–168.

Dinerstein, Michael, Liran Einav, Jonathan Levin, and Neel Sundaresan. 2018. "Consumer price search and platform design in internet commerce." *American Economic Review*, 108(7): 1820–59.

Draganska, Michaela, Michael Mazzeo, and Katja Seim. 2009. "Beyond plain vanilla: Modeling joint product assortment and pricing decisions." *QME*, 7(2): 105–146.

Eizenberg, Alon. 2014. "Upstream innovation and product variety in the us home pc market." *Review of Economic Studies*, 81(3): 1003–1045.

Ellison, Glenn, and Sara Fisher Ellison. 2018. "Match quality, search, and the Internet market for used books." National Bureau of Economic Research.

Ershov, Daniel. 2020. "Consumer product discovery costs, entry, quality and congestion in online markets." *Unpublished manuscript*.

Fan, Ying. 2013. "Ownership consolidation and product characteristics: A study of the US daily newspaper market." *American Economic Review*, 103(5): 1598–1628.

Fan, Ying, and Chenyu Yang. 2020. "Competition, product proliferation, and welfare: A study of the US smartphone market." *American Economic Journal: Microeconomics*, 12(2): 99–134.

Fan, Ying, and Chenyu Yang. 2022. "Estimating discrete games with many firms and many decisions: An application to merger and product variety." National Bureau of Economic Research.

Farronato, Chiara, Andrey Fradkin, and Alexander MacKay. 2023. "Self-preferencing at Amazon: evidence from search rankings." National Bureau of Economic Research.

Feasey, Richard, and Jan Krämer. 2019. *Implementing effective remedies for anti-competitive intermediation bias on vertically integrated platforms.* Centre on Regulation in Europe asbl (CERRE).

Fishman, Arthur, and Nadav Levy. 2015. "Search costs and investment in quality." *The Journal of Industrial Economics*, 63(4): 625–641.

Ghose, Anindya, and Sang Pil Han. 2014. "Estimating demand for mobile applications in the new economy." *Management Science*, 60(6): 1470–1488.

Goldfarb, Avi, and Catherine Tucker. 2019. "Digital economics." *Journal of Economic Literature*, 57(1): 3–43.

Hagiu, Andrei, Tat-How Teh, and Julian Wright. 2022. "Should platforms be allowed to sell on their own marketplaces?" *The RAND Journal of Economics*, 53(2): 297–327.

Hausman, Jerry A. 1996. "Valuation of new goods under perfect and imperfect competition." In *The economics of new goods*. 207–248. University of Chicago Press.

Heiss, Florian, and Viktor Winschel. 2008. "Likelihood approximation by numerical integration on sparse grids." *journal of Econometrics*, 144(1): 62–80.

Hortaçsu, Ali, and Chad Syverson. 2004. "Product differentiation, search costs, and competition in the mutual fund industry: A case study of S&P 500 index funds." *The Quarterly journal of economics*, 119(2): 403–456.

Hristakeva, Sylvia. 2022. "Vertical contracts with endogenous product selection: An empirical analysis of vendor allowance contracts." *Journal of Political Economy*, 130(12): 3202–3252.

Janssen, Rebecca, Reinhold Kesler, Michael Kummer, and Joel Waldfogel. 2021. "GDPR and the Lost Generation of Innovative Apps."

Koulayev, Sergei. 2014. "Search for differentiated products: identification and estima-

tion." *The RAND Journal of Economics*, 45(3): 553–575.

Lam, H Tai. 2021. "Platform Search Design and Market Power."

Lee, Kwok Hao, and Leon Musolff. 2021. "Entry Into Two-Sided Markets Shaped By Platform-Guided Search."

Leyden, Benjamin T. 2019. "There's an app (update) for that: Understanding product updating under digitization." Working paper, Cornell University.

Mazzeo, Michael J. 2002. "Product choice and oligopoly market structure." *RAND Journal of Economics*, 221–242.

Moraga-González, José L, and Yajie Sun. 2023. "Product quality and consumer search." *American Economic Journal: Microeconomics*, 15(1): 117–141.

Moraga-González, José Luis, Zsolt Sándor, and Matthijs R Wildenbeest. 2023*a*. "Consumer search and prices in the automobile market." *The Review of Economic Studies*, 90(3): 1394–1440.

Moraga-González, José Luis, Zsolt Sándor, and Matthijs R Wildenbeest. 2023*b*. "A Framework for the Estimation of Demand for Differentiated Products with Simultaneous Consumer Search." Tinbergen Institute Discussion Paper.

Nevo, Aviv. 2000. "Mergers with differentiated products: The case of the ready-to-eat cereal industry." *The RAND Journal of Economics*, 395–421.

Orlov, Eugene. 2015. "The effect of the internet on performance and quality: Evidence from the airline industry." *Review of Economics and Statistics*, 97(1): 180–194.

Salz, Tobias. 2020. "Intermediation and competition in search markets: An empirical case study." National Bureau of Economic Research.

Seim, Katja. 2006. "An empirical model of firm entry with endogenous product-type choices." *The RAND Journal of Economics*, 37(3): 619–640.

Singh, Amandeep, Kartik Hosanagar, and Aviv Nevo. 2021. "Network Externalities and Cross-Platform App Development in Mobile Platforms." *Available at SSRN 3911638*.

Small, Kenneth A, and Harvey S Rosen. 1981. "Applied welfare economics with discrete choice models." *Econometrica: Journal of the Econometric Society*, 105–130.

Stigler, George J. 1961. "The economics of information." *Journal of political economy*,

69(3): 213–225.

Ursu, Raluca M. 2018. "The power of rankings: Quantifying the effect of rankings on online consumer search and purchase decisions." *Marketing Science*, 37(4): 530–552.

Weitzman, Martin L. 1979. "Optimal search for the best alternative." *Econometrica: Journal of the Econometric Society*, 641–654.

Wolinsky, Asher. 2005. "Procurement via sequential search." *Journal of Political Economy*, 113(4): 785–810.

Wollmann, Thomas G. 2018. "Trucks without bailouts: Equilibrium product characteristics for commercial vehicles." *American Economic Review*, 108(6): 1364–1406.

Zennyo, Yusuke. 2022. "Platform encroachment and own-content bias." *The Journal of Industrial Economics*, 70(3): 684–710.

For Online Publication

Appendix A Details on Data

A.1 Sample Selection

Here I describe my sample selection. A category is in the sample if it has benchmark conversion rates data from AppTweak. Within each category, the set of apps and keywords are selected with the following process.

App Selection Process:

1. Find apps that have ranked top-50 in the top-grossing charts for the given category on any day between April 2019 and September 2019.
2. Conditional on selection into step 1, find the 50 mostly downloaded apps in the given category based on the annual downloads in 2019.
3. Repeat the above steps for top-paid charts, in order to ensure enough price variation to identify price elasticity.[57]
4. Drop an app-month observation if i) the app has zero downloads in that month, or ii) the app has unobserved file size or ratings in that month. I assume the missing is at random.

Keyword Selection Process:

1. Find keywords that have entered the list of recently used keywords of a selected app, based on AppTweak's keyword suggestions. The "recency" on AppTweak is the last 3 months, which corresponds to March 2020 to June 2020. The suggested keywords are keywords that either i) the app has ranked in top 100 in the keyword recently; ii) the app's title, subtitle, or description contains the keyword.
2. For each app/keyword pair, track the app's historical search ranking in the keyword on each day during 2019. Define an app's adoption of a keyword as the app showing up in the top 500 search results of the keyword. For each category, find the 60 keywords that have the most apps in the category adopting the keyword.

[57] To be clear, the sample does not include apps that only show up in top-free charts. Examples include apps that monetize only through in-app advertising or provide complementary values to other products owned by the firm, such as Google Map. These apps are likely to have different objective functions than those modeled in this paper and do not quite rely on the app store to earn profits.

A.2 Variable Definition and Data Sources

- **Downloads** are the estimated monthly downloads from AppTweak. A download is the first-time installation by a unique consumer.
- **Revenues** are the estimated monthly revenues from AppTweak. Revenues include installation revenue and before-Apple-tax revenues from in-app purchase and subscription.
- **Conversion rate** is the average ratio of downloads over impressions across connected apps in the same type/category/month group in percentage. An impression is a view by a consumer. Connected apps are apps owned by customers of AppTweak. These customers are third-party app developers. By connecting to AppTweak and receiving service from AppTweak, they allow AppTweak to get access to their business data, including conversion rates. For confidentiality, AppTweak only provides conversion rate data at type/category/month level. I observe conversion rates for 1337 type/category/month combinations. However, 411 type/category/month combinations miss conversion rates data, which I assume as missing at random.
- **Type** indicates whether an app requires payment for installations. It is observed at app level. Type data is from AppTweak.
- **Apple** indicates whether an app is developed by Apple. Developer ID data is from AppTweak.
- **Offer In-app Purchase** indicates whether an app offers in-app purchase and subscription (IAP). It is observed at app level. IAP availability data is from AppTweak.
- **Installation Price | Paid** is the price for installation conditional on paid apps. Occasionally, the price may be zero even for paid apps. It might reflect temporary sales of the app. Historical price data is from AppTweak.
- **Top-50 in Search Results** indicates whether an app shows up in the top-50 positions of any keyword selected for its category on any day in a given month. It changes across apps, categories, and months.
- **Search Ranking** is the average ranking of an app across the keywords and days where it shows up in the top-50 positions in a category and month. Historical

search results data is from AppTweak for non-preinstalled apps and SensorTower for pre-installed apps.

- **Update Frequency** is the weighted number of updates in a month. The weight is based on category-specific quartiles of release notes. Specifically, for each category, I calculate the three quartiles of release note length across all versions released by apps in the category during the sample period. Then, if an update has a release note shorter than the first quartile, the update is weighted by 0.25. If an update has a release note longer than the first (second) quartile but shorter than the second (third) quartile, the update is weighted by 0.5 (0.75). If an update has a release note longer than the third quartile, the update is weighted by 1. Then, the sum of the weighted updates of an app in a month based on the category-specific weights gives the update frequency of the app in the category and month. Version history data is from AppTweak.

- **Average Rating** is the average number of stars of an app's ratings from consumers in a category and month. A rating ranges from 1 to 5 stars in integers.[58] Historical rating data is from AppTweak.

- **Age** is the number of months since the release date of the app. Release date data is from AppTweak.

- **File Size** is the average size of an app in a month in megabytes (MB), weighted by the number of days. Historical file size data is from AppTweak.

- **#Screenshots** is the average number of screenshots on the view page of an app in a month, weighted by the number of days when the set of screenshots show up. Historical screenshots data is from AppTweak.

- **Description Length** is the average number of characters of the descriptions of an app in a month, weighted by the number of days when the description shows up.

[58] Developers may reset an app's summary rating when releasing a new app version, while written customer reviews are kept unchanged. The app's product page will display a message stating that the app's summary rating was recently reset, until enough customers rate the new version and a new summary rating appears. Therefore, the reset behavior will lead to a reduced number of ratings as time goes by. In my data, on a daily base, 1.4 percent to 2.5 percent of observations see a decrease in the number of a given star-level ratings as time goes by. Intuitively, there is a cost of resetting ratings: the resulting fewer ratings may discourage consumers from downloading the app. Thus, I argue that reset behaviors rarely happen in my sample.

Historical description data is from AppTweak.

- **Market Share** is the ratio of downloads over market size in percentage.
- **#iPhone users** is the number of iPhone users in the US in a month, from Comscore. I use it as the measurement of market size.

Now I provide definitions for other variables used in the empirical analysis. Table D.3 provides their summary statistics.

- **Brand Keyword** indicates whether a keyword is an app's name. It is provided by AppTweak.
- **Search Volume** is an integer between 5 and 100 that indexes how many consumers search for a keyword on a day. It is constructed by Apple. I obtain the historical search volume data from SensorTower.
- **Title (Subtitle) Match** is the average value of an app/keyword/day-specific indicator across different keywords and days for a given app in a category and month, weighted by keywords' search volume. The indicator tells whether a keyword contains any word in the title (subtitle) of the app on a given day. It measures the average likelihood that an app's title (subtitle) matches the popular keywords in a category in a month. Historical title and subtitle data is from AppTweak.
- **%5-star ratings (%4-star ratings)** is the ratio of 5-star (4-star) ratings over the number of ratings an app has received until a given day.
- **#Pre-installed Apps** is the number of pre-installed apps in the category. During the sample period, there is no newly pre-installed apps on Apple. The data is from public information.
- **min. Top-50** is the minimum value of the indicators for getting into the top-50 positions across pre-installed apps in a category in a month. It serves as the counterpart of "Top-50 in Search Results" to calculate the $\mu_{0gt}(\lambda)$ for the outside option.
- **min. Search Ranking** is the minimum value of the "Search Ranking" defined above across pre-installed apps in a category in a month. It serves as the counterpart of "Search Ranking" to calculate the $\mu_{0gt}(\lambda)$ for the outside option. Historical search results data for pre-installed apps are from SensorTower.

A.3 Descriptive Patterns from the Search Volume Data

Here I provide descriptive test results for consumers having rational expectations of the observable app characteristics before search. I exploit data on search volumes of keywords, an integer between 5 and 100 indicating the number of consumers searching for the keyword. Different keywords have different sets and orderings of apps in the search results. If consumers know app characteristics before search, they should search for keywords that have more high-value apps more and search for keywords that have less high-value apps less. In other words, the characteristics of apps in the search results of an keyword should affect the search volume of the keyword. The above theoretical implication motivates the following regression equation:

$$\begin{aligned} SearchVolume_{kgt} =& \beta_1^V AllPreinstall_{kgt} + AllPreinstall_{kgt} \times \{\bar{\boldsymbol{x}}_{kgt}^V \cdot \boldsymbol{\beta}_2^V + \beta_3^V \bar{p}_{kgt}\} \\ &+ \beta_4^V \overline{Apple}_{kgt} + \beta_5^V \overline{Preinstall}_{kgt} + \beta_5^V brand_k + \lambda_{gt}^V + \varepsilon_{kgt}^V \end{aligned} \tag{A.21}$$

where $AppPreinstall_{kgt}$ indicates whether all the top50 search results in keyword k month t contain no apps in category g other than pre-installed apps. Within each category/month pair gt, in the case of there are non-pre-installed apps showing up in the top50 search results for a keyword k, I calculate the average prices across these apps, denoted by $\bar{p}_{kgt}$. Similarly, $\bar{\boldsymbol{x}}_{kgt}$ denotes the vector of average levels of app characteristics, including update levels, average rating, age, file size, number of screenshots, description length, offer in-app-purchase or not, and paid installation or not. To capture the idea that higher-ranked products are more considered by consumers, these characteristics are weighted by $1/log(1+ranking_{jkt})$, where $ranking_{jkt}$ is the average ranking of app j in keyword k in month t. $\overline{Apple}_{kgt}$ denotes the ratio of observed positions that are taken by Apple's apps in search results of keyword k. Similarly, $\overline{Preinstall}_{kgt}$ denotes the ratio of observed positions that are taken by pre-installed apps in search results of keyword k. $brand_k$ indicates whether the keyword is a brand-name keyword.λ_{gt} denotes category-month fixed effects, capturing unobservables that are keyword-invariant and change across markets. In the robustness check, I use keyword-category fixed effects and month-fixed effects, omitting the brand-name keyword indicator. The keyword-category fixed effects capture unobservables that are time-invariant and change across keyword/category pairs. The month-fixed effects capture unobservables that are keyword/category-invariant and change over time.

Table D.7 reports the summary statistics of the data for estimating the above equation.

The left panel reports the data used for the main specification: search volume and average app characteristics across observed apps in top50 search results. The right panel reports the data used for robustness check: search volume and app characteristic of the observed top1 app. It shows that, for an average keyword/category/month pair, there are 2% of observed top50 positions taken by Apple's apps. It also shows that, about 5% of observed top1 positions are taken by Apple's apps. It also presents that, compared to an average observed top50 app, an average top1 app has more experiences, larger file size, more screenshots, longer descriptions and higher installation prices. In the left panel, it shows that, among keyword/category/month combinations that have observed top50 search results, the average search volume is 48.29 with a standard deviation of 13.88, and 34% of them are generated with brand-name keywords.

Table D.8 presents the estimation results of Equation A.21. The first column reports the main specification results. It shows that consumers are significantly more likely to search for keywords whose top50 search results contain apps that on average update more, have more experience, smaller in file size, have more screenshots and shorter description, offer in-app-purchase, have lower or even zero installation price. It indicates that consumers at least know these app characteristics to some extent such that they can predict what apps they will see in the search results and thus choose what keywords to search for. Although average rating does not significantly affect search volume in the main specification, its coefficient is insignificantly positive across all specifications, and becomes significantly positive when only looking at top1 search results and controlling for keyword-category fixed effects and month-fixed effects. Overall, the descriptive evidence is consistent with the idea that consumers know the observable app characteristics to some extent before search.

Appendix B Details on the Model and Estimation

B.1 Instrumental Variables

Table D.9 presents the F-statistics of first-stage IV regressions on the demand side (column 1-5) and supply side (column 6). All F-statistics for excluded instruments are larger than 40. Table D.10 lists the included and excluded instrument variables and reports the coefficients and standard errors in the first-stage estimation.

During demand estimation, I consider five endogenous variables: installation price

(p_{jt}), update frequency (a_{jgt}), average rating (x_{2jgt}), and search ranking (E_{jgt}, $ranking_{jgt}$). From the empirical model, search ranking is correlated with η_{jgt} through the app quality index, and update frequency is endogenous because developers know the unobserved demand shocks η_{jgt} at the beginning of the update competition game. Intuitively, price and ratings may also be correlated with unobserved demand shifters such as advertising.

Apart from the two exogenous variations explained in the main text, I use two additional groups of excluded instruments following the existing approach in the literature. First, I construct cost proxies following the approach in Hausman (1996) and Nevo (2000). Specifically, I calculate cost proxies as the average value of the endogenous variable of other apps owned by the same developer in other categories in the same month, provided that the app's developer has other apps in other categories. The exclusion of these cost proxies relies on independent category-specific demand shocks so that the average product characteristics of the same developer in other categories will not be correlated with the unobserved demand shock in the given category through cross-category correlations of demand shocks. To that end, I use month-fixed effects as included instruments on the right-hand-side of Equation (7) to absorb national demand shocks.

Among other cost proxies, two variables constructed based on search results need explanation: i) ratio of *keywords* where the app does not show up in the search results at all, and ii) ratio of *days* that the app does not show up in any search results of the selected keywords in a given month. One example of the underlying cost shifters of these search-result variables is subscribing to app-store-optimization (ASO) service where developers can monitor search rankings of their own apps as well as their competitors across different keywords. Based on such unobserved (to researchers) ASO service, developers may strategically change the keywords where they would like their products to show up over time. Exclusion of these cost shifters relies on that such ASO service does not directly affect the indirect utility of consumers downloading apps.

Second, I use BLP-type instruments that provide exogenous variation for market shares, including i) category-fixed effects; ii) the ratio of multiple-app developers. As discussed in Berry and Haile (2016), together with the cost proxies and markup shifters, these direct market-share shifters, or choice set variation, help identify the random coefficient on the endogenous product characteristic.

On the supply side, developers know unobserved (to researchers) marginal update cost shocks ω_{gt} when choosing update frequency (a_{jgt}). It makes a_{jgt} an endogenous variable in Equation (16). For example, subscribing to ASO services might save the additional cost

of updating an app one more time by exposing the developer to suggestions for changes. Such unobserved advice is positively correlated with update frequency and will bias the coefficient on update upwards in Equation (16).

To deal with endogenous update frequency, I use category-fixed effects and pre-determined app features as excluded instruments. The exclusion restriction relies on the timing assumption that marginal cost shocks ω_{jgt}'s are realized after the update portfolio choice. Thus, the pre-determined app features are determined before the second-stage supply model, such as price, file size, and app titles, among other instrumental variables listed in Table D.10.

B.2 Details on Demand Estimation

Here I provide more details about demand estimation. First, to be clear, the downloads and market shares are observed at app/month level, meanwhile, there are apps that operate in multiple categories simultaneously. However, due to the lack of category-specific downloads data, I take a simplification solution: treat the same app that shows up in multiple categories as a unique app in each category. Then, the market shares are well-defined at app/category/month level, denoted as s_{jgt}, which is to be matched with model-predicted values given by Equation (6) during estimation.

Second, note that $(\rho, \boldsymbol{\lambda})$ are non-linear parameters in Equation (7), since ξ_{jgt-1} is unobserved and $\mu_{jgt}(\cdot)$ is non-linear. To speed up estimation, I get around the non-linearity by subtracting $\rho\tilde{\delta}_{jgt-1}$ from $\tilde{\delta}_{jgt}$. Specifically, I use $\tilde{\delta}_{jgt-1} = \alpha p_{jt-1} + \boldsymbol{x}_{jt-1}\boldsymbol{\beta} + \xi_{jgt-1} - \mu_{jgt-1}(\boldsymbol{\lambda}) + I_g\mu_{0gt-1}(\boldsymbol{\lambda})$ and Equation (7). This returns a linear equation as below, given a guess of $(\sigma, \rho, \boldsymbol{\lambda})$.

$$\tilde{\delta}_{jgt}(\boldsymbol{s}_{gt};\sigma) - \rho\tilde{\delta}_{jgt-1}(\boldsymbol{s}_{gt-1};\sigma) = \alpha\dot{p}_{jt} + \dot{\boldsymbol{x}}_{jt}\boldsymbol{\beta} + \gamma a_{jt} + \eta_{jgt} - \dot{\mu}_{jgt}(\boldsymbol{\lambda}) + I_g\dot{\mu}_{0gt}(\boldsymbol{\lambda}) \tag{B.1}$$

where $\dot{v}_{jt} = v_{jt} - \rho v_{jt-1}$ for any variable y in the above equation. Then, removing the search cost parameters to the left-hand side, I define the dependent variable as below.

$$y^d_{jgt}(\boldsymbol{s}_{gt};\sigma,\rho,\boldsymbol{\lambda}) \equiv \tilde{\delta}_{jgt}(\boldsymbol{s}_{gt};\sigma) - \rho\tilde{\delta}_{jgt-1}(\boldsymbol{s}_{gt-1};\sigma) + \dot{\mu}_{jgt}(\boldsymbol{\lambda}) - I_g\dot{\mu}_{0gt}(\boldsymbol{\lambda}) \tag{B.2}$$

Therefore, Equation (B.1) is equivalent to the following linear regression equation,

given a guess of $(\sigma, \rho, \boldsymbol{\lambda})$.

$$y^d_{jgt}(\boldsymbol{s}_{gt}; \sigma, \rho, \boldsymbol{\lambda}) = \alpha \dot{p}_{jt} + \dot{\boldsymbol{x}}_{jt}\boldsymbol{\beta} + \gamma a_{jt} + \eta_{jgt} \tag{B.3}$$

where η_{jgt} is the econometric error term.

Now I provide details on the GMM estimation of Equation (7). For notation, let $\boldsymbol{x}^0_{jt} := (Age_{jt}, FileSize_{jt}, \#Screenshots_{jt}, DescriptionLength_{jt})$ denote the vector of exogenous time-varying app characteristics. Let $\boldsymbol{x}_j := (AppleOwnership_j, PaidInstallation_j, InAppPurchase_j)$ denote the vector of time-invarying app characteristics. Let $\boldsymbol{z}^{d,e}_{jgt}$ denote the vector of excluded instruments explained in Appendix B.1.

The moment condition based on Equation (7) is given by

$$\mathbb{E}[\boldsymbol{z}^d_{jgt}\eta_{jgt}] = 0, \qquad \boldsymbol{z}^d_{jgt} := (1, \boldsymbol{x}_j, \boldsymbol{x}^0_{jt}, \xi_{jgt-1}, \boldsymbol{z}^{d,e}_{jgt}). \tag{B.4}$$

I take the following steps to compute the GMM objective function, which a sample analog to the moment condition.

1. Given a guess of σ, use BLP inversion to calculate $\tilde{\delta}_{jgt}(\boldsymbol{s}_{gt}; \sigma)$.
2. Given a guess of $(\rho, \boldsymbol{\lambda})$, calculate the dependent variable $y^d_{jgt}(\boldsymbol{s}_{gt}; \sigma, \rho, \boldsymbol{\lambda})$ defined in Equation (B.2), and the independent variables $(\dot{p}_{jt}, \dot{\boldsymbol{x}}_{jt})$ where $\dot{p}_{jt} = p_{jt} - \rho p_{jt-1}$ and likewise for variables in the vector $\dot{\boldsymbol{x}}_{jt}$.
3. Estimate the linear equation (B.3) with the instruments $\boldsymbol{z}^{int}_{jgt} := (1, \boldsymbol{x}_j, \boldsymbol{x}^0_{jt}, \boldsymbol{z}^{d,e}_{jgt})$. This step will return the estimates on the linear parameters $(\alpha, \boldsymbol{\beta}, \gamma)$ given the estimated $(\sigma, \rho, \boldsymbol{\lambda})$.
4. Calculate η_{jgt} based on Equation (B.3).
5. Calculate ξ_{jgt} based on Equations (7) and (3).
6. Calculate the following GMM objective function:

$$\left(\frac{1}{n}\sum_{j,g,t} \boldsymbol{z}^d_{jgt}\eta_{jgt}\right)' \left(\frac{1}{n}\sum_{j,g,t} \boldsymbol{z}^d_{jgt}\boldsymbol{z}^{d\,\prime}_{jgt}\right)^{-1} \left(\frac{1}{n}\sum_{j,g,t} \boldsymbol{z}^d_{jgt}\eta_{jgt}\right) \tag{B.5}$$

For each guess of $(\sigma, \rho, \boldsymbol{\lambda})$, I use the above steps to compute the GMM objective function until the objective function is minimized.

The last detail involves the initial guess of (σ, ρ, λ). To have a good initial guess and thus to speed up estimation, I use the following steps to find the initial guess $(\sigma^0, \rho^0, \lambda^0)$.

1. Search for (ρ^0, λ^0) that minimizes the above GMM objective function while setting $\sigma = 0$, with the initial guess being (0,0,0).

2. Search for σ^0 that minimizes the above GMM objective function while setting $(\rho, \lambda) = (\rho^0, \lambda^0)$, with the initial guess being 0.

Then, for the main estimation, I search for $(\hat{\sigma}, \hat{\rho}, \hat{\lambda})$ that minimizes the above GMM objective function, with the initial guess being $(\sigma^0, \rho^0, \lambda^0)$.

B.3 Details on the Search Ranking Model

Why Rank-Ordered Logistic Model? One advantage of this model lies in the model-implied intuitive correlations between rankings. To see it, a regression equation with search ranking on the left-hand side and an additively separable error term on the right typically assumes that the error terms are independent across observations. Thus, a higher ranking of a product may not result in a lower ranking of others. In contrast, the rank-ordered logistic model assumes independent error terms for a latent variable, namely the *score*, and ranks products according to the score. It implies that a higher probability of one product being ranked first is associated with a lower probability of another product being ranked first.

Specification Details. In the ranking score Equation (9), I consider five exhaustive groups of categories: game categories ($k = Games$), non-game categories that do not have Apple's non-preinstalled apps ($k = 0$), and non-game categories that have 1 or 3 or 7 Apple's non-preinstalled apps ($k = 1, 2, 3$). For θ_{1tk}, only the last three groups matter. For ϑ_k, I normalize the game categories.

The vector z^s_{jgt} includes a_{jgt}, an indicator for paid installation, price, the match between app title (subtitle) and keywords, one-month lagged ratios of 5-star and 4-star ratings and the number of ratings. I argue that these variables capture quite some important factors. For example, they match with what Apple says about its search algorithm: "Apple has agreed that its Search results will continue to be based on objective characteristics like downloads, star ratings, text relevance, and user behavior signals."[59] Some examples

[59] See Apple. 2021. "Apple, US developers agree to App Store updates that will support businesses and

of unobserved score shifters may be consumers' usage of apps, uninstallations, and retention. Finally, there is no constant term because the model fits the ordering of products rather than the exact values of rankings.

Estimation. The conditional log-likelihood for observing product orderings in the data is given by

$$L(\boldsymbol{\theta}|\boldsymbol{x}^s) = \sum_{g,t} \log \mathbb{P}[\boldsymbol{y}_{gt}|\boldsymbol{x}^s_{gt}] \tag{B.6}$$

where the ranking probability $\mathbb{P}[\boldsymbol{y}_{gt}|\boldsymbol{x}^s_{gt}]$ is defined in Equation (10). Notice that, the ranking probability can be interpreted as the product of the probability of the first-ranked product being ranked in position 1, and the probability of the second-ranked product being ranked in position 2 conditional on the first-ranked product being ranked in position 1, until the probability of the last-ranked product being ranked in the last position conditional on all other products being ranked before it (which is equal to 1).

During estimation, for products with no appearance in the top 50 search results across the popular keywords in a given category and month, because they do not have average search ranking conditional on showing up in the top 50 search results, they are ranked below the other apps that have shown up in the top 50 search results, and they are pooled together as ties. Moreover, because we use lagged ratings, there is no estimation result for the first month in the sample.

Alternative Specifications. To illustrate the roles of the ranking shifters, I start with the simplest version of Equation (9) by abstracting away from category-specific and month-specific effects and focusing on average effects. Table D.12 reports the estimates of the baseline search ranking model. First, it summarizes the intuitive roles of the ranking shifters: apps with higher quality, lower installation price, a more matched title or subtitle with keywords, and better previous ratings performance are more likely to receive a higher search ranking. It also shows that updates have a significant positive effect on search ranking, conditional on the effect of quality. In particular, a back-of-envelope calculation gives that a one-percent increase in the quality of an average app leads to a 2.7 percent increase in the latent score. The positive effects of quality and update on search ranking will provide indirect incentive for developers to upgrade their apps.

Second, it also shows evidence for self-preferencing: Apple's ownership has a significantly positive effect on ranking performance, conditional on app quality. In other words,

maintain a great experience for users". August 26. https://www.apple.com/newsroom/2021/08/apple-us-developers-agree-to-app-store-updates/.

the higher quality of Apple's apps is not strong enough to justify their higher search ranking; some self-preferencing of the platform is necessary to explain the dominance of platform-owned products in top positions.

Then, I add monthly flexibility of self-preferencing. Table D.13 shows that there is no month seeing significantly negative self-preferencing. Last, I add category-specific self-preferencing without monthly self-preferencing. Table D.14 shows that there is significant self-preferencing in categories with more than one Apple's non-preinstalled apps, namely, music, entertainment, and utilities; while the self-preferencing in categories with only one Apple's non-preinstalled apps is insignificant. The search ranking model reported in the main text allows the most flexibility regarding self-preferencing as well as the effects of other ranking shifters.

B.4 Details on the Supply Model

B.4.1 The Model of In-App-Purchase-and-Subscription Revenues

While I do not observe in-app prices for purchase and subscription, I calculate the revenues from in-app purchase and subscription by subtracting installation revenues, $p_{jt}Q_{jt}$, from the observed total revenues. Then, I fit a linear equation as below for the in-app-purchase-and-subscription (IAP) revenue, R_{jgt}.

$$\begin{aligned} R_{jt} =& \tau_0 + (\tau_1 + \tau_2 \times game_j) \times Q_{jt} + (\tau_3 + \tau_4 \times game_j) \times Q_{jt}^2 \\ &+ (\tau_5 + \tau_6 game_j) \times a_{jgt} + \lambda_g^R + \lambda_t^{[0]} + game_j \times \lambda_t^{[1]} + e_{jt}^R \end{aligned} \tag{B.7}$$

where $game_j$ indicates game apps, λ_g^R are category-fixed effects, $\lambda_t^{[0]}$ are month-fixed effects, $\lambda_t^{[1]}$ are month-fixed effects interacted with the game indicator, e_{jgt}^R are conditional mean-zero idiosyncratic error terms. Apart from the effect of downloads on revenues, in the case of new in-app-purchase items available along with updates, the model includes a direct revenue effect of update, captured by τ_5 and τ_6. The error term e_{jt}^R, therefore, captures transient, month-to-month variations of IAP revenue shocks specific to an app, category, and month combination. I assume the transitory shock e_{jt}^R is uncorrelated with the explanatory variables.

Define function $R(Q_{jt}, a_{jgt})$ as $R(Q_{lgt}, a_{lgt}; \boldsymbol{\tau}) := (\tau_1 + \tau_2 \times game_j) \times Q_{jgt} + (\tau_3 + \tau_4 \times game_j) \times Q_{jgt}^2 + (\tau_5 + \tau_6 game_j) \times a_{jgt}$. Note that $R(Q_{lgt}, a_{lgt}; \boldsymbol{\tau})$ is the model-

implied variable IAP revenue with respect to updates.

Estimation. With the rich fixed effects in the model, I use Ordinary Least Square (OLS) to estimate Equation (B.7). The estimation sample excludes observations without valid IAP revenues. Specifically, 10.0% of observations of apps with in-app purchases and subscriptions have negative IAP revenues. These negative estimated in-app-purchase-and-subscription revenues may reflect flaws in the estimated revenues from AppTweak. Their total downloads account for 0.67% of total downloads of apps with in-app purchases and subscriptions. I also drop these observations from the sample for supply-model estimation. In the simulations, these apps' update frequency will be fixed.

Meanwhile, I apply the following constraints during estimation to guarantee conditional profit maximization in Nash Equilibrium.

1. first-order increasingness: $\tau_1 > 0$, $\tau_1 + \tau_2 > 0$
2. concavity: $\tau_3 < 0$, $\tau_3 + \tau_4 < 0$

The estimation results are reported in Table D.15. The estimation results for in-app-purchase and in-app-subscription revenues show intuitive results: i) revenues increase with downloads concavely, and ii) updates directly contribute to revenues, especially for game apps. The estimates imply that an average app receives $4.3 from in-app purchases and subscriptions with each new download (consumer) within the sample.

B.4.2 Heuristic Belief Space for Possible Orderings

For markets with strictly more than 5 products, I use a truncated set of possible orders $\mathscr{B}_{\boldsymbol{a}}$ to construct firms' heuristic beliefs on search rankings given updates $\boldsymbol{a}$. Now I explain how $\mathscr{B}_{\boldsymbol{a}}$ is constructed. The most-likely ordering, denoted as $\boldsymbol{y}^*$, is the descending order of products according to the mean ranking scores. It maximizes the ranking probability in Equation (10). For convenience, I copy the ranking probability equation here

$$\mathbb{P}[\boldsymbol{y}] = p_{\boldsymbol{y}(1)} \cdot \frac{p_{\boldsymbol{y}(2)}}{1 - p_{\boldsymbol{y}(1)}} \cdot \frac{p_{\boldsymbol{y}(3)}}{1 - p_{\boldsymbol{y}(1)} - p_{\boldsymbol{y}(2)}} \cdot \cdots \cdot \frac{p_{\boldsymbol{y}(J-1)}}{p_{\boldsymbol{y}(J-1)} + p_{\boldsymbol{y}(J)}} \cdot \frac{p_{\boldsymbol{y}(J)}}{p_{\boldsymbol{y}(J)}}$$

Notice that the numerator of $\mathbb{P}[\boldsymbol{y}]$ is unchanged with $\boldsymbol{y}$. Thus, to find the other highly likely orderings, I only need to enlarge the denominator of $\mathbb{P}[\boldsymbol{y}]$. In particular, I consider the following five tractable ways to enlarge the denominator.[60] For the exposition, I use

[60] Another type of permutation that also marginally enlarges the denominator is to alter the positions of two products with the closest mean ranking scores. However, this will cause the truncation set to be sensitive to marginal changes in update levels. Therefore, it's not considered here.

12345 to denote the first five products in the most likely ordering. Furthermore, I only consider products that show up in the top-50 search results for possible orderings, and denote their number as $J_m^{[1]}$.

1. first-layer enlargement-1: alter the positions of two nearby products. For example, 12345 $\rightarrow$ 21345. $\forall j \in [1, J_m^{[1]} - 1]$.
2. first-layer enlargement-2: alter the positions of two products that only have one other product located between them. For example, 12345 $\rightarrow$ 32145. $\forall j \in [1, J_m^{[1]} - 2]$.
3. second-layer enlargement-1: alter the positions of two nearby products; then for the positioned higher product among the two after the shift, alter its position with the product that's right above it. For example, 12345 $\rightarrow$ 13245 = 13245 $\rightarrow$ 31245. $\forall j \in [2, J_m^{[1]} - 1]$.
4. second-layer enlargement-2: alter the positions of two nearby products; then for the positioned lower product among the two after the shift, alter its position with the product that's right below than it. For example, 12345 $\rightarrow$ 21345 = 21345 $\rightarrow$ 23145. $\forall j \in [1, J_m^{[1]} - 2]$.
5. second-layer enlargement-3: alter the positions of two nearby products; then alter the positions of two nearby products that are right below them. For example, 12345 $\rightarrow$ 21345 $\rightarrow$ 21435. $\forall j \in [1, J_m^{[1]} - 3]$.

The above enlargements have an intuitive interpretation as limited sophistication of developers. Specifically, these enlargements require the developers only consider up to 2 sequential swaps of products based on the most likely ordering. In the last step to construct $\mathscr{B}_{\boldsymbol{a}}$, I conduct the above enlargement until position 30, whenever it applies. The resulting size of $\mathscr{B}_{\boldsymbol{a}}$ is

$$|\mathscr{B}_{\boldsymbol{a}}| = 5 \times \min\{J_m, 30\} - 9 \leq 141$$

B.4.3 Second-Stage Supply Model

Remarks on Equation (16). A technical assumption underlies the first-order-condition approach: the objective function is locally differentiable with respect to updates a_j. However, updates might discretely affect the expected variable profits by changing $\mathscr{B}_{\boldsymbol{a}}$. Thus, I assume that marginal changes in update a_j do not change $\mathscr{B}_{\boldsymbol{a}}$. I argue that this assumption is not strong for two reasons. First, when $\mathscr{B}_{\boldsymbol{a}} = \mathscr{B}$, this assumption is a fact. Second, because the rank-ordered logistic regression model only requires a higher score to be ranked

higher in the most likely ranking rather than a one-to-one mapping from score to ranking, marginal changes of updates a_j typically change the ranking scores without changing the most likely ranking (but it will change the ranking probability and the distribution on $\mathscr{B}_{\boldsymbol{a}}$).[61]

Equation (16) implies an ambiguous effect of self-preferencing on updates. Assuming that an independent app j is boosted up in search results due to eliminating self-preferencing, then the demand curve shifts up, which increases the marginal download from updates, i.e., higher $\left(\frac{\partial Q_j}{\partial a_j}(ranking_j)\right)$ with smaller $ranking_j$. At the same time, the shifted-up demand curve moves the developer rightward along the marginal revenue curve, which will decrease the marginal revenues from downloads, i.e., lower $(0.7p_j + 0.7\frac{\partial R(Q_j,a_j)}{\partial Q_j} + \psi_1 + 2\psi_2 Q_j)$ with smaller $ranking_j$, when the revenue curve is concave in quantity. The ultimate effect of platform self-preferencing on the update frequency of independent apps depends on the dominating force between the two.

Estimation Details. I apply the following constraints during the estimation of Equation (16) to guarantee conditional profit maximization in Nash Equilibrium.

1. Necessary Second-order Condition (negative Hessian Diagonal) for update level of app j, $a_j > 0$:
$$\frac{\partial^2 \pi_f^{II}(a_j, \boldsymbol{a}_{-j}, \omega_j; \phi, \psi)}{\partial a_j^2} < 0$$
2. Non-negative marginal update benefits: $g'(a_{jgt}, \omega_{jgt}; \phi) \geq 0$.
3. Non-negative marginal in-app-advertising profit wrt downloads: $F'(Q_{jgt}; \psi) \geq 0$.
4. Constraints on signs of parameters:
   (a) Higher update, higher costs: $\phi_1 > 0$.
   (b) Increasing and concave in-app-advertising profit with respect downloads: $\psi_1 > 0$, $\psi_2 < 0$.

After the supply estimation of the first-stage game, I run a profit-maximization simulation for each firm to confirm that the observed update frequencies are optimal conditional on observed update portfolios, given the model estimates. Only 1 observation out of 25,326 observations fails the test, whose update frequency is treated as exogenous in the bounds computation and counterfactual simulation.

[61] In the data, there are 9 of the 52,959 observations that violate this assumption. I drop these 9 observations from the supply-model estimation and take its update frequency as exogenous in the counterfactual simulation.

B.4.4 First-Stage Supply Model: Bounds Computation

In order to calculate the upper (lower) bound for an app j in category g and month t, following Fan and Yang (2020), I compute the change of expected second-stage variable profits of its owner f due to adding (dropping) the app j into (from) the update portfolio D^f_{gt}. The expectation is over marginal update cost shocks ω_{gt}. I draw the marginal update cost shocks from their empirical distribution. The empirical distribution is based on the backed-out ω_{jgt} from the estimation of the second-stage supply model in Equation (16). I draw the vector ω_{gt} based on the sparse-grid integration methods (Heiss and Winschel, 2008).

For each drawn vector ω_{gt} and specified update portfolio $\boldsymbol{D}$, I compute the second-stage equilibrium for each cost-shock draw as explained in the following section. This step returns the second-stage variable profits for the given draw. Finally, I take the average of these second-stage variable profits across all cost-shock draws and obtain the expected second-stage variable profits.

B.4.5 Algorithm to Find Second-Stage Equilibrium

Name $log(1+\text{update frequency})$ as "update level". Given a drawn vector of marginal cost shocks, ω, and an update portfolio $\boldsymbol{D}$, I find the equilibrium update levels (including zeros), $\boldsymbol{a}^*(\omega;\boldsymbol{D})$ in the following steps. They ensure that the equilibrium beliefs and update frequencies are reinforcing each other.

1. Initial update levels: $\boldsymbol{a}_0 = (0, \boldsymbol{a}_{-j})$ if $D_j = 0$; $\boldsymbol{a}_0 = (a_j, \boldsymbol{a}_{-j})$ if $D_j = 1$, where
   - if $D_j = 1$ in the data, then a_j takes the value in the data;
   - otherwise, $a_j = \log(1.25)$.
2. Use the truncation method to construct the heuristic belief space $\mathscr{B}_0$ with the following specific steps:
   (a) predict probabilities to be ranked first, $p_{0j} = \exp(\boldsymbol{x}^s_{0j} \cdot \boldsymbol{\theta}^s) / (\sum_{l \in \mathscr{J}} \exp(\boldsymbol{x}^s_{0l} \cdot \boldsymbol{\theta}^s))$, based on the rank-ordered logit model, given $\boldsymbol{a}_0$ as the defining entry of the vector variable $\boldsymbol{x}^s_{0j}$.
   (b) use the probabilities, p_{0j}, to find $\mathscr{B}_0 := \mathscr{B}_{\boldsymbol{a}_0}$.
3. Solve for the equilibrium update levels, $\boldsymbol{a}^*$, based on the ranking set, $\mathscr{B}_{\boldsymbol{a}_0}$.
4. Check if $\mathscr{B}_{\boldsymbol{a}_0} = \mathscr{B}_{\boldsymbol{a}^*}$ or $||\boldsymbol{a}^* - \boldsymbol{a}_0||_\infty < 0.001$.
   - If true, equilibrium found.
   - If false, set $\mathscr{B}_{\boldsymbol{a}_0} = \mathscr{B}_{\boldsymbol{a}^*}$, $\boldsymbol{a}_0 = \boldsymbol{a}^*$, repeat steps 3 and 4.

Appendix C Details on Simulation

C.1 Details on Counterfactual Simulations

This section provides more details on counterfactual simulations than Section 6.

Simulation Range. The simulations involve all categories where, according to the search ranking model, eliminating self-preferencing would have an effect on the market outcome. This criterion rules out three cases: i) categories without Apple's apps; ii) categories that only have pre-installed Apple's apps; iii) categories that have Apple's non-preinstalled apps, but these apps do not occupy top-50 positions in search results. The second case needs further explanation. Because pre-installed apps do not have well-defined downloads, they do not show up in the demand model and thus the search ranking model. Therefore, due to data constraints, the welfare effect of self-preferencing for pre-installed apps in search results is out of the scope of this paper. I argue that non-preinstalled apps are more relevant for self-preferencing in search. Figure E.4 plots the average search ranking of non-preinstalled Apple's apps around July 2019. Comparing Figure E.4 to Figure 1, one can see that Apple's non-preinstalled apps have a sharper drop in search rankings after July 2019 than an average Apple's app.

One market, Music/July, is additionally excluded from full-game simulations, due to a lack of estimated bounds on the constant costs of updates. In particular, during the estimation of the bounds, when I add (or drop) certain products from the vector of update decisions in the Music/July market, I cannot find the corresponding equilibrium, and thus I cannot find the additional variable profits from updating the product. After taking out this market, based on the seven categories that satisfy the above criterion and the two months with the estimated largest self-preferencing parameters, there are 13 markets involved in the full-game simulations. Then, considering the top-5 developers' apps whose cost bounds are calculated, I have 105 independent apps whose update frequencies are subject to changes in the counterfactuals without platform self-preferencing. On average, these active apps account for 45.1% of total downloads in a market. In Appendix Table D.17 and D.18, I allow all independent developers to change update frequency but hold update portfolios fixed in the counterfactual. The results are robust.

Details on Simulation Steps. In the first step during counterfactual simulation, I follow Fan and Yang (2020) to draw the constant costs of update. For each updated app in the data, I have obtained an upper bound for the fixed cost of updating it, $\overline{C}_{jgt}$. For such

an app, I uniformly draw five sunk-cost draws from the range $[0.5\overline{C}_{jgt}, \overline{C}_{jgt}]$. On the other hand, for each app that is not updated in the data, I have obtained a lower bound for the fixed cost of updating it, $\underline{C}_{jgt}$. For such an app, I uniformly draw five sunk-cost draws from the range $[\underline{C}_{jgt}, 5\underline{C}_{jgt}]$.

In the second step where I compute the first-stage equilibrium update portfolios, to deal with the computational burden due to the high dimensionality of the action space when the developers have multiple apps, I follow the heuristic algorithm for finding the best-response product portfolio in Fan and Yang (2020).

Details on Reported Simulation Results. To be clear, Table 8 reports the *expected* search rankings and downloads in equilibrium. The expectation is with respect to potential drawn ω_{jgt}'s for newly updated apps and the possible orderings of products. In particular, during best-response iterations, given a vector of update frequencies in a market, the active developers form a heuristic belief space of possible orderings as described in Section B.4.2 and a probability function on the belief space according to the search ranking model. Based on the equilibrium beliefs, I calculate the expected search rankings and downloads as the average values of rankings and downloads across each possible ordering of apps within the heuristic belief space, weighted by the probability of the ordering. Notice that the expected search ranking and downloads may be different from the observed ones even with the status-quo search algorithm and update frequency. To exclude this confounding factor for welfare effects, I calculate and report the status-quo expected search rankings and downloads in Table 8.

When reporting market outcomes other than update frequency, the market outcomes are calculated using all observed products. For example, in Table 8, the average search rankings of independent apps is the average across all observed independent apps in a given market, and the reported value is the average across the 13 markets. For the top-5 developers' products who are subject to changes in update frequency in counterfactuals, their search rankings may change due to the direct effect of the re-shuffling after eliminating self-preferencing as well as the indirect effect from their new update frequency. For the other apps, their search rankings are subject to the same direct effect and a different indirect effect through the top-5 developers' new update frequency; their own update frequencies are fixed in counterfactuals. Similarly, the reported third-party profits are total profits summed across all observed independent products, based on the inferred variable costs of updates from the supply model. Since inactive independent apps' update frequencies are fixed in the counterfactual, these inferred variable costs of updates cancel with

each other when computing the change in third-party profits.

C.2 Compare with Difference-in-Differences Estimates

Here I compare the structural and difference-in-differences (DiD) estimates of the average treatment effect (ATE) of the search algorithm change in July 2019. To compute the structural estimates of the ATE, I simulate the post-July market outcomes if the self-preferencing parameters were backed to the values in July 2019 – the normalization month in the DiD specification. The simulations involve the same categories as those in Section 6 and the months from August to November 2019. The simulation process is the same as described in Section 6, allowing top-5 developers to change both update portfolios and update frequencies while holding the other apps' update frequency fixed. The structural estimate of the average treatment effect on a market outcome is the change of the market outcome in the simulations compared to the status quo.

Table D.16 reports the simulation results. It shows that the structurally estimated ATEs have the same signs as those estimated from DiD, with smaller magnitudes. The reasons for the discrepancy in magnitudes are three-fold: i) the DiD specifications cover categories where there are only pre-installed apps or below-50 non-preinstalled Apple's apps, while the structural simulations do not; ii) the expected search ranking does not perfectly fit the observed search ranking; iii) the search algorithm change in July 2019 not only changes the self-preferencing parameters but also changes how the other app characteristics affect search ranking, which is not captured by the structural simulations.

C.3 Calculation of Consumer Surplus

This appendix gives details on computing the expected consumer welfare in Equation (20). The expectation is over i) the vector of consumer-app-specific unobserved match-values (ε), ii) the vector of consumer-app-specific search costs ($\boldsymbol{c}$), iii) consumer-specific random coefficients over updates ($\boldsymbol{\sigma}$), iv) the vector of app-specific search rankings ($\boldsymbol{y}$), and v) the vector of marginal cost shocks of updates for newly updated apps ($\boldsymbol{\omega}$).

To compute consumer welfare, I take 10,000 draws of the idiosyncratic unobserved match values (ε) and consumer search costs ($\boldsymbol{c}$). The match values are drawn from the Type-I Extreme Value distribution. The search costs are drawn from the app/category/mon-

th-specific distribution given in Equation (4).[62] On top of these random shocks, there are i) 4 sparse-grid nodes for the one-dimension random coefficient, σ, generated from the sparse-integration method in Heiss and Winschel (2008) with an accuracy level of 4; ii) the truncated set of possible search rankings (y); iii) sparse-grid nodes for J^{add}-dimension marginal cost shocks of updates when the equilibrium update frequencies involve J^{add} apps that are newly updated in simulations (ω). Given each drawn ω, there is an equilibrium vector of update frequency. Given each equilibrium vector of update frequency, each draw of the random coefficients, each vector of possible search rankings in the truncated set, and each draw of match values and search costs, I simulate the optimal sequential search problem in each market(a category/month pair) in the following steps based on the three rules in Weitzman (1979):

1. Implement the *searching rule*. Sort all apps in the markets by reservation values in descending order. This is the order of search by the consumer. The consumer-app specific reservation values, r_{ij}, are computed based on Lemma 1 in Moraga-González, Sándor and Wildenbeest (2023*a*), i.e., $r_{ij} = \delta_{ij} + H_0^{-1}(c_{ij})$, where δ_{ij} is the known utility before search, equating $u_{ij.} - \varepsilon_{ij.}$ in Equation (2); c_{ij} is the consumer-app specific search costs; and the function $H^0(\cdot)$ is given in Equation (4).

2. Implement the *stopping rule*. Along the order of search, compare the highest realized utility to the reservation value of the next app to be searched. The consumer stops search when the highest realized utility is higher than the reservation value of the next app to be searched. The *search costs* incurred by the consumer is the sum of search costs for all the apps that have been searched.

3. Implement the *purchasing rule*. The consumer downloads the app with the highest realized utility among the apps that have been searched. This is also the realized utility of this consumer. The welfare of this consumer is (realized utility - search costs) in the unit of *util*. Divide the consumer welfare by the estimated price coefficient in Table 3 gives the consumer welfare in dollars.

Then I take the average of simulated consumer welfare and multiply it by market size to get market-level consumer surplus. Notice that, by simulating the optimal sequential

[62] A useful detail in computing the consumer-product specific search costs is that I save the search costs across consumers for each evaluated pair of product-position. This significantly saves computational time.

search problem, I have computational market shares derived from the discrete choices of simulated consumers. Therefore, there might be some distance between the computational market shares and analytical market shares derived from the closed-form choice probability. This can be used to measure the accuracy of the consumer surplus measurement. Table D.20 presents the computational error during the computation of consumer surplus. It shows that all computational errors are smaller than 0.4%.

Appendix D Additional Tables

Table D.1: App Categories in the Sample

Non-Game Categories		Game Categories	
(1)	Book	(23)	Games-Action
(2)	Business	(24)	Games-Adventure
(3)	Education	(25)	Games-Arcade
(4)	Entertainment	(26)	Games-Board
(5)	Finance	(27)	Games-Card
(6)	Food & Drink	(28)	Games-Casino
(7)	Health & Fitness	(29)	Games-Family
(8)	Lifestyle	(30)	Games-Music
(9)	Medical	(31)	Games-Puzzle
(10)	Music	(32)	Games-Racing
(11)	Navigation	(33)	Games-Role Playing
(12)	News	(34)	Games-Simulation
(13)	Newsstand	(35)	Games-Sports
(14)	Photo & Video	(36)	Games-Strategy
(15)	Productivity	(37)	Games-Trivia
(16)	Reference	(38)	Games-Word
(17)	Shopping		
(18)	Social Networking		
(19)	Sports		
(20)	Travel		
(21)	Utilities		
(22)	Weather		

Table D.2: List of Pre-installed Apps and Data Availability

App Name	Data Available?	Category	App Name	Data Available?	Category
App Store	0		Measure	1	Utilities
Calculator	1	Utilities	Messages	0	
Calendar	0	Productivity	Music	1	Music
Camera	0		News	1	News
Clock	0		Notes	0	Productivity
Compass	0	Navigation	Numbers	1	Productivity
Contacts	1	Utilities	Pages	1	Productivity
FaceTime	1	Social Networking	Passbook	0	
Files	1	Utilities	Phone	0	
Find My Friends	1	Social Networking	Photos	0	
Find My iPhone	1	Utilities	Podcasts	1	Entertainment
Game Center	0		Reminders	0	Productivity
Health	0		Safari	0	
Home	1	Lifestyle	Settings	0	
iBooks	1	Book	Stocks	1	Finance
iCloud Drive	0		Tips	1	Utilities
iMovie	1	Photo & Video	TV	1	Entertainment
iTunes Store	1	Entertainment	Videos	0	
iTunes U	1	Education	Voice Memos	1	Utilities
Keynote	1	Productivity	Wallet	0	Finance
Mail	0	Productivity	Watch	0	Utilities
Maps	1	Navigation	Weather	1	Weather

Notes: For apps without data availability, there is category information if it shows up on the Apple App Store.

Table D.3: Additional Summary Statistics

Variable	Mean	Median	SD	Min	Max	Obs
Brand Keyword?	0.34	0	0.47	0	1	1,340[a]
Search Volume	47.68	48	13.88	5	100	718,500[b]
Title Match	0.02	0.02	0.03	0	0.16	56,570[c]
Subtitle Match	0.02	0.01	0.03	0	0.17	56,570
%5-star ratings	0.73	0.77	0.16	0	1	56,570
%4-star ratings	0.12	0.11	0.06	0	1	56,570
#Pre-installed Apps	0.63	0	1.32	0	7	38[d]
Among categories with pre-installed apps:						
min. Top-50	0.83	1	0.37	0	1	299[e]
min. Search Ranking	6.93	3.13	9.31	0	50	299

[a] These observations are at keyword level.
[b] These observations are at keyword/day level.
[c] These observations are at app/category/month level.
[d] These observations are at category level.
[e] These observations are at category/month level.

Table D.4: Variation of App Characteristics: Overall v.s. Within Apps

Variable	SD			Range		
	overall	within app	ratio	overall	within app	ratio
Update Frequency	1.00	0.48	0.49	11	1.6	0.15
Average Rating	0.55	0.09	0.15	4	0.23	0.06
Installation Price|Paid	4.69	0.32	0.07	99.99	0.82	0.01
File Size	411.01	17.84	0.04	4095.28	49.41	0.01

Notes. Overall SD and Range are taken from Table 1. Given a variation measurement (standard deviation or range), the average within-app variation is the average of X_j, where $X_j =$ (the variation measurement of x_{jt} across months indexed by t). For update frequency, x_{jt} is average x_{jgt} across categories indexed by g. The ratios between within-app variation and overall variation show that update frequency changes more within apps than how the other app characteristics change within apps.

Table D.5: Variation of Product Offerings by Multiple-Category Developers

Variable	Within-Developer SD[a]	SD of Developers' Within-Market SD[a]
Update Frequency	0.73	0.52
Price ($)	0.34	0.75
Average Rating	0.09	0.13
Ever Top50	0.15	0.17

[a]The reported figure is the average SD cross categories within developer/month pairs. Specifically, the reported figure is the average of X_{ft}, where $X_{ft} =$ (standard deviation of $(1/\#\mathscr{J}_{fgt})\sum_{j\in\mathscr{J}_{fgt}} x_{jgt}$ across categories indexed by g), where $\mathscr{J}_{fgt}$ is the set of apps owned by developer f in category g/month t. The column shows that developers' product offerings are different across markets. Such variation is exploited to construct cost proxies as excluded instruments.

The reported figure is the average SD cross developers' within-category/month SD. Specifically, it is the average of Y_{gt}, where $Y_{gt} =$ (standard deviation of $X_{f(j)t}$ across apps $j \in \mathscr{J}_{gt}$), where $X_{f(j)t}$ is the X_{ft} of the developer of app j, and $\mathscr{J}_{gt}$ is the set of apps in category g/month t. The column shows that competing developers in the same market are different in how diversified their product offerings are across markets. This variation is exploited to construct indirect markup shifters as excluded instruments.

Table D.6: Summary Statistics Before and After the Search Algorithm Change

	Categories with Apple				Categories without Apple			
	Before Jul.2019		After Jul.2019		Before Jul.2019		After Jul.2019	
Variable	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Panel A. App Data								
Downloads	0.07	0.22	0.06	0.26	0.06	0.23	0.04	0.14
Price	1.95	3.74	2.00	3.67	1.75	3.77	1.90	3.89
log(1+Update Freq.)	0.40	0.48	0.39	0.49	0.36	0.45	0.33	0.45
Average Rating	4.32	0.62	4.30	0.63	4.43	0.52	4.40	0.58
File Size	98.49	143.90	93.01	139.30	302.80	503.80	302.60	512.50
#Screenshots	5.60	1.94	5.68	1.96	5.72	1.81	5.83	1.87
Description Length	2.40	1.05	2.38	1.06	2.06	1.00	2.03	1.00
Search Ranking |Top 50	24.08	11.71	23.95	11.82	24.07	11.73	24.20	11.67
Obs |Top 50	1,552		3,103		2,266		4,651	
Obs(app/category/month)	2,709		5,784		3,729		8,219	
Panel B. Conversion Rates Data								
Rates	0.06	0.05	0.09	0.09	0.04	0.03	0.06	0.06
Type	0.35	0.48	0.39	0.49	0.21	0.41	0.28	0.45
Obs(type/category/month)	46		99		58		127	

Notes. "Type" indicates whether the average conversion rate is average across paid apps, given a category/month pair. Alternatively, the average conversion rate is average across free apps.

Table D.7: Summary Statistics: Search Volume and Characteristics of Apps in the Search Results of Keywords

	Top50 Search Results			Top1 Search Results		
Variable	Obs	Mean	SD	Obs	Mean	SD
Search Volume	41,974	48.29	13.88	18,789	48.90	14.07
Brand-name Keywords?	41,974	0.34	0.47	18,789	0.42	0.49
Pre-installed	41,974	0.02	0.08	18,789	0.05	0.22
Apple	41,974	0.02	0.09	18,789	0.06	0.24
Update Level	41,908	0.60	0.33	17,858	0.60	0.50
Average Rating	41,908	4.55	0.22	17,858	4.55	0.34
Age(month)	41,908	50.27	19.32	17,858	53.72	28.03
File Size (GB)	41,908	0.24	0.29	17,858	0.27	0.47
#Screenshots	41,908	5.78	1.29	17,858	5.86	1.94
Description Length(1,000 characters)	41,908	2.44	0.63	17,858	2.46	0.98
Offer In-app-purchase	41,908	0.93	0.19	17,858	0.92	0.27
Paid Installation	41,908	0.11	0.23	17,858	0.13	0.33
Price	41,908	0.53	1.57	17,858	0.73	2.68

Notes. App characteristics are reported as average levels across the observed apps in the top50 or top1 search results for a given keyword/category/month combination.

Table D.8: Estimation Results: Known App Characteristics before Search

	Top50 Search Results		Top1 Search Results	
	Search Volume			
Variable	(1)	(2)	(1)	(2)
All Pre-install?	1.18	-2.00***	5.86**	-3.14***
	(1.97)	(0.74)	(2.29)	(1.04)
Update Level	1.39***	0.46***	1.13***	0.16**
	(0.28)	(0.08)	(0.22)	(0.07)
Average Rating	0.33	0.06	0.24	0.30*
	(0.33)	(0.11)	(0.29)	(0.17)
Age(month)	0.01***	-0.01***	0.02***	-0.02***
	(0.00)	(0.00)	(0.00)	(0.00)
File Size(GB)	-0.95***	0.22	-0.34	0.06
	(0.37)	(0.17)	(0.26)	(0.20)
#Screenshots	0.79***	0.05**	0.40***	0.01
	(0.07)	(0.02)	(0.05)	(0.03)
Description Length(1,000 characters)	-1.11***	-0.09*	-0.50***	0.14
	(0.14)	(0.05)	(0.11)	(0.09)
Offer In-app-purchase	3.14***	-1.75***	3.66***	-3.00***
	(0.63)	(0.35)	(0.51)	(0.66)
Paid Installation	-12.61***	-0.71**	-8.76***	-2.68***
	(0.57)	(0.34)	(0.46)	(0.85)
Price	-0.21***	0.00	-0.09*	0.15***
	(0.08)	(0.04)	(0.05)	(0.04)
Apple	10.99***	-1.61***	12.31***	-2.35***
	(1.52)	(0.51)	(1.08)	(0.84)
Pre-install	-2.58	1.49**	-10.23***	3.21***
	(1.73)	(0.60)	(2.06)	(0.98)
Brand-name Keywords?	5.33***		5.74***	
	(0.13)		(0.18)	
Constant	40.32***	50.05***	40.36***	51.22***
	(1.67)	(0.67)	(1.38)	(0.91)
Category-Month FE	YES		YES	
Keyword-Category FE		YES		YES
Month FE		YES		YES
Observations	41,974	41,964	18,787	18,708
R-squared	0.26	0.96	0.35	0.97
Mean level	48.29		48.90	

Notes. Robust standard errors in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$. The dependent variable is the monthly average search volume of keywords. The independent variables are the average characteristics of apps shown in the top-50 search results (columns 1 and 2) or the top-1 search result (columns 3 and 4) of the keyword in the month. The results shed light on whether consumers' choice of keywords is correlated with the search results of the keywords.

Table D.9: First-stage IV Regression Results: F statistics

	Demand					Supply
Variables	(1) Price	(2) Average Rating	(3) $\log(1+\text{Update Frequency})$	(4) Search Ranking $\times$Ever Top50	(5) Ever Top50	(6) $\log(1+\text{Update Frequency})$
F^a	591.6	84.04	322.8	335.8	1011	1645
Excluded F^b	51.28	81.76	68.89	68.59	217.5	40.47

[a] On the demand side, the reported F is F(84,52874). On the supply side, the reported F is F(73,25252)
[b] On the demand side, the reported excluded F is F(56,52874). On the supply side, the reported excluded F is F(50,25252).

Table D.10: First-Stage IV Regression Results

	Demand					Supply
Variables	(1) Price	(2) Average Rating	(3) log(1 + Update Frequency)	(4) Search Ranking ×Ever Top50	(5) Ever Top50	(6) log(1 + Update Frequency)
Included Instruments						
Apple	-0.53***	-0.14**	-0.07**	1.01	0.24***	
	(0.14)	(0.06)	(0.03)	(1.12)	(0.03)	
Paid Installation?	4.22***	-0.11***	-0.27***	-10.41***	-0.41***	-0.07***
	(0.05)	(0.01)	(0.01)	(0.18)	(0.01)	(0.01)
Offer In-app-purchase?	-0.56***	0.04***	0.10***	1.12***	0.03***	0.10***
	(0.06)	(0.01)	(0.00)	(0.18)	(0.01)	(0.01)
log(Age) (month)	0.13***	0.01*	-0.02***	-0.28**	0.02***	
	(0.02)	(0.00)	(0.00)	(0.11)	(0.00)	
log(File Size)(MB)	0.49***	0.02***	0.04***	0.26***	0.02***	
	(0.02)	(0.00)	(0.00)	(0.05)	(0.00)	
#Screenshots	-0.06***	0.02***	0.03***	0.03	-0.00	
	(0.01)	(0.00)	(0.00)	(0.03)	(0.00)	
log(1 + Description Length)(1,000 characters)	0.30***	0.04***	0.01***	0.84***	0.04***	
	(0.03)	(0.00)	(0.00)	(0.09)	(0.00)	
Constant	-3.99***	3.74***	0.14***	7.15***	0.08***	0.44***
	(0.25)	(0.05)	(0.04)	(1.08)	(0.03)	(0.04)
Month-FE	YES	YES	YES	YES	YES	YES
Excluded Instruments						
Search Ranking Shifters:						
Title Match	1.61***	0.51***	0.10	39.95***	4.40***	0.20**
	(0.58)	(0.09)	(0.08)	(2.43)	(0.06)	(0.09)
Subtitle Match	0.81*	1.62***	0.96***	47.98***	2.35***	0.53***
	(0.45)	(0.08)	(0.08)	(2.53)	(0.06)	(0.10)
Apple×Post	0.37***	0.04	0.01	9.78***	0.10**	
	(0.14)	(0.09)	(0.04)	(1.38)	(0.04)	
AppleCompetitor×Post	0.18***	-0.01	-0.00	-0.05	-0.01	
	(0.06)	(0.01)	(0.01)	(0.24)	(0.01)	
Cost Proxies: Average Value across Other Apps Owned by the Same Developer in the Other Categories in the Same Month						
Multiple-Category Developer?	0.05	-2.16***	-0.14***	-1.21	-0.41***	0.04
	(0.23)	(0.08)	(0.04)	(1.46)	(0.05)	(0.07)
#Category Where the Developer Operate in the Month	0.065***	-0.01***	-0.00***	-0.08***	-0.00***	
	(0.01)	(0.00)	(0.00)	(0.03)	(0.00)	
Price	0.19***	0.01***	-0.00*	0.03	-0.00	-0.01***
	(0.02)	(0.00)	(0.00)	(0.04)	(0.00)	(0.00)
log(1 + Update Frequency)	0.12***	-0.02***	0.16***	-0.10	-0.02***	
	(0.02)	(0.00)	(0.01)	(0.14)	(0.00)	
Average Rating	0.13***	0.47***	-0.02**	0.16	0.10***	-0.02
	(0.05)	(0.02)	(0.01)	(0.25)	(0.01)	(0.01)
Top50 in Search	0.07	-0.00	0.04*	1.16	-0.09***	
	(0.18)	(0.04)	(0.02)	(0.96)	(0.03)	
Ratio. Keywords without Search Results	-1.57***	0.15***	0.05**	0.32	0.19***	
	((0.10)	(0.02)	(0.02)	(0.66)	(0.02)	
Ratio. Days without Search Results	-0.55***	-0.01	0.02	-0.08	-0.13***	
	(0.15)	(0.03)	(0.02)	(0.93)	(0.03)	
Indirect Markup Shifters: Average Value of Cost Proxies of Apps Owned by Competitors in the Same Category and Month						
Price	-0.48***	0.02	0.04*	0.62	0.03**	0.05***
	(0.09)	(0.02)	(0.02)	(0.58)	(0.02)	(0.02)
log(1 + Update Frequency)	-1.15***	0.01	-0.08	0.75	0.03	
	(0.22)	(0.05)	(0.04)	(1.41)	(0.04)	
Average Rating	0.95**	0.15**	-0.24***	-2.27	-0.12**	-0.04***
	(0.37)	(0.07)	(0.06)	(1.88)	(0.05)	(0.01)
Top50 in Search	-2.79*	-0.22	0.57**	21.69***	1.28***	
	(1.64)	(0.30)	(0.26)	(8.33)	(0.23)	
Ratio. Keywords without Search Results	1.79***	-0.56***	0.60***	-14.27***	-1.00***	
	(0.67)	(0.16)	(0.16)	(4.37)	(0.12)	
Ratio. Days without Search Results	-2.45*	-0.55**	0.22	14.61**	0.86***	
	(1.48)	(0.26)	(0.24)	(7.42)	(0.20)	
BLP-Style Instrument:						
Ratio. Multiple-App Developers	0.83***	0.11**	0.06	3.60**	0.08*	0.14**
	(0.27)	(0.06)	(0.05)	(1.46)	(0.04)	(0.07)
Supply Model: Pre-Determined Characteristics						
Age (month)						0.00
						(0.00)
log(File Size)(MB)						0.05***
						(0.00)
Price						0.00**
						(0.00)
Ratio. Paid Apps						0.12*
						(0.07)
Category-FE	YES	YES	YES	YES	YES	YES
Observations	52,959	52,959	52,959	52,959	52,959	25,325
Adjusted R-squared	0.37	0.18	0.30	0.64	0.75	0.07

Notes. Robust standard errors in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$.

Table D.11: Estimates of the Search Ranking Model

Variables	Parameter	Standard Error
Quality	0.126	0.013
Squared Quality ($\times$ 0.1)	−0.065	0.010
Squared Quality ($\times$ 0.01)	−0.046	0.005
log(1+Update Frequency)	0.073	0.011
Apple $\times$ Non-Game Group-1	0.636	0.213
Apple $\times$ Non-Game Group-2	1.981	0.184
Apple $\times$ Non-Game Group-3	1.534	0.207
Paid Installation $\times$ Non-Game Group-0	−0.436	0.024
Paid Installation $\times$ Non-Game Group-1	−0.643	0.048
Paid Installation $\times$ Non-Game Group-2	−0.414	0.079
Paid Installation $\times$ Non-Game Group-3	−0.627	0.053
Paid Installation $\times$ Game	−0.740	0.034
Price $\times$ Non-Game Group-0	−0.020	0.003
Price $\times$ Non-Game Group-1	−0.027	0.003
Price $\times$ Non-Game Group-2	−0.063	0.004
Price $\times$ Non-Game Group-3	−0.035	0.002
Price $\times$ Game	−0.013	0.003
Title Match $\times$ Non-Game Group-0	15.345	0.448
Title Match $\times$ Non-Game Group-1	17.581	0.721
Title Match $\times$ Non-Game Group-2	16.478	0.756
Title Match $\times$ Non-Game Group-3	15.766	1.204
Title Match $\times$ Game	11.424	0.354
Subtitle Match $\times$ Non-Game Group-0	7.740	0.518
Subtitle Match $\times$ Non-Game Group-1	11.567	0.368
Subtitle Match $\times$ Non-Game Group-2	7.894	0.727
Subtitle Match $\times$ Non-Game Group-3	1.981	0.974
Subtitle Match $\times$ Game	4.434	0.345
Lagged % 5-stars $\times$ Non-Game Group-0	0.876	0.052
Lagged % 5-stars $\times$ Non-Game Group-1	0.430	0.092
Lagged % 5-stars $\times$ Non-Game Group-2	0.614	0.105
Lagged % 5-stars $\times$ Non-Game Group-3	0.628	0.088
Lagged % 5-stars $\times$ Game	0.520	0.042
Lagged % 4-stars $\times$ Non-Game Group-0	1.303	0.107
Lagged % 4-stars $\times$ Non-Game Group-1	1.265	0.224
Lagged % 4-stars $\times$ Non-Game Group-2	1.691	0.364
Lagged % 4-stars $\times$ Non-Game Group-3	−0.463	0.386
Lagged # Ratings $\times$ Non-Game Group-0	1.687	0.124
Lagged # Ratings $\times$ Non-Game Group-1	0.367	0.033
Lagged # Ratings $\times$ Non-Game Group-2	0.012	0.038
Lagged # Ratings $\times$ Non-Game Group-3	6.942	0.527
Lagged # Ratings $\times$ Game	0.581	0.041
Apple $\times$ Months (Omit July 2019)	YES	
Observations	52,959	
Average Latent Score	-0.393	
Pseudo R-sq	0.069	

Notes: Clustered standard errors at category-month level. There are five groups of categories: Game apps, Non-Game Group-0 (non-game categories without Apple's non-preinstalled apps), Non-Game Group-1/2/3 (non-game categories with 1/3/7 Apple's non-preinstalled apps, covering all categories with Apple's non-preinstalled apps). Coefficients on Apple $\times$ Months (omit July 2019) are reported in Figure 3.

Table D.12: Estimates of the Baseline Search Ranking Model

Variables	Parameter	Standard Error
Apple	0.678	0.102
Quality	0.144	0.012
log(1+Update Frequency)	0.071	0.011
Paid Installation	−0.566	0.020
Price	−0.023	0.002
Title Match with Keywords	14.321	0.273
Subtitle Match with Keywords	6.924	0.265
One-month Lagged %5-star Ratings	0.741	0.032
One-month Lagged %4-star Ratings	0.710	0.084
One-month Lagged #Ratings	0.269	0.037
Squared and Cubed Quality	YES	
Observations	52,959	
Average Latent Score	-0.437	
Pseudo R-sq	0.065	

Notes: Clustered standard errors at category-month level.

Table D.13: Search Ranking Estimation: Self-Preferencing Estimates by Month

Variables	Parameter	Standard Error
Quality	0.124	0.013
log(1+Update Frequency)	0.072	0.011
Apple × May 2018	0.141	0.711
Apple × Jun 2018	0.016	0.819
Apple × Jul 2018	0.911	0.470
Apple × Aug 2018	0.389	0.636
Apple × Sep 2018	0.771	0.692
Apple × Oct 2018	0.316	0.544
Apple × Nov 2018	0.539	0.521
Apple × Dec 2018	−0.117	0.415
Apple × Jan 2019	0.522	0.429
Apple × Feb 2019	0.638	0.410
Apple × Mar 2019	−0.164	0.499
Apple × Apr 2019	0.715	0.474
Apple × May 2019	0.956	0.479
Apple × Jun 2019	1.589	0.403
Apple × Jul 2019	1.314	0.253
Apple × Aug 2019	0.718	0.315
Apple × Sep 2019	0.500	0.311
Apple × Oct 2019	0.621	0.306
Apple × Nov 2019	0.671	0.314
Apple × Dec 2019	0.543	0.285
Apple × Jan 2020	0.413	0.262
Apple × Feb 2020	0.395	0.237
Squared and Cubed Quality	YES	
Paid Installation × Category Groups	YES	
Price × Category Groups	YES	
Title Match × Category Groups	YES	
Subtitle Match × Category Groups	YES	
Lagged %5-star Rating × Category Groups	YES	
Lagged %4-star Rating × Category Groups	YES	
Lagged #Ratings × Category Groups	YES	
Observations	52,959	
Average Latent Score	-0.377	
Pseudo R-sq	0.069	

Notes: Clustered standard errors at category-month level. There are five groups of categories: Game apps, Non-Game Group-0 (non-game categories without Apple's non-preinstalled apps), Non-Game Group-1/2/3 (non-game categories with 1/3/7 Apple's non-preinstalled apps, covering all categories with Apple's non-preinstalled apps).

Table D.14: Search Ranking Estimation: Self-Preferencing Estimates by Group of Categories

Variables	Parameter	Standard Error
Quality	0.126	0.013
log(1+Update Frequency)	0.074	0.011
Apple × Non-Game Group-1	0.061	0.120
Apple × Non-Game Group-2	1.377	0.098
Apple × Non-Game Group-3	0.693	0.274
Squared and Cubed Quality	YES	
Paid Installation × Category Groups	YES	
Price × Category Group	YES	
Title Match × Category Groups	YES	
Subtitle Match × Category Groups	YES	
Lagged %5-star Rating × Category Groups	YES	
Lagged %4-star Rating × Category Groups	YES	
Lagged #Ratings × Category Groups	YES	
Observations	52,959	
Average Latent Score	-0.392	
Pseudo R-sq	0.069	

Notes: Clustered standard errors at category-month level. There are five groups of categories: Game apps, Non-Game Group-0 (non-game categories without Apple's non-preinstalled apps), Non-Game Group-1/2/3 (non-game categories with 1/3/7 Apple's non-preinstalled apps, covering all categories with Apple's non-preinstalled apps).

Table D.15: Estimates of the In-App-Purchase-and-Subscription Revenue Model

Variables	Parameter	Standard error
Downloads (million)	5.619	0.641
Downloads × Game	−2.765	0.701
Squared Downloads	−0.401	0.196
Squared Downloads × Game	0.163	0.231
log(1+Update Frequency)	0.078	0.030
log(1+Update Frequency) × Game	0.400	0.040
Constant	0.052	0.188
Category-FE	YES	
Month-FE	YES	
Month-FE × Game	YES	
Average Marginal Revenue ($)	4.298	
Observations	37,382	
adjusted R-sq	0.27	

Notes: Marginal revenue is with respect to downloads.

Table D.16: Compare Structural Estimates with Difference-in-Differences Estimates of Average Treatment Effects of the Search Algorithm Change

ATE	$\log(1+\text{Update Frequency})$	$\log(\text{Search Ranking})$	$\log(\text{Downloads})$
Structural Estimates	0.0078	-0.0123	0.0032
DiD Estimates	0.0212	-0.0355	0.2210

Notes. Difference-in-Differences (DiD) estimates are taken from Table 2. Structural estimates are computed from differences between market outcomes with and without the search algorithm change in categories with Apple's apps during the same post-change period as the DiD specification. When simulating the counterfactual without the search algorithm change, I set the self-preferencing parameter back to the value in July 2019, the normalization month in the DiD specification, and allow top-5 developers to change both update portfolios and update frequencies while holding the other apps' update frequency fixed. Thus, the reported structural estimates of the ATE for $\log(1+\text{update frequency})$ is conditional on these top-5 developers' apps.

Table D.17: Partial-Game Simulation: Effects of Self-preferencing on Update Frequency without Update Portfolio Adjustment

	obs	Status-quo	Shut-down	Percentage Change(%)			
		mean	mean	mean	std	min	max
Average Update Freq.	14	1.31	1.32	0.57	1.11	0.00	3.57
Update Freq.	535	1.33	1.34	0.28	3.72	-17.22	65.31

Notes. In the partial game, all independent apps with valid profit functions and updates may change their update frequencies, while update portfolios are holding fixed. Update frequency is the monthly number of updates weighted by the length of release notes. Average update frequency is the average value of update frequency across active independent apps in the market. Each market is a category/month pair. For the status-quo case, there is the estimated self-preferencing. For the shut-down case, there is no self-preferencing.

Table D.18: Partial-Game Simulation: Effects of Self-preferencing on Search Rankings, Installations and Welfare without Update Portfolio Adjustment

	Variable	Status-quo	Shut-down	Mean Δ	Mean $\%\Delta$
(1)	Average Search Rankings	40.23	40.23	0.00	0.00
(2)	- Independent apps	40.95	40.43	-0.52	-1.27
(3)	- Apple's apps	15.32	32.79	17.46	175.02
(4)	Total Installations (million)	6.52	6.68	0.16	1.48
(5)	- Independent apps	6.43	6.61	0.18	1.97
(6)	- Apple's apps	0.09	0.07	-0.02	-23.10
(7)	Consumer Surplus (million \$)	300.21	301.15	0.94	0.28
(8)	- Search Costs	23.56	23.50	-0.06	0.16
(9)	- Choice Quality	323.77	324.65	0.88	0.25
(10)	Third-Party Profits (million \$)	48.30	48.64	0.35	0.70
Number of Markets			14		

Notes. In the partial game, all independent apps with valid profit functions and positive update frequencies may change their update frequencies, while update portfolios are holding fixed. For the status-quo case, there is the estimated self-preferencing. For the shut-down case, there is no self-preferencing.

Table D.19: Percentage change of Update Frequency without Platform Self-Preferencing (%)

Case	Obs.	Min	Q1	Q2	Q3	Max
Positive Update Effect	28	0.00	0.15	0.53	1.42	65.61
Negative Update Effect	17	-21.42	-0.70	-0.09	-0.01	0.00

Notes: Each observation is a combination of app/category/month. There are 60 observations that do not see a change in update frequency without platform self-preferencing.

Table D.20: Computational Error in Consumer Surplus

	Max Relative L2 Norm		Max Relative L-infinity Norm	
	Status-quo	Shut down	Status-quo	Shut down
Error (%)	0.35	0.35	0.28	0.22

Notes. Figures are the percentage of computational errors with respect to analytical market shares. The computational error is the distance between the computational market shares from the simulated optimal sequential search model for computing consumer surplus and analytical market shares. The distance is measured by the maximum relative L2 norm in the left panel, and the maximum relative L-infinity norm in the right panel, across all simulated observations.

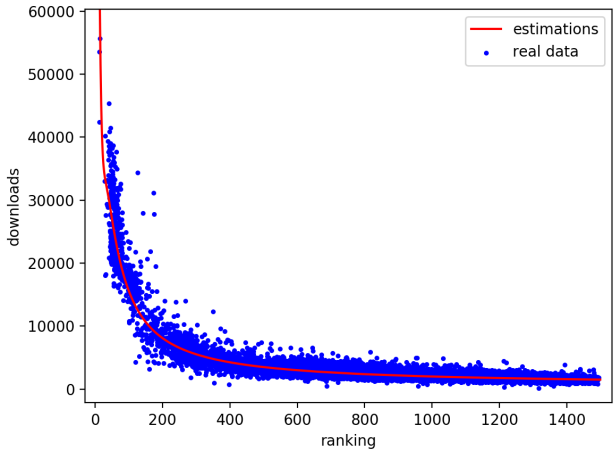
Table D.21: Effects of Self-Preferencing on Revenues and Profits

Variable	Status-quo	Shut-down	Mean Δ	Mean $\%\Delta$
Panel A. Subsample: active apps				
Revenue	11.19	11.59	0.40	2.13
Revenue + IAA Profits	11.58	11.99	0.41	2.13
Profits	6.76	6.95	0.19	2.67
Mean %active apps		9.20		
Panel B. Full Sample: all apps				
Revenue	24.52	25.06	0.54	1.55
- Independent apps	24.51	25.05	0.54	1.58
- Apple's apps	0.01	0.01	0.00	−18.88
Revenue + IAA Profits	25.35	25.90	0.55	1.53
- Independent apps	25.32	25.88	0.56	1.58
- Apple's apps	0.03	0.02	−0.01	−22.49
Number of Markets		13		

Notes. Both panels report average values across 13 markets. Profits are subject to a constant that does not change with downloads. The top panel only considers apps that may adjust updates in counterfactuals, all of which are independent apps. The bottom panel considers all observed apps. The ratio of active apps ranges from 5.56% to 16.16% across the 13 markets, with a median of 7.53%. Considering the 30% commission rates, the platform's revenue increases by 0.16 million dollars with the elimination of the estimated self-preferencing.

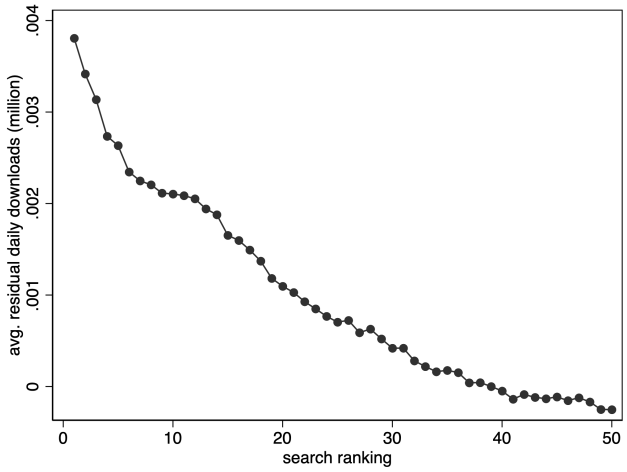
Appendix E Additional Figures

Figure E.1: From AppTweak: Fitness of Estimated Downloads for Actual Downloads of Apps in All-Categories Top Charts in the US Market



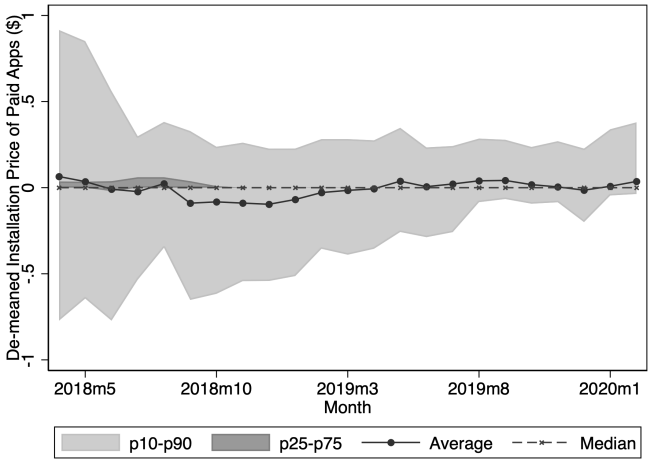
Notes: The figure is from AppTweak, source: https://www.apptweak.com/en/aso-blog/introducing-worldwide-ios-download-and-revenue-estimates.

Figure E.2: Weighted Average Residual Downloads and Search Ranking



Notes. The x-axis is keyword/day-specific search ranking. The y-axis is the weighted average residual downloads. The downloads are observed at app/day level. The residuals are from category-fixed effects, daily fixed effects, free/paid indicator, and installation price. The average is across apps, keywords, and days for a given search ranking. The weight is based on keyword/day-specific search volume, an integer between 5 and 100, constructed by Apple to index how many consumers search for the keyword.

Figure E.3: Within-App De-Meaned Installation Price of Paid Apps ($)



Notes. The figure shows that there is limited within-app price change over time. The median within-app deviation from the mean price is zero, the average is between -$0.10 and $0.06, and the second and third quartiles are tight surrounding zero. Overall, installation prices are sticky over time in the data.

Figure E.4: Average Search Rankings of Non-preinstalled Apple's Apps around the Search Algorithm Change in July 2019.

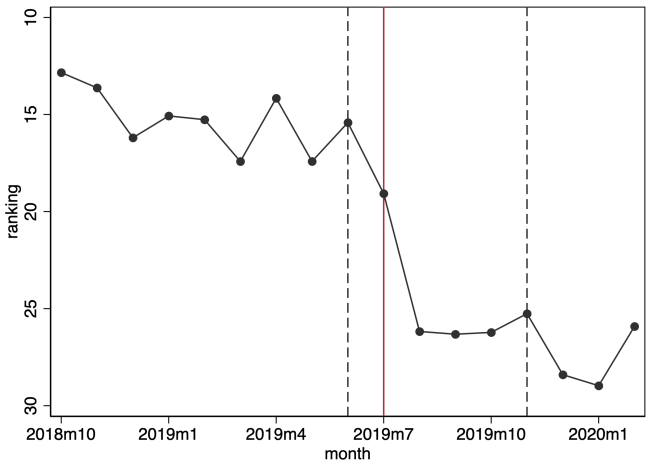
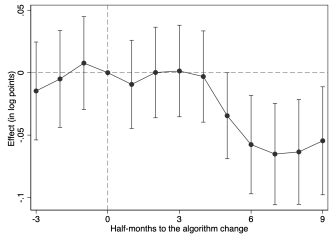
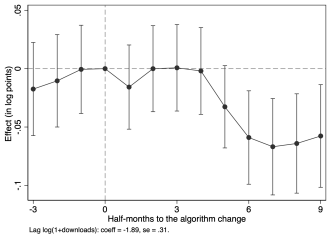


Figure E.5: Effect of Reduced Dominance of Platform-owned Products on Independent Apps, by Half-month from Search Algorithm Change

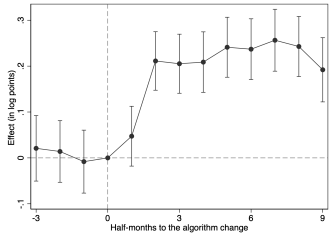
(a) Search Ranking



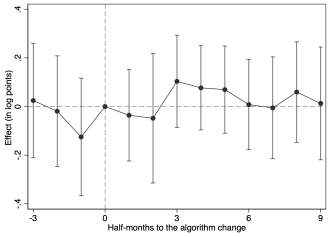
(b) Search Ranking (with lagged downloads)



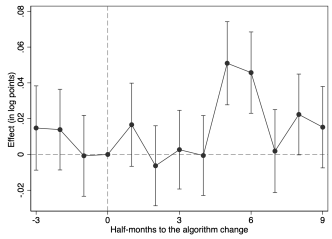
(c) Downloads



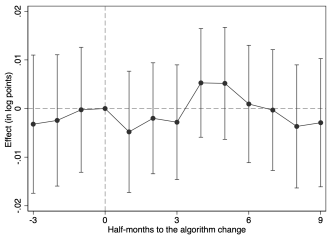
(d) Conversion Rate



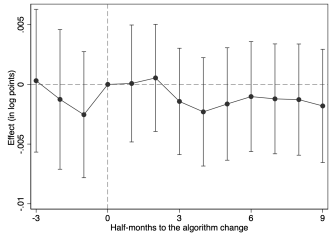
(e) Update Frequency



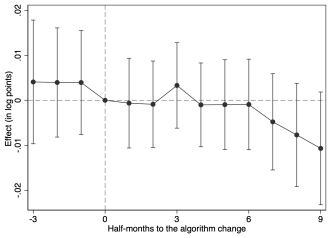
(f) Price



(g) Average Rating



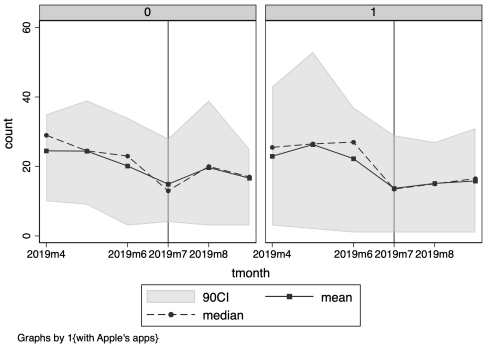
(h) File Size



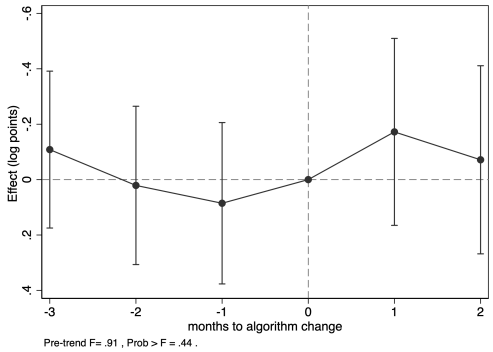
Notes. The charts present point estimates for each half-month using the difference-in-differences specification as specified in Section 3.2. The omitted period is the half-month prior to launch of the search algorithm change. Error bars indicate 95% confidence interval using standard errors robust to heteroscedasticity. Panel (b) include controls for lag downloads.

Figure E.6: Entry Around the Search Algorithm Change

Panel A. Number of New Apps



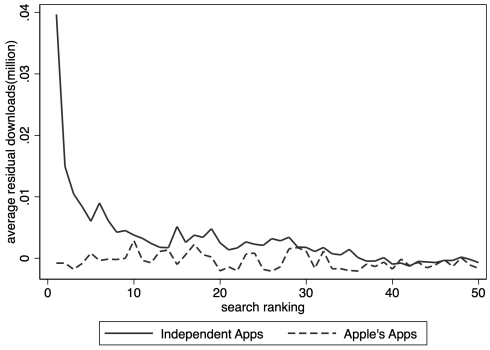
Panel B. DiD Estimates



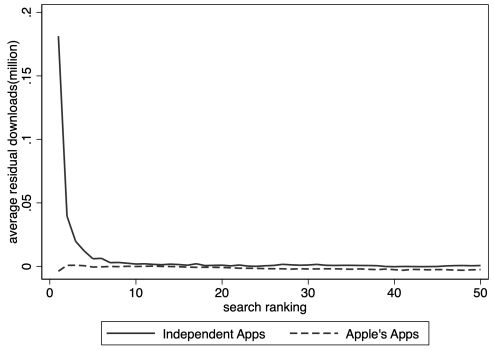
Notes. The sample to draw the figures include independent apps that were ever ranked top50 in category-specific top grossing charts during April - September 2019. Panel A shows the number of new apps in categories with Apple's apps and categories without Apple's apps. To generate Panel B, I regress the logarithm of category-month specific number of new apps as an outcome variable on the interaction terms of monthly indicator and whether the category contains Apple's apps (taking July 2019 as the reference point), as well as category-fixed effects and month-fixed effects. Panel B reports the coefficients on the interaction terms for each month. The results indicate that entry of competing independent apps did not significantly change due to the search algorithm change.

Figure E.7: Independent Apps *v.s.* Apple's Apps: Residual Downloads at Each Search Ranking, Before and After the Search Algorithm Change

Panel A. Before

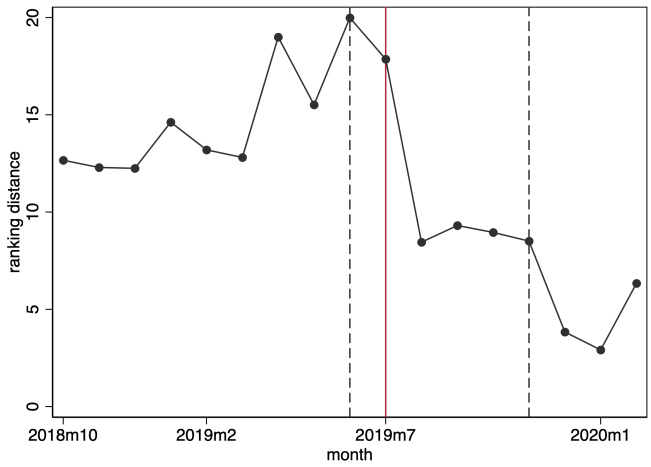


Panel B. After



Notes. The figure compares the average residual downloads of independent apps and Apple's apps on the same position in the search results for the same keyword across different days.The residual downloads are residuals from price, installation payment type,category-fixed effects and daily fixed effects. Panel A presents the comparison result before the search algorithm change(July 2019). Panel B presents the comparison result after the search algorithm change(July 2019). While Apple's apps always need lower residual downloads to achieve the same position than independent apps, the gap is smaller after the search algorithm change.

Figure E.8: Observed Rankings Relative to Rankings by Residual Downloads of Apple's Apps



Notes. The figure presents the gap between average observed within-market search rankings of Apple's apps to average within-market rankings of residual downloads of Apple's apps in each month. The residual downloads are residuals from price, installation payment type, category-fixed effects, and daily fixed effects. It shows that an average Apple's apps would be ranked lower according to residual downloads in each of the month. More importantly, it shows that the gap were flat before April 2019, and reached peak during April and July 2019, then significantly dropped after July 2019. Such pattern is consistent with identified self-preferencing across months.

Figure E.9: Estimated Bounds of the Fixed Costs of Updates (million $)