Teng, Xuan

Working Paper

Self-Preferencing, Quality Provision, and Welfare in Mobile Application Markets

CESifo Working Paper, No. 10042

Provided in Cooperation with:
Ifo Institute – Leibniz Institute for Economic Research at the University of Munich

Suggested Citation: Teng, Xuan (2022) : Self-Preferencing, Quality Provision, and Welfare in Mobile Application Markets, CESifo Working Paper, No. 10042, Center for Economic Studies and ifo Institute (CESifo), Munich

This Version is available at:
https://hdl.handle.net/10419/267275

CESifo WORKING PAPERS

10042
2022

October 2022

Self-Preferencing, Quality Provision, and Welfare in Mobile Application Markets

Xuan Teng

CESifo

Impressum:

CESifo Working Papers
ISSN 2364-1428 (electronic version)
Publisher and distributor: Munich Society for the Promotion of Economic Research - CESifo GmbH
The international platform of Ludwigs-Maximilians University's Center for Economic Studies and the ifo Institute
Poschingerstr. 5, 81679 Munich, Germany
Telephone +49 (0)89 2180-2740, Telefax +49 (0)89 2180-17845, email office@cesifo.de
Editor: Clemens Fuest
https://www.cesifo.org/en/wp
An electronic version of the paper may be downloaded
· from the SSRN website: www.SSRN.com
· from the RePEc website: www.RePEc.org
· from the CESifo website: https://www.cesifo.org/en/wp

CESifo *Working Paper No. 10042*

Self-Preferencing, Quality Provision, and Welfare in Mobile Application Markets

Abstract

Platforms may give preferential treatment to their own products in search results. Whether and how to regulate this self-preferencing behavior is an intensely debated antitrust issue. This paper identifies self-preferencing and quantifies its equilibrium welfare effects in Apple App Store. I start by examining the effect of a change in the platform's search algorithm that dropped several Apple's apps from top positions. I find that the search algorithm change leads to significantly higher installations and update frequencies of independent apps that compete with Apple's apps in the same categories. Then I develop an empirical model of consumer search and update competition allowing for potential self-preferencing. The model is estimated with aggregate data on consumer search and purchase, search ranking, and app characteristics. Estimation results point to self-preferencing: Apple's apps are more likely to be ranked higher than independent apps conditional on app quality, price, ratings, and title match with search terms. Based on counterfactual simulations, I find that eliminating the identified self-preferencing modestly increases the quality of independent apps on average. Furthermore, the elimination improves consumer surplus by $2.2 million and profits of independent developers by $1.6 million per month.

JEL-Codes: D120, D430, D830, L130, L410, L860.

Keywords: search algorithm, consumer search, endogenous product characteristics, mobile application.

Xuan Teng
Department of Economics
University of Munich (LMU)
Akademiestr. 1/III
Germany – 80799 Munich
xuan.teng@econ.lmu.de

September 2022
This paper was previously circulated as "Preferential Search Ranking, Quality Provision, and Welfare in Mobile Application Markets". It 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 Francine Lafontaine, Chenyu Yang, Benjamin Leyden, Alessandra Allocca, and participants at Michigan IO Lunch and IO 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. All errors are mine.

1 Introduction

Search algorithm shapes the competition landscape on digital platforms, and is designed by the platform. This raises up the concerns that dual-role platforms, which present third-party products as well as selling platform-owned products, may give preferential treatment to their own products in search results. This self-preferencing behavior can lead to dominance of platform-owned products in top positions, and disadvantaged accessibility of competing independent products. While the press covers multiple stories indicating the existence of self-preferencing, platforms deny it.[1] Meanwhile, self-preferencing is an intensely debated antitrust issue.[2] For example, some third-party suppliers complain that unfair search rankings discourage them from innovation efforts and thus quality improvement.[3] To shed light on the two debates, this paper studies two research questions: i) whether there is self-preferencing in dual-role platforms; ii) what is the equilibrium welfare effect of self-preferencing.

With self-preferencing, third-party products are less likely to get ranked high than without self-preferencing. How the resulting lower accessibility affects the third-party suppliers' trade-off between exploitation and expansion determines whether self-preferencing is anti-competitive or pro-competitive. On the one hand, lower accessibility decreases demand elasticity, and thus encourages exploitation of remaining consumers by lower quality or higher price, which is anti-competitive. On the other hand, lower accessibility can be compensated by higher quality or lower price from the perspective of consumers, encouraging expansion to new consumers with quality improvement or price reduction, which is pro-competitive. Thus, given self-preferencing, its supply-side effect and thus equilibrium welfare effect are theoretically ambiguous.

To empirically address the research questions, I compile a new dataset on Apple App Store, develop and estimate an empirical model of consumer search and update competition with potential self-preferencing. The model is motivated from reduced-form effects of an un-anticipated search algorithm change on consumer search and purchase, as well as independent apps. Notice that the key challenge for addressing the first question is to disentangle self-preferencing from consumer preference: platform-owned products might deserve higher search rankings if platform users prefer platform-owned products and platforms simply present what consumers like most in the most accessible positions. To that end, the algorithm change also helps with separate identification of search cost and consumer preference by introducing exogenous

[1] Among others, example stories include (1) Dougherty, Conor. 2017. "Inside Yelp's Six-Year Grudge Against Google." The New York Times, July 1.https://www.nytimes.com/2017/07/01/technology/yelp-google-european-union-antitrust.html, (2) Mattioli, Dana. 2019. "Amazon Changed Search Algorithm in Ways That Boost Its Own Products" The Wall Street Journal, Sept. 16. https://www.wsj.com/articles/amazon-changed-search-algorithm-in-ways-that-boost-its-own-products-11568645345, (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. https://www.nytimes.com/interactive/2019/09/09/technology/apple-app-store-competition.html. As an example of how platforms deny the usage of self-preferencing, article (3) wrote "They (Apple's executives) said, Apple apps generally rank higher than competitors because of their popularity and because their generic names are often a close match to broad search terms".

[2] Example antitrust discussions include i) one clause (section-2-(a)-(1)) in a proposed bill in the U.S. House providing that certain discriminatory conduct by covered platforms shall be unlawful, see 117th Congress, 2021, "H.R.3816 - American Choice and Innovation Online Act", congress.gov, June 11. https://www.congress.gov/bill/117th-congress/house-bill/3816/text; ii) Google's antitrust cases, see Molla, Rani and Estes, Adam Clark. 2020. "Google's three antitrust cases, briefly explained". Vox, Dec 17. https://www.vox.com/recode/2020/12/16/22179085/google-antitrust-monopoly-state-lawsuit-ad-tech-search-facebook

[3] A quote from an app developer on Apple App Store, reported by Wall Street Journals in 2019: "We want to invest more on that aspect (innovation), however, instead of hiring another two AI Ph.D.s, we have to use that money to just get ranked higher." See Mickle, Tripp. 2019. "Apple Dominates App Store Search Results, Thwarting Competitors", The Wall Street Journal, July 23. https://www.wsj.com/articles/apple-dominates-app-store-search-results-thwarting-competitors-11563897221

variation in search ranking. Given the identified consumer preference, using a standard decomposition approach, I look for evidence of larger likelihood to be ranked higher with Apple ownership, in order to detect self-preferencing. Given the evidence, I conduct counterfactual simulations to quantify the effect of self-preferencing on app quality and welfare.

Apple App Store provides an ideal context to study self-preferencing for three reasons. First, Apple sells its own apps on the platform in various categories.[4] Second, search is an important channel to access apps on the platform. While consumers may discover an app by navigating the editorial contents, 65% of all downloads on the platform happen after search during 2020.[5] Third, data on historical non-personalized search ranking is available for this platform. While some platforms' search results are personalized, I did not find solid evidence or report on personalized search results in Apple App Store.[6] This lack of personalization alleviates mis-specification concerns in demand estimation when individual-level data is unavailable.

For this project, I compile a new dataset on Apple App Store from multiple data sources. The main data comes from AppTweak, a third-party App-Store-Optimization (ASO) tool company. It covers product-month level information on downloads, revenue, search ranking, and app characteristics in various categories between April 2018 and February 2020.[7] To capture consumers' search behavior, I augment the main dataset with category-month level data on consumers' conversion from search to installation for free apps and paid apps respectively, and keyword-daily level data on how frequently a keyword is searched by consumers.

Dominance of platform-owned products may indicate self-preferencing. Thus, I start with a descriptive analysis where I exploit an unanticipated search algorithm change on the Apple App Store to study the effects of reduced dominance of platform-owned products on competing independent apps. Specifically, The algorithm change dropped some Apple's apps from top positions. It happened in July 2019 but was very firstly reported in two months later by *New York Times*.[8] Using a difference-in-differences (DiD) approach, I compare independent apps competing with Apple in the same categories and independent apps in categories without Apple's apps, before and after July 2019. I find that the search algorithm change boosted up affected independent apps by 3.6% in search results, leading to a 22% increase in their installations. However, it did not significantly affect their conversion rates. The result indicates that search ranking does not affect consumer choice conditional on search, which motivates me to assume that search ranking does not affect consumer preference and only affect search costs.[9] This is a useful assumption that enables me to

[4]Examples include Apple Music and GarageBand for music apps, Files and Voice Memos for utilities apps, Apple TV and iTunes Remote for entertainment apps, among others. For a complete and up-to-date list of Apple's apps, please see https://apps.apple.com/us/developer/apple/id284417353?mt=12.

[5]Editorial contents may be "Featured App", or top charts, among others. The figure comes from Apple, 2021. "Be discovered." https://searchads.apple.com/

[6]One reason for non-personalized search results maybe Apple's claim on strong privacy protection. Meanwhile, editorial contents are personalized, leaving less benefit from personalized search results.

[7]Specifically, the data covers all non-game app categories and 16 out of 18 game app categories. Table E.12 lists the covered app categories. Because actual downloads and revenues are highly confidential in the industry, literature on the mobile application industry commonly relies on estimated downloads and revenues, so does this paper. Section 2 shows that the estimated downloads and revenues fit the actual data well.

[8]Nicas, Jack and Collins, Keith. 2019. "How Apple's Apps Topped Rivals in the App Store It Controls". The New York Times, Sept. 9. https://www.nytimes.com/interactive/2019/09/09/technology/apple-app-store-competition.html

[9]Similarly, Ursu (2018) exploits experimental data on an online hotel platform and finds that rankings affect what consumers search, but conditional on search, do not affect purchases.

separately identify search costs and consumer preference in the demand model.[10] On the supply side, I find that the search algorithm change increased update frequency of independent apps by 2.1%. [11] However, I find no significant effects on price, average ratings, and file size. The result motivates me to focus on app developers' upgrading decisions.

On the demand side, I present a discrete-choice model in which consumers search to learn idiosyncratic match values and purchase the most-preferred searched app in the way of optimal sequential search. This model returns an app quality index containing utility contribution of observable non-price attributes and unobservable shifters. To that end, I take two steps. First, to estimate the model with aggregate data, I borrow a parametric assumption on the distribution of consumer-product specific search costs from Moraga-González, Sándor and Wildenbeest (2022). The parametric assumption rationalizes a closed-form choice probability in a Berry, Levinsohn and Pakes (1995) framework. Second, to identify search cost parameters on search rankings, I leverage instruments constructed from the search algorithm change and apps' title match with search terms. With additional instruments based on cross-category variation in developers' product portfolios, demand estimation shows that iPhone users significantly prefer Apple's apps over independent apps, indicating an explanation for the dominance of Apple's apps in top positions and the potential non-existence of self-preferencing. It also shows that consumers significantly prefer apps with more and better updates on average, indicating that app update is a positive quality shifter.

Next, I fit the search ranking data with a rank-ordered logistic regression model to detect self-preferencing. The model captures the effects of well-accepted influential factors in flexible ways, by allowing monthly category-specific effects of price, title match, and lagged app ratings. Then, conditional on app quality and the influential factors, I capture dynamic self-preferencing with monthly effects of Apple ownership. Estimation shows that Apple's apps have significantly higher ranking scores than independent apps between April and August in 2019, but not in the early and late months during the sample period. This result points to self-preferencing on the Apple App Store between April and August 2019. In other words, although consumers have a preference for Apple apps as estimated from the demand model, the preference is not large enough to justify the higher rankings of Apple apps. Furthermore, consistent with the search algorithm change in July 2019, I also find that the self-preferencing significantly decreases after July 2019.

On the supply side, I develop and estimate an update competition model to recover information about update costs so as to examine how self-preferencing affects update and thus quality in equilibrium. In the model, when making upgrading decisions, developers face uncertainty about search rankings and form beliefs on possible search rankings based on the identified search ranking model. Therefore, apart from the direct effect of updates on installations, developers also consider the indirect effect of updates on installation through affecting search ranking probabilities. The indirect incentive is typically missing from existing

[10] In principle, separate identification of search cost and indirect utility is still feasible even if I allow search ranking to have direct effect on indirect utility, because conversion rates and downloads are affected by search costs and indirect utility in different ways, and I observe both. However, as mentioned, conversion rates are observed at coarser level than downloads. Thus, I go with the simplification assumption.

[11] To capture the idea that not every update is as valuable as each other, I weight update frequency by the length of release notes. I assume that an update (release of a new version) is more likely to be valuable with longer release notes. Leyden (2019) classifies updates into bug-fix updates and feature updates based on natural language processing and machine learning techniques, where he also exploits release note information.

supply models, but it is essential for studying platform design.

It is challenging to capture the indirect incentives of update because the number of possible orderings of products is the factorial of the number of products in the market. For example, an average market has 65 apps in the data, implying $65!(\approx 8.2 \times 10^{90})$ possible orderings of products. Thus, allowing developers to consider all possible search rankings is infeasible for computation. To achieve tractability, I use a heuristic algorithm to truncate the set of possible search rankings and use the estimated search ranking model to calculate developers' beliefs on the truncated set. The behavioral assumption for the truncation method is that developers only consider some most likely search rankings when making update decisions. Results from within-sample tests show that the algorithm performs well at least in markets with no more than 10 products.[12]

Supply-side estimation shows that marginal update costs increases in update frequency. I also obtain bounds on fixed costs for upgrading an app to rationalize the no-update observations following the approach taken by Fan and Yang (2020*a*). Specifically, I assume that an app developer's observed set of updated apps is profit-maximizing in a Bayesian Nash equilibrium. Therefore, upgrading or not upgrading an app should not increase the developer's expected profit. Based on these conditions, I obtain market-specific upper bounds for apps that are updated and lower bounds for apps that are not updated. Estimation results show that the bounds are positively correlated with app qualities.

Based on the estimated consumer preference, search cost, ranking probability, and update costs, I conduct counterfactual simulations to quantify the equilibrium welfare effects of self-preferencing. Specifically, I eliminate the identified self-preferencing in search during June and July 2019, the two months with the most substantial estimated self-preferencing. I find that removing the self-preferencing improves consumer surplus by $2.2 million (0.2 percent) and the expected profit of independent developers by $1.8 million (0.7 percent) per month in equilibrium.

Zooming into app quality adjustment, the simulation results imply positive but modest average effect of eliminating the identified self-preferencing on app quality. Specifically, I find that an average market sees a 0.4% increase in average update frequencies and thus a 0.01% increase in average app qualities after eliminating the identified self-preferencing. One reason for the small average effects is heterogeneity in the direction of changes. In particular, I find that the expected update frequency of an independent app might decrease up to 20% and might as well increase up to 25% after the elimination, which results in a small average product-level effect of 0.3%. Similar heterogeneity shows up in the market-level effects. The heterogeneity results are consistent with the previously mentioned theoretically ambiguous effect of self-preferencing on quality improvement. Given the positive and modest supply-side effects, while the improved quality contributes to the welfare gains, most of the gains come from more efficient match between consumers and apps.

Overall, these counterfactual simulation results suggest that self-preferencing hurts both consumers and third-party suppliers, and thus support regulation policies against self-preferencing. The surprisingly small welfare effects are due to estimated consumer preference: iPhone users prefer Apple's apps, which limits

[12] In tests with all markets with no more than 10 products, the truncated set of possible search rankings can capture 60% to 100% of the top10 most-likely ordering of products, with an average coverage rate of 85%.

the extent of identified self-preferencing, and thus its welfare effects. It indicates that the benefit from regulating self-preferencing might be confined by confounding consumer preference, calling for highly efficient technology to implement regulation policies.

1.1 Related Literature

This paper contributes to four strands of literature. First and foremost, this paper is related to the broad literature on information frictions and product competition, dated back to Stigler (1961) and Diamond (1971).[13] Information frictions have been theoretically and empirically shown to be able to shape the competition landscape in a large variety of markets.[14] A common focus of this literature is universal changes of information frictions for some or all products without affecting information frictions for others.[15] In contrast, this paper, by studying self-preferencing, focuses on differential changes of information frictions that reduce the search costs for some products but increase the search costs for others. Such differential changes of information frictions are commonly seen in papers studying platform design. Examples include Yao and Mela (2011), Arnosti, Johari and Kanoria (2014), Ghose, Ipeirotis and Li (2014), Nosko and Tadelis (2015), Fradkin (2017), Santos, Hortaçsu and Wildenbeest (2017), Dinerstein et al. (2018), Huang (2018), Choi and Mela (2019), Lam (2021), and Lee and Musolff (2021). In terms of topics, this paper is closely related to Dinerstein et al. (2018). However, there are three differences between Dinerstein et al. (2018) and this paper: design of search result display *v.s.* self-preferencing, homogeneous products *v.s.* differentiated products, pricing response *v.s.* product quality response. Despite this growing literature, to my knowledge, there is no evidence on the equilibrium welfare effects of platform design with endogenous non-price product attributes, and thus this paper complements the literature.

By studying the upgrading decisions of app developers, this paper is also related to the literature on endogenous product choices.[16] The literature has studied endogenous product choices in a variety of industries, including retail video (Seim, 2006), retail eyeglasses (Watson, 2009), ice-cream (Draganska, Mazzeo and Seim, 2009), TV (Chu, 2010; Crawford and Yurukoglu, 2012; Crawford, Shcherbakov and Shum, 2019), CPU (Nosko, 2010), newspapers (Fan, 2013), home PC (Eizenberg, 2014), movie (Orhun, Venkataraman and Chintagunta, 2016), ratio (Berry, Eizenberg and Waldfogel, 2016), smartphones (Wang, 2017; Fan and Yang, 2020*a*), trucks (Wollmann, 2018), vendor allowances contracts (Hristakeva, 2019), and retail craft beer (Fan and Yang, 2020*b*). However, none of them has examined endogenous product choices in the mobile application industry; neither is there much evidence on the effects of platform design on endogenous product choices. Thus, this paper complements the literature.

[13] For related review, please see Goldfarb and Tucker (2019), Anderson and Renault (2018) and Ratchford (2009).

[14] Theoretical work and empirical work have studied information frictions and competition in markets for health care, airlines, mutual funds, personal computers, automobiles, trade waste, books, and consumer electronics. Some examples include Dranove et al. (2003), Brown (2017), Brown (2019), Orlov (2015),Hortaçsu and Syverson (2004), Goeree (2008), Tadelis and Zettelmeyer (2015), Salz (2020), Bar-Isaac, Caruana and Cuñat (2012), Ellison and Ellison (2018), and Baye, Morgan and Scholten (2004).

[15] For example, Fishman and Levy (2015) theoretically examines the effects of search costs on investment in qualities. Brown (2017) examines how price transparency affects equilibrium prices and welfare in the medical imaging services market and finds a 22 percent reduction in prices if all patients had full information. Allen, Clark and Houde (2019) quantifies the role of search costs and brand loyalty for market power and finds that search frictions reduce consumer surplus by $12/month/consumer in mortgage markets.

[16] For related review, please see Crawford (2012).

Finally, this paper is also related to the emerging literature on mobile applications. Examples include Ghose and Han (2014), Bresnahan, Orsini and Yin (2014), Mendelson and Moon (2016), Mendelson and Moon (2018), Huang (2018), Leyden (2019), Ershov (2020), Li, Bresnahan and Yin (2016), Wang, Li and Singh (2018), Leyden (2021), Allon et al. (2021), Singh, Hosanagar and Nevo (2021) and Janssen et al. (2021). In terms of methodology, this paper is most closely related to Leyden (2019) which also develops an empirical model to rationalize app developers' upgrading decisions for a different research question. However, whereas Leyden (2019) assumes that consumers have perfect information about apps, my model allows consumers to have imperfect information about match values with apps. While imperfect information is not necessary for Leyden (2019) to study the effects of digitization, it is a necessary assumption to study the effect of self-preferencing in this paper. In terms of topics, this paper is most closely related to Ershov (2020) which studies the effect of reduced consumer discovery costs on product entry and consumer welfare after re-categorization of game apps in the Google Play Store. However, whereas Ershov (2020) focuses on congestion effects and does not explicitly model consumer search, this paper builds on an explicit model of consumer search to separate search costs from utilities and thus quantifies welfare effects. Therefore, I complement these papers by studying the equilibrium welfare effects of self-preferencing with explicit consumer search model and endogenous upgrade decisions.

1.2 Roadmap

The remainder 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 the descriptive evidences. Section 4 describes the empirical model of consumer search and download, 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.

2 Data

My dataset is a product-category-month panel on U.S. Apple App Store from April 2018 to February 2020. For an app in a category in a month, I obtain the estimated number of downloads, estimated revenue, developer, app title and subtitle, installation price, ratings, released versions, as well as other app features that are observable to consumers on the product page of the app. I use estimated downloads and revenues because data on actual downloads and revenues of apps are highly confidential and typically unavailable. For exposition, I use "downloads" and "revenues" in referring to the estimated downloads and revenue.[17] Furthermore, for an app/keyword combination on a day, I obtain the position of the app in the search results for the keyword. Lastly, to capture consumers search behavior, I obtain keyword-day level search volume and

[17]Each download is an installation by a unique new consumer. Thus, repeated installation by the same consumer will not increase downloads. Revenues include installation revenue, in-app-purchase and in-app-subscription revenue before Apple tax. The estimates are from AppTweak based on a machine learning algorithm by linking top charts rankings with actual downloads and revenues they observe from connected app developers. AppTweak reports that 42% of the top50 grossing apps are their active customers. Figure F.1 shows plausible fitness of the estimated downloads.

category-month-type level conversion rates, where type indicates whether the apps are free or paid apps.[18] The data comes from AppTweak and Sensor Tower, two app store optimization service providers.[19]

Following the literature, to capture the most important effect, I focus on popular apps and popular keywords.[20] The popular apps are selected based on top charts ranking and annual downloads in 2019. For example, within each category, the popular apps contain the 50 mostly downloaded apps in 2019 among apps that ever showed up in top50 positions in top-grossing charts for the category. For each category, I construct a set of popular keywords based on the number of popular apps that show up in the search results for the keyword. Figure F.2 plots residual downloads against search ranking in these popular keywords, showing a negative correlation between the two, indicating that the construction is plausible. Appendix A.1 explains the sample selection process in greater detail. In the end, the sample covers 38 categories, 3,110 apps, and 2,265 keyword/category pairs, constituting 56,570 observations at app-category-month level.[21]

Table 1 presents summary statistics of the app/category/month panel data. On average, an app has 60,000 unique new consumers downloading it in a month and receives $0.37 million from installation, in-app purchase and subscription. Most apps are free: 58 percent of observations come from free apps. Conditional on paid apps, the average installation price is $4.15. Apple only provides a small portion of apps on the platform: one percent of observations come from Apple's non-preinstalled apps. Because installation of pre-installed apps is not within the choice of consumers, features of pre-installed apps are aggregated to category-month level, and their summary statistics are provided in Appendix Table E.13. This appendix table also provides summary statistics of other variables that are not observed at app/category/month level.[22]

Row (6) in Table 1 reports the summary statistics for update frequency, a variable to capture app developers' efforts to improve quality. Specifically, I construct update frequency as the number of released versions in a month, weighted by release note length. The weight is to give major updates (e.g. versions with new features) higher weights than minor updates (e.g. bug-fixing versions). Row (6) shows that given such a construction, on average, an app in a category updates 0.68 times in a month. Appendix A.2 gives more details on the construction of update frequency as well as other variables in the table.

The data presents wide variation in app performance and characteristics. For example, the second column of Table 1 shows that the standard deviation of downloads is 3.8 times its mean, and the standard deviation of revenues is 4.9 times its mean. Importantly, among paid apps, the standard deviation of installation price is 1.1 times its mean, providing essential variation to identify price elasticity in the mobile application markets. Similarly useful variation shows up in other app characteristics as well: across the app characteristics shown in rows (6) to (12), their standard deviations are about 13 percent to 1.8 times their corresponding means.

[18] Search volume is an integer between 5 and 100 constructed by Apple to index the number of consumers searching for a keyword on the app store. The conversion rate in my dataset captures the conversion from impression to installation. It is the ratio of consumers who download an app among consumers who see the app on the app store. Note that seeing the app in search results without clicking on the app counts as impression. These category-month-type average conversion rates are actual and not estimated, and they are calculated based on independent apps.

[19] Additionally, for market size, I obtain the monthly number of U.S. iPhone users during the sample period from Comscore.

[20] For example, Kim, Albuquerque and Bronnenberg (2017) uses data covering 200 best-selling camcorders on Amazon.com.

[21] Table tab:appendix.ctglist lists the 22 non-game categories and 16 game categories in the sample.

[22] For example, Table E.13 presents that on average, a category has an average conversion rate of 5.2% across apps in a month, while a within-sample keyword has a search volume of 45 out of 100 on a day.

Table 1: Summary Statistics

		Overall				Over-time Variation	
	Variable	Mean	SD	Min	Max	Avg SD[c]	Avg range[d]
(1)	Downloads (million)	0.06	0.23	10^{-6}	7.00	0.03	0.09
(2)	Revenues (million $)	0.37	1.81	0	56.51	0.07	0.22
(3)	Paid Installation	0.42	0.49	0	1	0	0
(4)	Paid Price	4.15	4.69	0[a]	99.99	0.32	0.82
(5)	Apple	0.01	0.07	0	1	0	0
(6)	Update Frequency	0.68	1.00	0	11	0.48	1.60
(7)	Average Rating	4.40	0.55	1	5	0.08	0.23
(8)	Age (month)	51.16	32.66	1	140	4.65	14.51
(9)	File Size	225.70	411.00	0.73	4096	17.84	49.41
(10)	#Screenshots	5.54	1.96	0	10	0.56	1.56
(11)	Description Length	2.21	1.04	0	4.00	0.15	0.40
(12)	Offer In-App-Purchase	0.74	0.44	0	1	0	0
(13)	Top50 in Search Results	0.62	0.49	0	1	0.09	0.22
(14)	Search Ranking|Top 50	23.57	11.58	1	50	4.83	14.11
(15)	Apple's apps[b]	16.14	12.01	1	50		
(16)	Independent apps	23.59	11.59	1	50		
(17)	Title Match (x10)	0.24	0.29	0	1.64	0.02	0.06
(18)	Subtitle Match (x10)	0.22	0.28	0	1.74	0.04	0.10
	Number of app/category/months		56,570				
	Number of apps					3,110	

[a] 9.87% paid apps have ever reduced their prices to 0 in the sample.
[b] Include both pre-installed and non-pre-installed Apple's apps, which constitutes 644 app/category/month observations. Pre-installed apps are not included when calculating the summary statistics for other row variables. Search rankings of pre-installed apps are from Sensor Tower. The data for all the other variables come from public information and/or AppTweak.
[c] Average of X_j, where $X_j = (\text{ standard deviation of } x_{jt} \text{ across months indexed by } t)$. For search ranking related variables, x_{jt} is average x_{jgt} across categories indexed by g.
[d] Average of X_j, where $X_j = (\text{ range of } x_{jt} \text{ across months indexed by } t)$. For search ranking related variables, x_{jt} is average x_{jgt} across categories indexed by g.

Rows (13) to (16) in Table 1 report search-related variables. Many apps in the sample have shown up in top50 search results. Specifically, given an app/category/month combination, the likelihood for the app to appear in top50 search results of at least one popular keyword of the category on at least one day in the month is 0.62.[23] Conditional on such appearance, I calculate the average search ranking the app/category/month combination across keywords and days, using search volume as weights for keywords.[24] This conditional weighted average search ranking is reported in row (14), with an average of 23.57, indicating that on average, an app is ranked near the 24-th position across keywords if it shows up in the top50 search results in a

[23] I focus on top50 search results to capture the most relevant search ranking. Specifically, in the data, total downloads from top50 apps in search account for 60% total downloads from top500 apps in search.

[24] I aggregate search ranking from app-keyword-day level to app-category-month level for two reasons: i) I define a market as a pair of category/month, ii) I would like to keep the model tractable by avoiding studying keyword-specific ranking effect. An alternative market definition is the set of apps appearing in the search results given a pair of keyword/period, but historical records on such information are unavailable in mobile application industry.

category in a month. For exposition, I use "search ranking" to refer to this conditional weighted average search ranking. Rows (14) and (15) show that on average, the search ranking of an Apple's app is 7.5 positions higher than an independent app. In Section 5, I examine to which extent this difference is due to consumer preference *v.s.* self-preferencing.

Rows (17) and (18) report summary statistics of a source of exogenous variation in search ranking: how much is an app's title and subtitle matched to the corresponding keyword. By assuming that consumers do not directly receive utility from titles and subtitles, I argue that title and subtitle only affect downloads through affecting search ranking. On average, an app is matched with 0.24 popular keywords in title and 0.22 popular keywords in subtitle in a category in a month. In Section 3, I will illustrate the other source of exogenous variation: an un-anticipated search algorithm on the app store.

The right panel of Table 1 explores over-time variation for each row variable, in order to distinguish short-term strategic choices of app developers from long-term ones. The first column of the right panel shows that the average within-app over-time standard deviation of update frequency is 48 percent of its overall standard deviation, the same ratio for file size is just 4 percent, indicating rigidity of file size for a given app.[25] We see similar patterns when comparing comparing the average within-app over-time range to the overall range in the second column. For example, an average app's file size may change up to 49.4MB, accounting for 1% of the overall range of file sizes; while an average app's update frequency may change up to 1.6 updates, accounting for 15% of the overall range of update frequency. We see similar stickiness price and average rating. Overall, the results indicate greater flexibility of update frequency over time for a given app, compared to other app characteristics.

Table 2: Summary Statistics on App Characteristic Dispersion based on Multiple-Category Developers

Variable	Average cross-category SD within Developer/Month[a]	Average SD of developer's cross-category SD within Category/Month[b]
Update Frequency	0.73	0.52
Price ($)	0.34	0.75
Average Rating	0.09	0.13
Ever Top50	0.15	0.17

[a] Average of X_{ft}, where $X_{ft} = (\text{ standard deviation of } (1/\#\mathscr{J}_{fgt})\sum_{j\in\mathscr{J}_{fgt}} x_{jgt} \text{ across categories indexed by } g)$, where $\mathscr{J}_{fgt}$ is the set of apps owned by developer f in category g/month t.
[b] Average of Y_{gt}, where $Y_{gt} = (\text{ standard deviation of } X_{f(j)t} \text{ across apps } j \in \mathscr{J}_{gt})$, 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.

Table 2 looks at app characteristic variation in more details. The first column looks at within-developer dispersion of app characteristics across categories. Note that a developer may publish apps in different categories and thus form category-specific portfolios of apps. Are these category-specific portfolios different from each other in terms of app features within a developer/month pair? The results in the first column indicates a positive answer. For example, on average, a developer/month pair has a standard deviation of

[25] By construction, update frequency may increase due to more versions released, and/or increased complexity of released versions (measured with longer release notes).

0.73 for update frequency across category-specific portfolios. Furthermore, within a market, are competing developers different from each other in terms of how diversified their portfolios are across categories?[26] The second column reports relevant summary statistics and provides a positive answer.[27] For example, on average, a market has a standard deviation of 0.52 across apps in terms of their developers' standard deviations for update frequency across category-specific portfolios.

3 Descriptive Evidence

Dominance of platform-owned products may indicate self-preferencing. This section exploits an unanticipated search algorithm change on Apple App Store to find reduced-form evidence on how dominance of platform-owned products in search results affects consumers and independent products. The findings are also useful for motivating the empirical model of the mobile application industry with potential self-preferencing.

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".[28] 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.[29] 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.[30]

Figure 1 shows that the search ranking of Apple's apps were lowered after the search algorithm change, on average.[31] The vertical axis is the average search ranking across a given group of apps: (1) Apple's pre-installed and non-preinstalled apps (solid line); or (2) other multiple-app developers' apps (dashed line); or (3) single-app developers' apps (dotted line). The horizontal axis is the month. As mentioned earlier, the new search algorithm tweaked a feature that groups apps by makers. Thus, all multiple-app developers

[26] In the data, an average market has 52.7 developers, 20.5 of whom are multiple-category developers.

[27] Specifically, I match each app with its developer's cross-category portfolio variation that was calculated in the first column. Then, within each market, I calculate the standard deviation of these cross-category changes of product portfolios across apps in each market. The second column reports the average of this standard deviation across markets.

[28] Nicas,Jack and Collins,Keith. 2019. "How Apple's Apps Topped Rivals in the App Store It Controls". The New York Times, Sept. 9. https://www.nytimes.com/interactive/2019/09/09/technology/apple-app-store-competition.html

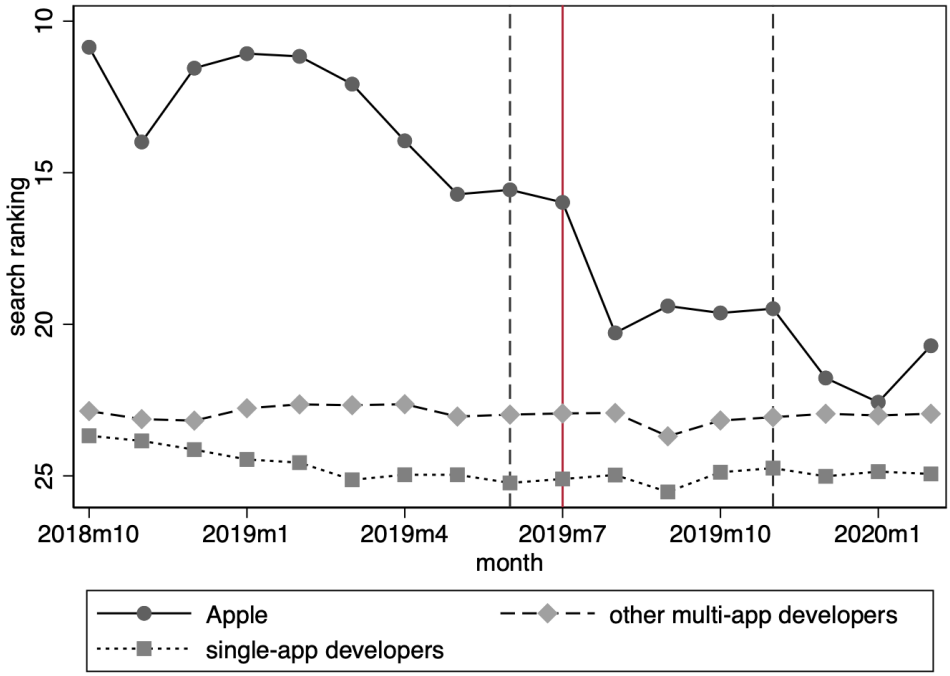
[29] Liptak, Adam and Nicas, Jack. 2019. "Supreme Court Allows Antitrust Lawsuit Against Apple to Proceed". The New York Times, May. 13. https://www.nytimes.com/2019/05/13/us/politics/supreme-court-antitrust-apple.html

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

[31] The change is even more evident among non-preinstalled Apple's apps, as shown in Figure F.3.

may be affected. However, Figure 1 shows that the change of search ranking mainly happens to Apple's apps rather than other multiple-app developers' apps, confirming that the algorithm change reduced the dominance of Apple's apps. Lastly, to focus on the effect of the search algorithm change, I examine the sample period between June and November of 2019.

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



The reduced dominance of Apple's apps exposes Apple's competitors to an exogenous shock on search ranking after the search algorithm change, relative to independent apps that do not compete with Apple. In particular, in my sample, 16 non-game categories contain Apple's apps while 22 non-game and game categories do not. In the next section, I will exploit such exogenous variation due to the launch of the search algorithm change interacted with the existence of Apple's apps in the category, in order to study the causal effect of the dominance of platform-owned apps on independent apps.[32]

3.2 Difference-in-Differences Analysis

I use a difference-in-differences approach to study the causal effects of the search algorithm change on independent apps. As explained in the last section, independent apps that compete with Apple's apps in the same category are exposed to the shock of reduced dominance of platform-owned products, relative to independent apps in other categories. Therefore, I define my treatment groups as those independent apps in

[32]Table E.15 shows the summary statistics on observations in categories with and without Apple's apps, before and after the search algorithm change. For example, a simple difference-in-differences estimate using the average download reported in Table E.15 implies that the downloads of apps in categories with Apple's apps decrease less by 0.01 million after the algorithm change relative to the downloads of apps in categories without Apple's apps. However, there is a sizable deviation within each group and period, and it is not ensured that the two groups are comparable.

categories that contain Apple's apps and define my control groups as those independent apps in categories that do not contain Apple's apps. Following the literature, to estimate the average treatment effect of the search algorithm change on independent apps, I use the following 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. The coefficient on the interaction term, β, captures the average treatment effect of the search algorithm change. The specification includes app-category fixed effects, λ_{jg}, and month-fixed effects, λ_t. The idiosyncratic error term ν_{jgt} is assumed to be orthogonal to the other variables on the right-hand side of the equation. Lastly, y_{jgt} is a general notation of outcome variables of interests that include search ranking, downloads, conversion rate, update frequency, price, average rating, and file size. Because conversion rate is only available at type-category-month level, when applying Equation (1) to study conversion effect, I interpret j as type indicating whether the conversion rate is an average conversion rate across free apps or paid apps.

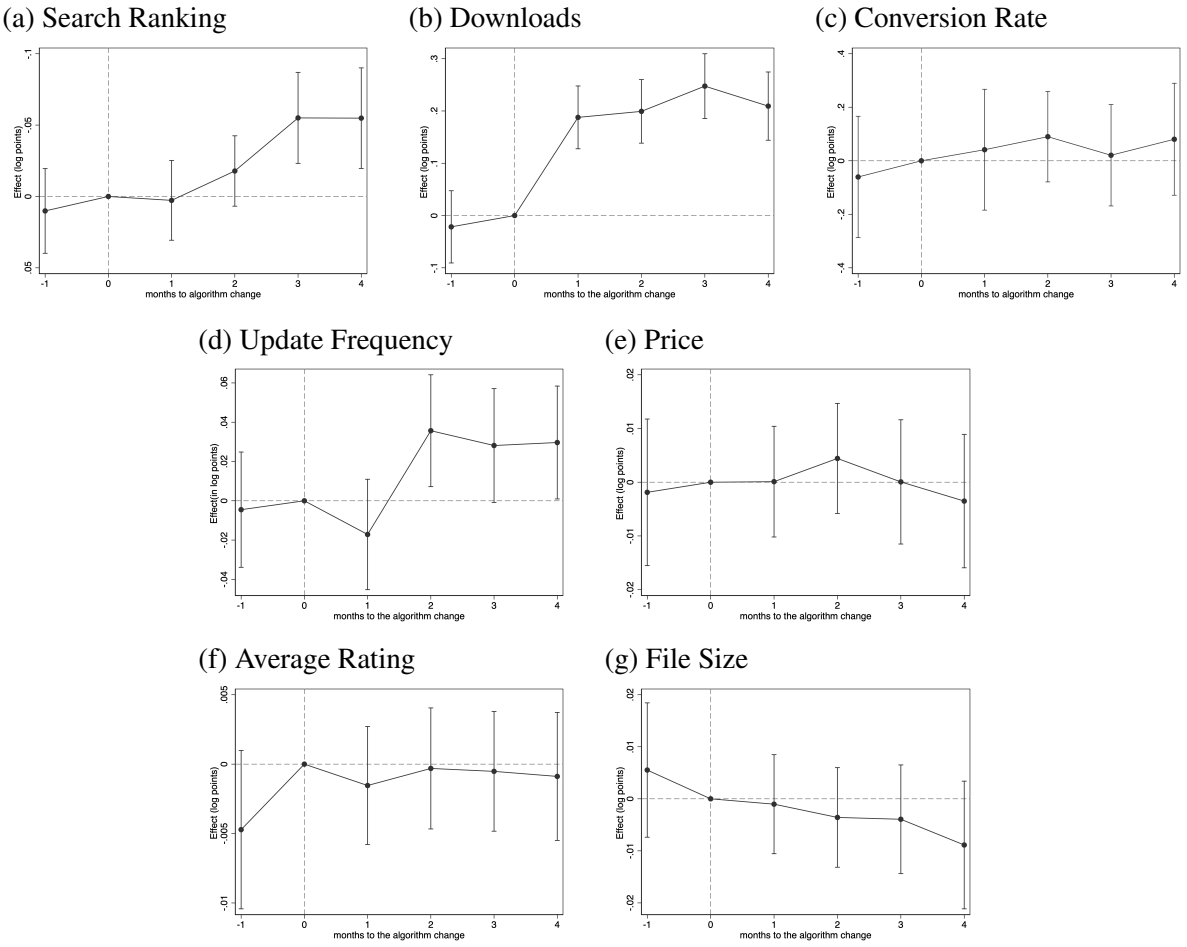
Figure 2 presents the main results by month with app-category fixed effects and month-fixed effects.[33] In the period before the search algorithm change, there is no significant effect for independent apps in categories that contain Apple's apps relative to independent apps in categories that do not 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 F.4 shows that the results are robust to multiple preperiods when examining half-month treatment effects.

Figure 2a and 2b show that right after the reduced dominance of Apple's apps in search results, the monthly downloads increased for independent apps that compete with Apple's apps relative to independent apps that do not. This download effect happens earlier than ranking effect: the search ranking of competing independent apps becomes higher starting from the second month after the algorithm change and grows over time. It indicates that the algorithm change initially leads to more rising-up of popular independent apps than dropping-down of unpopular independent apps in search results. As the new search algorithm interacts with new performance data on independent apps, all competing independent apps get boosted up on average, relative to independent apps that do not compete with Apple's apps. That said, Figure F.4b 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.

Figure 2c presents zero conversion effect of the reduced dominance: the average conversion rate across those independent apps in categories that contain Apple's apps remains parallel to the average conversion rate across those independent apps in categories that do not. Combined with the significant download effect and ranking effect, the evidence implies that the boosted-up search ranking of competing independent apps lead to as much increase in the number of consumers downloading these apps as the increase in the number of consumers seeing these apps in search results, in proportion. Furthermore, this evidence supports an

[33] 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 Reduced Dominance of Platform-owned Products on Independent Apps, by Month from Search Algorithm Change



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.

assumption on consumer search – search ranking does not affect product value, which will be applied in the empirical model. Specifically, if increased search ranking leads consumers to view an app as better than before, then conditional on seeing the product, consumers should be more likely to download the app. As this conditional probability is captured by conversion rate, the evidence shown in Figure 2c does *not* point to effect of search ranking on product value.

The last four panels of Figure 2 presents the supply-side effects of reduced dominance of platform-owned products: competing independent apps significantly update more starting from the second month after the algorithm change, relative to other independent apps; while the price, average rating and file size of competing independent apps remain unaffected.[34] This result is consistent with the over-time rigidity of the unaffected app characteristics shown in Table 1. Furthermore, the evidence motivates me to focus on update frequency as the primary developer response to self-preferencing in the empirical model. While my sample focuses on popular apps and thus is lack of entry observations, Figure F.5 shows that there was not significant effect on entry of competing independent apps due to the search algorithm change.

Table 3: Effects of Reduced Dominance of Platform-owned Products on Competitors: Difference-in-Differences Estimates

Outcome Variable	Estimated ATE	SE	Obs	Adj. R^2	FE	Mean Level
log(Search Ranking)	−0.036***	0.010	11,642	0.86	A	24.17
log(Downloads)	0.221***	0.021	20,423	0.95	A	0.055
log(Conversion Rates)	0.088	0.068	330	0.95	B	0.065
log(1+Update Frequency)	0.021**	0.009	20,423	0.62	A	0.63
log(1+Price)	0.001	0.004	20,423	0.98	A	1.91
log(Avg.Rating)	0.002	0.002	20,423	0.94	A	4.37
log(File Size)	−0.007*	0.004	20,423	0.99	A	216.30

Notes: Estimated ATE is the estimate of β in Equation 1 for the outcome variable on the row. FE: (A) app/category-fixed effects, month-fixed effects; (B) type/category-fixed effects, month-fixed effects, where type indicates paid or free apps. Conversion rates are observed at type/category/month level, the other row variables are observed at app/category/month level. Mean level is not in logarithms. SE is robust standard errors. *** p<0.01, ** p<0.05, * p<0.1.

Table 3 presents the estimated average treatment effects from Equation 1. After the search algorithm change that reduces the dominance of Apple's apps, we see significant increases in independent app's search ranking (3.6%), downloads (22.1%), and update frequency (2.1%) in categories that contain Apple's apps relative to categories that do not; while the other outcome variables of interest remain unaffected. The lower number of observations in the first row reflects that not all apps in the sample have ranked in top50 search results in a given category and month. The significantly lower number of observations in the third row is due to the more aggregate observational level of conversion rate: it is only available at type/category/month level, while the other row variables are observed at app/category/month level.

[34] Within the sample period, developers may choose to reset ratings after updates. However, while we see significant positive update effect, no significant rating effect is found. This is likely due to the fact that little rating-resets happen in my data. In particular, on a daily base, only 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. Indeed, there is a cost for developers to reset their ratings: it will cause fewer ratings and may discourage some people from downloading the app.

From this exercise, we learn that while dominance of platform-owned products in search results have impacts on independent competitors' search ranking by design, it also has real impacts on consumer choice and producer decisions. However, we should be careful to interpret the above effect as the effect of self-preferencing. Under the assumption that the reduced dominance is only due to less self-preferencing, we may interpret the average treatment effect as the effect of self-preferencing. But the algorithm change may shift other features of the search engine to achieve the reduced dominance even if there is no self-preferencing from the beginning, for example, by increasing the emphasis of ratings. Therefore, identifying self-preferencing is necessary before studying its effect. To that end, in the next section, I develop an empirical model for consumer search and competition on update frequency in the mobile application markets with potential self-preferencing.

4 Model

This section presents an empirical model of the mobile application market that explicitly incorporates consumer search and producers' competition on update frequency in the presence of potential self-preferencing. The model has three pieces. First, a demand model that returns an app quality index and a demand function. Second, a search ranking model that takes the constructed app quality index as an input to identify self-preferencing and returns a ranking probability function. Third, a supply model, where developers make update decision based on the demand function and the ranking probability function, given their heterogeneous update costs.

4.1 Demand

To study the effect of self-preferencing, incorporating the effect of search ranking on demand is necessary.[35] To that end, I use a random-coefficient discrete choice model augmented with optimal sequential consumer search to describe mobile application demand. In the model, before search, consumers are heterogeneous in the tastes, knowledge, and search costs for a given app. Therefore, different consumers may search for and thus download different apps. In the rest of this section, I first give the market definition, then specify the indirect utility and search cost of consumers.

I define a market as a category/month pair. That is, I assume a consumer may download apps from multiple categories in a month, but within a category, the consumer downloads no more than one app in a month.[36] The market share of an app j that operates in a category g in month t, is given by $s_{jgt} := Q_{jgt}/M_t$, where Q_{jgt} is the app's total downloads in month t and M_t is the number of iPhone users in month t.

[35] Moreover, the considerable number of products makes search and self-preferencing a major concern in the industry. For example, in the year 2019, there are 3.96 million apps available on Apple App Store. The figure comes from Statista. 2021. "Number of available apps in the Apple App Store from 2008 to 2021". https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/

[36] Choice among categories is beyond the scope of this paper. For examples of studies that consider choice among categories, Ghose and Han (2014) uses a random-coefficients nested logit model to incorporate consumers' choices of categories, assuming that a consumer downloads no more than one app on an app store on a day. In addition, Fershtman, Fishman and Zhou (2018) proposes a search model where consumers choose which categories to search, and firms respond to such more targeted search by strategically choosing the categories in which to list their products.

Indirect Utility. By downloading app j in category g and month t, a consumer i receives the following indirect utility:

$$\begin{aligned} u_{ijgt} &= \alpha p_{jt} + \boldsymbol{x}_{jt} \cdot \boldsymbol{\beta} + \tilde{\gamma}_i a_{jgt} + \xi_{jgt} + \varepsilon_{ijgt}, \qquad j \neq 0 \\ u_{i0gt} &= \beta_0 I_g + \varepsilon_{i0gt} \end{aligned} \tag{2}$$

where p_{jt} is the installation price of app j in month t, $\boldsymbol{x}_{jt}$ is a vector of observable (to researcher and consumer) features of app j in month t. [37] Note that $(\boldsymbol{x}_{jt} \cdot \boldsymbol{\beta})$ includes month-fixed effects, allowing month-specific average product values relative to the outside option of not downloading any app.[38] Moreover, a_{jgt} is $log(1 + \text{update frequency})$ of app j in category g and month t, ξ_{jgt} is a partially unobserved (to researchers) mean-taste shifter, I_g indicates whether category g contains pre-installed apps and thus may have a different outside-option value than categories than do not, and ε_{ijgt} is an idiosyncratic match values independently and identically drawn from Type I extreme value distribution.[39] Lastly, $\tilde{\gamma}_i$ is a mean-zero random coefficient drawn from the normal distribution $\mathcal{N}(0, \sigma)$, capturing consumers' heterogeneous tastes over updates, while the mean effect of a_{jt} on indirect utility lives in ξ_{jgt} following an AR(1) process as below.[40]

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

where η_{jgt} is unobserved (to researchers) mean-taste shifters, such as advertising, and is assumed to be orthogonal to the lagged partially unobserved quality ξ_{jgt-1}, as well as other instruments that will be specified in Section 5.1. Following Leyden (2019), the time-series process in Equation (3) intends to capture continual contribution of update to app quality and to avoid modelling update as a one-shot shock to app quality.

Now I define app j's mean-utility relative to the outside option, δ_{jgt}, and app quality, $\check{\delta}_{jgt}$, with the following equations:

$$\begin{aligned} \delta_{jgt} &:= \alpha p_{jt} + \boldsymbol{x}_{jt} \boldsymbol{\beta} + \xi_{jgt} - \beta_0 I_g \\ \check{\delta}_{jgt} &:= \delta_{jgt} - \alpha p_{jt} - \beta^{paid} paid_j \end{aligned} \tag{4}$$

where $paid_j$ indicates whether app j requires paid installation, which is included in the vector $\boldsymbol{x}_{jt}$ in Equation 2. Thus, the constructed app quality captures mean utility of an app that is irrelevant to installation payment. Moreover, note that $\boldsymbol{x}_{jt}$ includes the indicator of Apple ownership. Thus, the constructed app quality index captures consumers' preference for Apple's apps, i.e., Apple's brand effect.

Last but not least, note that Equation (2) does not include search ranking, meaning that search ranking does *not* affect product value. This is motivated by the insignificant effect of the search algorithm change on

[37] The vector of app features, $\boldsymbol{x}_{jt}$, includes average rating, an indicator of ownership of Apple, an indicator of paid installation, an indicator of offering in-app-purchase, the logarithm of age (in month), the logarithm of file size (in MB), $\log(1 + \text{length of description})$, and an indicator of game apps, and month indicators to capture month-fixed effects.

[38] Specifically, the outside option lumps all cases where consumers do not download any app in a given category in a month, which consists of 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 overtime as a consumer develops habit of using a previously downloaded app.

[39] Given the large maximum update frequency relative to mean shown in Table 1, I put the logarithm function on update frequency to avoid outliers driving estimation results. Because update frequency are normalized within categories during variable construction, update frequency may change across categories given a pair of app/month. Observations from apps that have different update frequencies in different categories in a given month account for 5.50% of all observations, 89 percent of which are game apps.

[40] The taste heterogeneity for update frequency may happen when some consumers like the availability of new features while others dislike the inconvenience of frequent updates.

conversion rates of affected independent apps, as shown in Section 3.2. This assumption helps with separate identification of consumer preference and search costs.[41]

Search Costs. To augment the discrete choice model with consumer search, I need to make assumptions on what consumers search for at what cost in which way. The short answer is that consumers only search for the idiosyncratic match value, ε_{ijgt}, at a stochastic search cost whose distribution depends on the search ranking of the app j in the category g and month t, following an optimal sequential search model in the spirit of Weitzman (1979). Next I give more details for each of the elements.

In my model, consumers are at a relatively late stage in search. Before search, they already know the mean-taste shifters: price, brand, average rating, update frequency, advertising, and other app features on the right-hand side of Equation (4) for relative mean-utility δ_{jgt}. Consumers only search to learn their idiosyncratic match values, ε_{ijgt}. Such match values could be how much the reviews of the app attract the consumer to use the app, and/or whether the consumer accepts the app's privacy practices. This information assumption on consumers implies that they do not infer product quality from search ranking, because they already know it, which is consistent with the assumption that search ranking does not affect perceived values of products by consumers. To support this information assumption, Appendix B.1 shows descriptive evidence that consumers at least know observed app features to some extent before search.

To learn about match values through search, consumers incur search costs. The search costs may include the cost of scrolling down the iPhone screen to see the app, clicking on the app, and checking its detailed information on the view page. Meanwhile, for a given app, some consumers might know it from friends before search, while others do not. Therefore, search cost depends on the search ranking in a stochastic way, which will be specified later. Now, to complete the information assumption, in the model, consumers have rational expectation in the sense that they know the distributions of the search costs and idiosyncratic match values.

During search, consumers behave as described in the optimal sequential search model (Weitzman, 1979). Specifically, consumers visit apps in the order of reservation values and stop searching when the highest realized utility so far is above the reservation value of the next product to be searched. To have a closed-form choice probability out of this search problem, I apply a distributional assumption on search costs as in Moraga-González, Sándor and Wildenbeest (2022). Specifically, I assume that the search cost of consumer i to find an app j in category g in month t , c_{ijgt}, follows the cumulative distribution function given by

$$F^{c}_{jgt}(c|\mu_{jgt}) = \frac{1-\exp(-\exp(-H_0^{-1}(c)-\mu_{jgt}))}{1-\exp(-\exp(-H_0^{-1}(c)))} \tag{5}$$

where $H_0(r) = \textit{Euler Constant} - r + \int_{\exp(-r)}^{\infty} \frac{\exp(-t)}{t} dt$; and μ_{jgt} is the app-category-month specific location parameter of the search cost distribution.[42] I call the distribution parameter μ_{jgt} as *search cost parameter*,

[41] In principal, 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.

[42] To give some intuitions of the search cost distribution function, as noted in Moraga-González, Sándor and Wildenbeest (2022), the function allows a mass of consumers to have zero search costs: $F_j(0) = \exp(-\mu_j)$. The smaller the distribution parameter μ_j,

which is a deterministic function of search ranking. In particular, because μ_{jgt} has to be positive to give a well-defined distribution function, following Moraga-González, Sándor and Wildenbeest (2022), I specify μ_{jgt} as below

$$\mu_{jgt}(\boldsymbol{\lambda}) := \log\left[1+\exp(\lambda_1 E_{jgt} + \lambda_2 E_{jgt}\log(ranking_{jgt}))\right] \tag{6}$$

where $E_{jgt} = 1\{top50\ in\ search\ results\}$, and $ranking_{jgt}$ is the search ranking of app j in category g in month g conditional on appearance in top50 search results.[43] Intuitively, search costs should be lower for apps that get into top50 search results ($E_{jgt}=1$) and have higher rankings (smaller $ranking_{jgt}$). Therefore, I expect λ_1 to be negative and λ_2 to be positive.

Following the Proposition 1 in Moraga-González, Sándor and Wildenbeest (2022), the distributional assumption on search costs in Equation (5) rationalizes a closed-form choice probability. Specifically, the probability of consumer i downloading app j in category g and month g is given by

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

where $\mathscr{J}_{gt}$ is the set of apps in category g and month t. $\boldsymbol{\theta}^D$ denotes the vector of consumer preference parameters, containing $(\alpha, \beta, \rho, \gamma, \sigma, \boldsymbol{\lambda})$. Then, I aggregate the consumer-level choice probability to market level and obtain the market share of app j in category g and month t as below:

$$s_{jgt}(\theta^D, \sigma) = \int s_{ijgt}(\boldsymbol{\theta}^D, \tilde{\gamma}_i) dF_{\tilde{\gamma}}(\tilde{\gamma}_i), \qquad \tilde{\gamma}_i \sim \mathscr{N}(0, \sigma).$$

Because the choice probability belongs to the Berry, Levinsohn and Pakes (1995) (BLP) framework, I apply the Berry inversion to back out search-augmented relative mean-utilities, $\tilde{\delta}_{jm}$, given a guess of σ. Specifically, the search-augmented relative mean-utility is relative mean-utility, δ_{jgt}, net of relative search cost parameter, which is given by

$$\tilde{\delta}_{jgt}(\boldsymbol{s}_{gt}; \sigma) := \delta_{jgt} - (\mu_{jgt}(\boldsymbol{\lambda}) - I_g \mu_{0gt}(\boldsymbol{\lambda}))$$

where $\boldsymbol{s}_{gt}$ is the vector of market shares in category m and month t, and $I_g \mu_{0gt}$ allows consumers to incur search costs to learn idiosyncratic match values with outside options in category g when the category contains are pre-installed apps.[44] Substituting Equation (4) and Equation (3) into the above equation, I obtain the main estimation equation for the demand model as below

$$\tilde{\delta}_{jgt}(\boldsymbol{s}_{gt}; \sigma) = \underbrace{\alpha p_{jt} + \boldsymbol{x}_{jt}\beta + \rho \xi_{jgt-1} + \gamma a_{jgt} - \beta_0 I_g + \eta_{jgt}}_{relative\text{-}mean\ utility} - \underbrace{(\mu_{jgt}(\boldsymbol{\lambda}) - I_g \mu_{0gt}(\boldsymbol{\lambda}))}_{relative\ search\ cost\ parameter} \tag{7}$$

the larger the mass, and thus the more likely for consumers to perfectly know product j before search.

[43] The variation of search ranking across categories given an app/month pair comes from different sets of popular keywords across categories.

[44] Such search costs may happen when consumers do not know which apps are pre-installed, or where are the pre-installed apps on the smartphone, or whether they have deleted the pre-installed apps. When category g 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}.

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 $\tilde{\delta}_{jgt}$, which returns

$$\tilde{\delta}_{jgt}(s_{gt};\sigma) - \rho\tilde{\delta}_{jgt-1}(s_{gt-1};\sigma) = \alpha\dot{p}_{jt} + \dot{x}_{jt}\beta - \beta_0(I_g - \rho I_g) - \dot{\mu}_{jgt}(\boldsymbol{\lambda}) + I_g\dot{\mu_0}_{gt}(\boldsymbol{\lambda}) + \gamma a_{jt} + \eta_{jgt}$$

where $\dot{y}_t = y_t - \rho y_{t-1}$ for any variable in the above equation. This equation can be estimated with linear regression, given a guess of $(\sigma, \rho, \boldsymbol{\lambda})$. The estimation of the demand model is based on the General-Methods-of-Moments (GMM) with iterated guesses of $(\sigma, \rho, \boldsymbol{\lambda})$. The moments are constructed based on Equation (7) and other instruments that will be specified in Section 5.1.

4.2 Search Ranking

I interpret self-preferencing as two products that are identical except for platform ownership but the platform-owned product shows up in top search results more frequently than the other. To isolate such effect of platform ownership on search ranking, I develop a flexible search ranking model that explicitly controls for partially unobserved app quality. Note that app quality is revealed by consumer demand as defined in Equation (4).

Given that Apple's search ranking algorithm is proprietary, I use a rank-ordered logistic regression model (Beggs, Cardell and Hausman, 1981) to approximate the algorithm. The advantage of the rank-ordered logistic framework is that it allows intuitive correlations between rankings of products in the same market. To clarify, a typical regression equation with search ranking on the left-hand side and an additively separable error term on the right assumes that the error terms are independent across observations. In contrast, the rank-ordered logistic model assumes independent error terms for a latent variable, namely the ranking score, and products in the same market are ranked according to the stochastic scores. Then, the model preserves intuitive correlations between rankings of products in the same market. For example, an increase in the probability of one product being ranked 1st is associated with a decrease in the probability of another product being ranked 1st. Next, I give more details on the ranking score, and then the mapping from scores to ranking probability.

The ranking score of an app j in category g and month t is given by

$$score_{jgt} = \sum_{\tau=1}^{\tau=23} \theta_{1,\tau} Apple_j * 1\{t=\tau\} + \theta_2\check{\delta}_{jgt} + \theta_3\check{\delta}^2_{jgt} + \boldsymbol{z}^s_{jgt}\cdot\boldsymbol{\vartheta} + e_{jgt} \tag{8}$$

where $Apple_j$ indicates ownership of Apple for app j, $\check{\delta}_{jgt}$ is app quality defined in Equation (4), $\boldsymbol{z}^s_{jgt}$ is a vector of observed search-related app features, e_{jgt} is an idiosyncratic error term independently and identically drawn from the Type-I Extreme Value distribution.[45] Some examples of unobserved score shifters may be consumers' usage of apps, un-installations, and retention. To increase the plausibility of the theoretical independence of the error term, I include well-accepted search ranking shifters that are observable in $\boldsymbol{z}^s_{jgt}$,

[45] Note that there is no constant term in the score equation because only within-market relative ranking matters for identification, the exact values of rankings do not matter.

and allow their effects to be as flexible as category-specific and month-specific. Examples of such shifters include title match, price, and ratings in the previous period.[46] For a complete list of variables in vector z^s_{jm}, please see Appendix B.2.

The parameters of interest are $\{\theta_{1,\tau}\}$, each of which governs the ranking effect of Apple-ownership in month τ. Specifically, if on average, self-preferencing exists, in other words, if an Apple's app that is identical to an independent app tends to receive higher ranking, then $\theta_{1,\tau}$ will be significantly positive for some month τ. Moreover, since app quality includes consumers' preference for Apple's apps relative to independent apps, the model can directly test a typical defense for dominance of platform-owned products: "our products are ranked higher simply because they are preferred by consumers". In particular, if consumer preference is enough to justify Apple apps' higher ranking, then θ_2 should be significantly positive and $\theta_{1,\tau}$ should be insignificantly different from zero in all months.

The search ranking model is estimated with Maximum-Likelihood-Estimation(MLE). To write down the conditional log-likelihood, I firstly write down the conditional probability of an app j to be ranked 1st in category g in month t:

$$pr_{jgt}(\boldsymbol{\theta}|\boldsymbol{x}^s_{gt}) = \frac{\exp(\boldsymbol{x}^s_{jgt}\cdot\boldsymbol{\theta}^s)}{\sum_{l\in\mathscr{J}_{gt}}\exp(\boldsymbol{x}^s_{lgt}\cdot\boldsymbol{\theta}^s)}$$

where $\boldsymbol{x}^s_{jgt} = (Apple_j, \check{\delta}_{jgt}, \boldsymbol{z}^s_{jgt})$, $\boldsymbol{\theta}^s = (\theta_{1,1}, \theta_{1,2}, \cdots, \theta_{1,23}, \theta_2, \theta_3, \vartheta)$, and $\boldsymbol{x}^s_{gt}$ is a collection of $\boldsymbol{x}^s_{jgt}$ across the J_{gt} apps in category g and month t. Now, let $\boldsymbol{y}_{gt}$ denote the ordering of the J_{gt} products in category g and month t, where $\boldsymbol{y}_{gt}(k)$ is the product that is ranked at position k. Then the conditional log-likelihood for observing product orderings in the data is given by

$$\begin{aligned} L(\boldsymbol{\theta}|\boldsymbol{x}^s) &= \sum_{g,t}\log\mathbb{P}[\boldsymbol{y}_{gt}|\boldsymbol{x}^s_{gt}] \\ &:= \sum_{g,t}\log\left[pr_{\boldsymbol{y}_{gt}(1)}\cdot\frac{pr_{\boldsymbol{y}_{gt}(2)}}{1-pr_{\boldsymbol{y}_{gt}(1)}}\cdot\frac{pr_{\boldsymbol{y}_{gt}(3)}}{1-pr_{\boldsymbol{y}_{gt}(1)}-pr_{\boldsymbol{y}_{gt}(2)}}\cdots\frac{pr_{\boldsymbol{y}_{gt}(J_{gt}-1)}}{pr_{\boldsymbol{y}_{gt}(J_{gt}-1)}+pr_{\boldsymbol{y}_{gt}(J_{gt})}}\cdot\frac{pr_{\boldsymbol{y}_{gt}(J_{gt})}}{pr_{\boldsymbol{y}_{gt}(J_{gt})}}\right] \end{aligned} \tag{9}$$

where the term in the bracket in the second line is 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).

4.3 Supply

I develop a supply model to describe how independent developers choose update frequency, taking other app features as exogenous. The motivation of this supply model comes from the significant update effect and insignificant price effect, rating effect, and file size effect of the search algorithm change discussed in

[46] I argue these variables capture quite some important factors, since they match with what Apple says about their 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.". See Apple. 2021. "Apple, US developers agree to App Store updates that will support businesses and maintain a great experience for users". August 26. https://www.apple.com/newsroom/2021/08/apple-us-developers-agree-to-app-store-updates/

Section 3.2. Recall that reduced dominance of platform-owned products after the search algorithm may or may not indicate previous presence of self-preferencing. But with identified self-preferencing from the search ranking model, this supply model can then predict how update frequency and thus app quality will change with self-preferencing.

The supply model is a static two-stage game of competition on update frequency between multiple-app developers. In the first stage, app developers choose which apps are to be updated and incur heterogeneous sunk costs for apps chosen to be updated. In the second stage, app developers choose how much to update for each of the chosen apps after revolution of idiosyncratic marginal update costs. To have some concrete ideas, for one update, one can imagine that developers first gather information from users' comments, app store's new instructions, newly available technology, and new security concerns. Such efforts to gather information are irreversible in the second stage even if the update ends up dealing with only a part of the concerns. Then, developers work on programming the update, and might encounter some unexpected bugs that requires extra effort to solve, and choose the final update frequency (quality). Lastly, as a caveat, the supply model does not consider dynamic incentives of update.[47] Next, I explain the game in backwards with more details .

Stage 2 - How much to update. When choosing update frequency for apps to be updated, developers balance the expected marginal benefit from extra updates with the resolved marginal update cost. The expectation is over possible search rankings given the vector of update frequency in equilibrium. Developers' beliefs on search ranking is self-fulfilling in equilibrium. To formalize these ideas, I first specify developers' profits that are variable with update frequency, then explain developers' search ranking belief to form the expected variable profits.

App developers receive revenues from three sources: i) installations, ii) in-app purchase and subscription; iii) in-app advertising. For an app j in category g and month t, its installation revenue is $p_{jt}Q_{jgt}$, and its revenue from in-app purchase and subscription, R_{jgt}, is given by

$$\begin{aligned} R_{jgt} =& \tau_0 + (\tau_1 + \tau_2 \times game_j) \times Q_{jgt} + (\tau_3 + \tau_4 \times game_j) \times Q_{jgt}^2 + (\tau_5 + \tau_6 game_j) \times a_{jgt} \\ &+ \lambda_g^R + \lambda_t^{[0]} + game_j \times \lambda_t^{[1]} + e_{jgt}^R \end{aligned} \tag{10}$$

where $game_j$ indicates whether app j is a game app, 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 direct revenue effect of update, captured by τ_5 and τ_6. To prepare for the expected variable profit function, denote the in-app-purchase-and-subscription revenues that are variable with respect to update levels as $R(Q_{lgt}, a_{lgt}; \boldsymbol{\tau}) := (\tau_1 + \tau_2 \times game_j) \times Q_{jgt} + (\tau_3 + \tau_4 \times game_j) \times Q_{jgt}^2 + (\tau_5 + \tau_6 game_j) \times a_{jgt}$.

Revenue data allows me to estimate Equation (10) separately from the entire supply model. However, the revenues and profits from in-app advertising are unavailable. Then, I estimate an in-app-advertising

[47] Due to the continuous effects of update on quality, developers may consider the impacts of current updates on future installations. But dynamic decisions of multiple-product firms is challenging. Future work should consider a dynamic problem.

profit function together with marginal update costs. Specifically, I assume that the in-app-advertising profit that is variable with respect to update is a simple quadratic function of downloads as below.

$$F(Q_{jgt};\psi) = \psi_1 Q_{jgt} + \psi_2 Q_{jgt}^2$$

Apple collects 30% commission from independent apps' installation revenues and in-app-purchase-and-subscription revenues.[48]. Additionally, I assume zero marginal distributional cost to serve extra consumers, which is reasonable for digital products.[49] Therefore, the variable profit of developer f in category g and month t given a vector of search rankings ($\boldsymbol{y}_{gt}$) is as below.

$$\pi^I_{fgt}(\boldsymbol{y}_{gt}) = \sum_{j\in\mathscr{J}_{fgt}} 0.7 p_{jt} Q_{jgt}(\boldsymbol{y}_{gt}) + 0.7R(Q_{jgt}(\boldsymbol{y}_{gt}), a_{jgt};\tau) + F(Q_{jgt}(\boldsymbol{y}_{gt});\psi) - g(a_{jgt},\omega_{jgt};\phi) \tag{11}$$

where $\mathscr{J}_{fgt}$ is the set of apps owned by the developer f in the market. The function $g(a_{jgt},\omega_{jgt};\phi)$ describes the variable update costs at positive update a_{jgt} with unobserved cost shock ω_{jgt}. Specifically, it is given by

$$g(a_{jgt},\omega_{jgt};\phi) = \phi_1 \exp(a_{jgt}) + (\boldsymbol{z}^g_{jgt}\phi + \omega_{jgt})a_{jgt} - \phi_1$$

where $\boldsymbol{z}^g_{jgt}$ contains app age and month-fixed effects.[50] Notice that when $a_{jgt} = 0$, the variable update cost is zero: $g(0,\omega_{jgt};\phi) = 0$. One useful assumption is that ω_{lgt} are not known by developers in the first stage. Therefore, ω_{lgt} is independent of the decision on which apps are updated. Apart from mitigating the selection issue, the assumption also helps with supply-side instrument construction in Section 5.1.

Computing developers' beliefs on search rankings is challenging in large markets because the number of possible orderings of products increases factorial with the number of products. For example, in a market of 10 products, the number of possible orders becomes $10!(\approx 3.6\times 10^6)$. In the data, the number of products is 64.7 on average and could be as large as 102. Thus, it is infeasible to allow developers to consider all possible orders of search rankings in the model. And it is likewise infeasible to closely approximate the set of all possible orders.[51]

To deal with the computational challenge, I assume that developers only consider most likely orderings when making update decisions. Then I approximate these most likely orderings with a method detailed in Appendix B.3. I test the method with all markets that have no more than 10 products. Out of 10 most likely orderings, the truncated set of possible orders captures 8.5 of them on average, and captures at least 6 of

[48]The commission rate is not 30% for every app at all times. For example, for the subscription revenues from a consumer that has subscribed to the app for more than one year, Apple collects 15% of the post-one-year subscription revenues instead of 30%. However, I do not have data on the distribution of new and old consumers at app level. Therefore, I use 30% as an approximation of the actual commission payment rate.

[49]It might be costly to acquire and thus serve extra consumers. For example, Li, Bresnahan and Yin (2016) studies app developers buying downloads to get their apps on the top chart and find that the median value of one organic download is 70% the cost of buying one download. Moreover, Armstrong and Zhou (2011) theoretically investigates multiple methods that firms can pay to become prominent and thereby influence the order in which consumers consider options. But such supply-side efforts are out of the scope of this paper.

[50]Following the literature, I assume that marginal variable update cost is convex in update a_{jgt} (I expect ϕ_1 to be positive) so that the profit function is concave in update.

[51]I test with a market consisting of 10 products in the data. Denote the probability of an order y with $\mathbb{P}[y]$. To to reach $\sum_{y\in\mathscr{B}}\mathbb{P}[y] \geq 0.2$, I need at least 1% of all possible orders in the set of orders $\mathscr{B}$, which corresponds to 36,288 orders.

them, and always captures the most likely one. Note that the truncated belief space changes with update: as one app updates more, the orderings with this app in top positions is more likely to happen and thus more likely to be included in the truncated belief space. Denote a truncated belief space given a vector of updates as $\mathscr{B}_{\boldsymbol{a}}$. Then, a well-defined probability measure on $\mathscr{B}_{\boldsymbol{a}}$ is the probability of an ordering conditional on that the ordering is in $\mathscr{B}_{\boldsymbol{a}}$, which is given by

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

where $\mathbb{P}[\boldsymbol{y}|\boldsymbol{a}]$ is the probability of an ordering $\boldsymbol{y}$ as defined in Equation (9), given the vector of updates $\boldsymbol{a}$. Then, the expected variable profits of a developer f in category g and month t, given a vector of updates $\boldsymbol{a}_{gt}$, is as below.

$$\begin{aligned}
&\pi^{II}_{fgt}(\boldsymbol{a}_{gt}, \boldsymbol{\omega}_{gt}) := \mathbb{E}_{\boldsymbol{y}}[\pi^{I}_{fgt}(\boldsymbol{y})|\boldsymbol{a}_{gt}, \boldsymbol{\omega}_{gt}] = \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}_{gt}}} \pi^{I}_{fgt}(\boldsymbol{y}) \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}_{gt}] \\
&= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}_{gt}}} \left\{ \sum_{l \in \mathscr{J}_{fgt}} 0.7 p_{lgt} Q_{lgt}(\boldsymbol{a}_{gt}, \boldsymbol{y}) + 0.7 R(Q_{lgt}(\boldsymbol{a}_{gt}, \boldsymbol{y}), a_{lgt}; \boldsymbol{\tau}) + F(Q_{lgt}(\boldsymbol{a}_{gt}, \boldsymbol{y}); \boldsymbol{\psi}) \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}_{gt}] \\
&\quad - \sum_{l \in \mathscr{J}_{fgt}} g(a_{lgt}, \omega_{lgt}; \boldsymbol{\phi})
\end{aligned} \tag{12}$$

In equilibrium, the following necessary conditions hold true: marginal expected variable profit with respect to update equals zero, given the updates of others. Let D_{jgt} indicates whether app j is updated in the market. Omitting the market index gt for exposition, the necessary conditions are given by,

$$MB_j(a_j, \psi) = \phi_1 \exp(a_j) + \boldsymbol{z}^g_j \boldsymbol{\phi} + \omega_j, \qquad \forall j \; s.t. \; D_j = 1 \tag{13}$$

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

$$\begin{aligned}
MB_j(a_j, \psi) &= MB^{[0]}_j + MB^{[1]}_j \psi_1 + MB^{[2]}_j \psi_2, \\
MB^{[0]}_j &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} (0.7 p_l + 0.7 R'_l) \frac{\partial Q_l}{\partial a_j} + 0.7(\tau_5 + \tau_6 game_j) \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] \\
&\qquad + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} (0.7 p_l Q_l + 0.7 R_l) \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j} \\
MB^{[1]}_j &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} \frac{\partial Q_l}{\partial a_j} \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} Q_l \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j} \\
MB^{[2]}_j &= \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} 2 Q_l \frac{\partial Q_l}{\partial a_j} \right\} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] + \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left\{ \sum_{l \in \mathscr{J}_{f(j)}} Q^2_l \right\} \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j}
\end{aligned}$$

where $MB^{[0]}_j$ is the marginal benefit of update from the sum of installation revenues and in-app-purchase-

and-subscription revenues, the sum of $MB_j^{[1]}\psi_1$ and $MB_j^{[2]}\psi_2$ is the marginal benefit of update from in-app-advertising profit, and R_l' denotes the first-order derivative the in-app-purchase and subscription revenue function $R(Q_l, a_l; \boldsymbol{\tau})$ with respect to downloads Q_l. Note that one can compute $MB_j^{[0]}$, $MB_j^{[1]}$, and $MB_j^{[2]}$ using observed data and estimates of the demand model, search ranking model, and in-app-purchase-and-subscription revenue equation. This makes the parameters in Equation (13) linear.

There is a technical assumption underlying the first-order-condition approach: the objective function is locally differentiable with respect to update a_j. However, update might discretely affect the expected variable profits by changing the truncated set of possible search rankings $\mathscr{B}_{\boldsymbol{a}}$. Then, I assume that marginal changes in update a_j does not change $\mathscr{B}_{\boldsymbol{a}}$. Note that this assumption is not strong for two reasons. Firstly, when $\mathscr{B}_{\boldsymbol{a}}$ equals the set of all possible search rankings, this assumption is a fact. Secondly, because the rank-ordered logistic regression model only requires higher ranking score to be ranked higher rather than one-to-one mapping from score to ranking, marginal change of update a_j typically changes the ranking scores without changing the rankings (but it will change the ordering likelihood).[52]

Equation (13) captures direct and indirect incentives of updates. For example, the first term in $MB_j^{[0]}$ captures the direct effect of update on downloads, fixing search rankings; while the second term in $MB_j^{[0]}$ captures the indirect effect of updates on revenue through search ranking likelihood $\tilde{\mathbb{P}}[\cdot]$.The partial derivatives $\frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j}$ are computed based on the estimated search ranking model with step `1e-4`.

The necessary conditions imply ambiguous effect of self-preferencing on update. 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 increasing update, 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 marginal revenue from downloads, i.e., lower $(0.7p_j + 0.7R_j' + \psi_1 + 2\psi_2 Q_j)$ with smaller $ranking_j$, if the revenue curve is concave in quantity. Appendix D discusses the ambiguity implication in more details.

Stage 1 - Which apps to update. When choosing which apps to update, developers balance expected additional profits from updating an app with the sunk cost to update the app. If the developer chooses to update an app j in category g and month t, then s/he incurs a sunk cost of C_{jgt}, and learns idiosyncratic marginal update cost shocks ω_{jgt} to determine a positive update frequency in the second stage. Otherwise, the developer pays nothing and has to remain the app's update at zero in the second stage.

The first-stage objective function is expected second-stage profits net of total sunk costs. Denote the equilibrium second-stage update frequency as $\boldsymbol{a}_{gt}^{+}(\boldsymbol{D}_{gt}, \boldsymbol{\omega}_{gt})$, which is a vector function of first-stage update decisions, $\boldsymbol{D}_{gt} := (D_{1gt}, \cdots, D_{J_{gt}})$, and second-stage marginal update cost shocks, $\boldsymbol{\omega}_{gt} := (\omega_{1gt}, \cdots, \omega_{J_{gt}})$, that at least satisfies Equation (13). Partition the market-level vector of update decisions $\boldsymbol{D}_{gt}$ into developer f's update decisions, $\boldsymbol{D}_{gt}^{f} := (D_{1gt}^{f}, \cdots, D_{J_{fgt}}^{f})$, and other developers' update decisions $\boldsymbol{D}_{gt}^{-f}$, where J_{fgt} denotes the number of apps owned by developer f in market gt. Then the objective function of developer f

[52] In fact, in the data, there are only 10 out of 56,570 observations violating this assumption, whose update are then treated as exogenous.

in the first stage is given by,

$$\pi^{III}_{fgt}(\boldsymbol{D}^{f}_{gt};\boldsymbol{D}^{-f}_{gt}) := \mathbb{E}_{\boldsymbol{\omega}_{gt}}[\pi^{II}_{fgt}(\boldsymbol{a}^{+}_{gt}(\boldsymbol{D}^{f}_{gt};\boldsymbol{D}^{-f}_{gt},\boldsymbol{\omega}_{gt}),\boldsymbol{\omega}_{gt})|\boldsymbol{D}^{f}_{gt};\boldsymbol{D}^{-f}_{gt}] - \sum_{j\in\mathcal{J}_{fgt}} C_{jgt}D_{jgt} \tag{14}$$

In a Nash Equilibrium, each developer chooses which apps to update that maximizes the objective function defined in Equation (14), given the update decisions of other developers. Following Fan and Yang (2020*a*), this implies no profitable deviation from observed update decisions and enables researchers to back out bounds on sunk costs. Specifically, when an app j is not updated in category g and month t, it must be that the additional expected second-stage variable profit of its developer $f(j)$ from updating it cannot cover its fixed cost: $\forall D_{jgt} = 0$,

$$\underline{C}_{jgt} := \mathbb{E}_{\boldsymbol{\omega}_{gt}}[\pi^{II}_{f(j)gt}(\boldsymbol{a}^{+}_{gt}(1,\boldsymbol{D}_{-jgt},\boldsymbol{\omega}_{gt}),\boldsymbol{\omega}_{gt})|1,\boldsymbol{D}_{-jgt}] - \mathbb{E}_{\boldsymbol{\omega}_{gt}}[\pi^{II}_{f(j)gt}(\boldsymbol{a}^{+}_{gt}(0,\boldsymbol{D}_{-jgt},\boldsymbol{\omega}_{gt}),\boldsymbol{\omega}_{gt})|0,\boldsymbol{D}_{-jgt}] \leq C_{jgt} \tag{15}$$

On the other hand, when an app j is updated in category g and month t, it must be that the additional expected second-stage variable profit of its developer $f(j)$ from updating it can fully cover its fixed cost: $\forall D_{jgt} = 1$,

$$\bar{C}_{jgt} := \mathbb{E}_{\boldsymbol{\omega}_{gt}}[\pi^{II}_{f(j)gt}(\boldsymbol{a}^{+}_{gt}(1,\boldsymbol{D}_{-jgt},\boldsymbol{\omega}_{gt}),\boldsymbol{\omega}_{gt})|1,\boldsymbol{D}_{-jgt}] - \mathbb{E}_{\boldsymbol{\omega}_{gt}}[\pi^{II}_{f(j)gt}(\boldsymbol{a}^{+}_{gt}(0,\boldsymbol{D}_{-jgt},\boldsymbol{\omega}_{gt}),\boldsymbol{\omega}_{gt})|0,\boldsymbol{D}_{-jgt}] \geq C_{jgt} \tag{16}$$

Therefore, Equation (15) obtains the lower bounds ($\underline{C}_{jgt}$) on the fixed costs for apps that are not updated, and Equation (16) obtains the upper bounds ($\bar{C}_{jgt}$) on the fixed costs for apps that are updated. I explain how equilibrium update decisions are computationally found in Section 6.1.

5 Estimation

5.1 Estimation Procedure

Demand. The estimation of demand is similar to that in Berry, Levinsohn and Pakes (1995). I construct moments using Equation (7) and estimate the parameters using GMM. However, there are richer endogeneity concerns in this paper. Specifically, update (a_{jgt}) is endogenous because developers know the unobserved demand shocks (η_{jgt}) when updating apps. Moreover, price, search ranking, and average ratings may also be correlated with the unobserved demand shocks. For example, advertising is unobserved and might be correlated with the above app characteristics and directly affects demand.

For the endogeneity concerning app characteristics, following the literature, I firstly construct basic instruments using the characteristics of other apps owned by the same developer in the other categories in the same month. These basic instruments follow the idea of the Hausman-type(Hausman and Bresnahan, 2008) instrument: common unobserved cost shifters among products owned by the same firm. Then, in the case of single-category developers, I construct instruments based on the basic instruments. In particular, I calculate the average basic instrument values of the apps of the competing developers in the same market. These instruments are similar to BLP instruments – they are characteristics of the apps of competing developers.

However, they are more indirect than BLP instruments because these characteristics are based on other apps in other markets.[53] The idea is that when a developer chooses app characteristics, s/he considers the cost features of her/his competitors, which are partially captured by their behaviors in other markets.[54]

As for the endogeneity in search rankings, I construct instruments based on the title match of the app with popular keywords in the market. The above estimation strategy relies on three assumptions: i) markets are independent; ii) the unobserved demand shocks are realized after app entry (but before update choices); iii) titles do not directly affect app values. The first two assumptions are commonly used in the literature. As for the third assumption, I control for systematic time effects using month-fixed effects. Therefore it seems reasonable that any app/category/month-specific shocks are uncorrelated with app titles. In addition to the above instruments, I include interaction terms between the post-July-2019 indicator and i)Apple-ownership indicator and ii)Apple-competitor indicator in the instruments.[55] These instruments are based on the exogenous search algorithm change in July 2019 that particularly affects categories with Apple's apps. The first-stage regression results for the above instruments are shown in Appendix B.4.

Search Ranking. The estimation of the search ranking model is based on MLE, where the log-likelihood function is given in Equation (9). Note that the estimation sample includes pre-installed apps, while in demand estimation, pre-installed apps are lumped into out-side option. Therefore, the number of observations for estimating the search ranking model exceeds that of demand model. Because the score equation (8) is quite flexible with category-specific and month-specific coefficients, I argue that there is no obvious endogeneity concern.

Update Costs. The estimation of supply has two steps. First, a reduced-form in-app-purchase and subscription revenue function is estimated based on Equation (10) in OLS, where invalid observations are dropped from the sample.[56] Then, I construct moments using Equation (13) and estimate the parameters using GMM. Notice that only positive update levels are included in the sample for estimating Equation (13). Similar to the demand model, there are endogeneity concerns. Specifically, update levels may be endogenous because developers know unobserved (to researchers) marginal update cost shocks(ω_{jgt}) when choosing the positive update levels. For example, an app might receive particularly positive and constructive comments in a certain month, and the developer is inspired to think of a straightforward way to improve user experiences. These unobserved comments increase the marginal benefits from the update and decrease the marginal cost of update, leading to higher update levels. Such omitted variables will bias the coefficient on update in Equation (13) upwards.

I construct instruments based on category-fixed effects and pre-determined characteristics, including price, file size, and title match with popular keywords. This estimation strategy is based on the timing

[53] Because update levels are endogenous, I cannot use classical BLP-type(Berry, Levinsohn and Pakes, 1995) instruments (e.g., average update levels of apps of competing developers).

[54] An average multiple-category developer develops 2.3 apps in 2.5 categories. The larger average number of apps versus categories reflects that some apps operate in multiple categories simultaneously. In such cases, the characteristics of the same app in the other categories will not be used to construct instruments.

[55] Because category-fixed effects are included in the excluded instruments, the Apple-competitor indicator is absorbed.

[56] Specifically, I drop observations that i) have negative in-app-purchase and in-app-subscription revenues; or ii) do not provide in-app-purchase or in-app-subscription services. Negative in-app-purchase-and-subscription revenues may reflect flaws in the estimated revenues from AppTweak.

assumptions that i) update portfolios are determined after unobserved demand shifters(ξ) and the above pre-determined characteristics, and ii) marginal update cost shocks are realized after update portfolios. The first-stage regression results for these supply-side instruments are reported in Appendix B.4. During the estimation, I apply constraints to guarantee profit maximization conditional on the positive update levels of competing apps, which are listed in Appendix B.5.

As for fixed costs, I use the inequalities (15) and (16) to obtain bounds. Following Fan and Yang (2020*a*), I calculate the bounds by calculating changes in expected second-stage variable profits from adding an un-updated app into update portfolios or dropping an updated app from update portfolios. The expectation is over marginal update cost shocks ω_m and possible search rankings. To compute the expected variable profits, I draw the cost shocks from their empirical distribution. Then, I compute the update equilibrium in the second-stage game for each cost-shock draw, which returns the second-stage (expected) variable profits. Then, I take the average of these second-stage variable profits across all cost-shock draws.

There is a computational burden when evaluating the equilibrium second-stage variable profits, mainly due to the high dimension of possible search rankings.[57] To alleviate the computational burden, I restrict the sample for computing the fixed-cost bounds with two steps. First, I focus on relevant markets. Specifically, I only consider apps in categories with Apple's apps during the difference-in-differences sample period (Jun.-Nov.,2019).[58] It reduces the number of markets from 874 to 96. Second, following the literature, I focus on the top5 developers in each category and only allow these top5 developers to change their updates.[59] With these restrictions, there are 556 fixed-cost upper bounds and 176 fixed-cost lower bounds to be estimated.

5.2 Estimates of Demand

Table 4 reports the estimates for parameters of the demand model. The demand estimation results show that an average consumer prefers apps with higher update levels, Apple ownership, higher average ratings, more experiences, larger file size, fewer screenshots, in-app-purchase availability, and lower installation price. For example, an average consumer is willing to pay $5.2 more for downloading an app developed by Apple than one developed by an independent developer. It might justify the observed high search rankings for Apple's apps and is taken into account by the search ranking model. In addition, the estimated standard deviation for consumers' taste for update levels is about 4.6 times the average taste, suggesting consumers are quite heterogeneous in their tastes for update levels. Lastly, the results show that consumers are more likely to incur higher search costs when searching for an app with lower (larger) search rankings in the top50 search results or an app absent from the top50 search positions.

Table 5 shows the price semielasticities (Panel A) and search ranking semielasticities (Panel B) for

[57] For example, an average market with 65 products has 141 different orderings of products in the truncated set of possible search rankings. So then, there are 9165 ($= 65 \times 141$) market shares to be computed. And such computation is embedded in the evaluation of each new vector of update levels when firms choose their best-response update levels. It is as if computing price equilibrium for a market with 9165 products. For more details on finding the equilibrium positive update levels, please see Appendix B.6.

[58] The purpose of computing the bounds is to simulate counterfactual update equilibrium when shutting down the self-preferencing. Therefore, only the categories with Apple's apps will have different counterfactual equilibrium than status-quo equilibrium.

[59] The category-specific top5 developers are found based on total downloads of owner apps during the post-change difference-in-differences sample period.

Table 4: Estimation Results for Demand

Variables	Parameter	Standard error
Quality Coefficients		
log(1+Update Frequency)	0.155	0.074
Apple	1.115	0.447
Average Rating	0.527	0.172
log(Age) (month)	0.330	0.143
log(File Size) (MB)	0.422	0.054
#Screenshots	−0.039	0.015
log(1 + Description Length)	0.033	0.121
Offer In-App-Purchase	0.230	0.134
Game	−0.037	0.146
Constant	−13.840	1.019
One-month Lagged Unobserved Mean Relative Utility	0.921	0.004
Outside Value: Exists Pre-installed Apps	−0.255	0.134
Price	−0.216	0.046
Paid	−1.754	0.239
Random Coefficients		
Update Level	0.711	0.105
Search Cost Parameter		
Ever Top50 in Search Results	−5.125	2.601
log(Search Ranking)|Ever Top50	1.374	0.737
Month-FE	YES	
Observations	52,959	

Notes: Update level is $\log(1 + \text{update frequency})$. Description length is in the unit of 1,000 characters.

Table 5: Demand Semielasticities with respect to Price and Search Ranking

	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	0.003
Hulu	0.010	0.007	−0.212	0.003
Amazon Prime Video	0.009	0.006	0.003	−0.214
	Panel B. Ranking Semielasticities			
Netflix	−0.174	0.007	0.004	0.002
TikTok	0.008	−0.175	0.003	0.002
Hulu	0.008	0.006	−0.173	0.002
Amazon Prime Video	0.007	0.005	0.003	−0.174

Notes. The top panel reports percentage change in market share of the column-product with a $1 increase in the row-product's installation price. The bottom panel reports percentage in market share of the column-product with a ten-position decline of the column-product's search ranking.

the top four apps in the entertainment category in July 2019. These apps are Netflix, TikTok, Hulu, and Amazon Prime Video. Panel A shows that a $1 increase in the installation price of an app leads to about 0.2% decrease in its demand.[60] Panel B shows that a ten-position decline in the search ranking of an app leads to about 0.17% decrease in its demand. Unsurprisingly, the own price (ranking) semielasticities are larger than the cross semielasticities in magnitude. Dividing the own ranking semielasticities by the own price semielasticities gives an intuitive measurement for position effect: a ten-position decline in search ranking is equivalent to increasing price by $0.85. This position effect is relatively small compared to those found in the other industries. [61] It suggests that some consumers are likely to know their match values with apps before searching (which is allowed by the assumed distribution of search costs), and thus the average search costs are small across consumers. For example, it may happen when some apps advertise a lot out of the App Store. Furthermore, Figure F.9 illustrates an inverse U-shape curve for estimated position effects across search rankings. It indicates i) inelastic demand for apps with high search rankings and ii) high search costs for apps with low search rankings.

Table 6: Demand Elasticities with respect to Update Level

	Netflix	TikTok	Hulu	Amazon Prime Video
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. The table reports percentage change in market share of the column-product with a 1 percent increase in the row-product's update level. Update level is $\log(1 + \text{update frequency})$.

Table 6 shows the elasticities of update level for the same top four apps in the same market. Across the four apps, a 1 percent increase in the update level is associated with a 0.6 percent to 1.4 percent increase in market shares. Similarly, the own update level elasticities are larger than the cross elasticities.

5.3 Estimates of self-preferencing

Table 7 reports the estimates for parameters of the search ranking model. The results are intuitive: higher-ranked apps are associated with higher quality, lower installation price, more text relevance with popular keywords, and more one-month lagged 5-star ratings. Furthermore, it shows that self-preferencing exists in the search algorithm of the Apple App Store. Specifically, in June and July 2019, the ranking score of Apple's apps are significantly higher than independent apps by 1.5 and 1.3. To mitigate such a disadvantage in the search algorithm, an average independent app needs to decrease installation price by $73.9($\approx 1.551/0.021$)

[60] I do not compute price elasticity because these four apps are free. The price semielasticities for the top four paid apps in the same market are similar.

[61] 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.

in June 2019 or \$60.0($\approx 1.261/0.021$) in July 2019.[62]

Table 7: Estimation Results for Search Ranking

Variables	Parameter	Standard error
Apple$\times$1{June 2019}	1.551	0.366
Apple$\times$1{July 2019}	1.261	0.235
Quality	0.167	0.019
Squared Quality	0.002	0.001
Paid	−0.512	0.077
Price	−0.021	0.005
Title Match with Popular Keywords	12.780	1.036
Subtitle Match with Popular Keywords	2.785	1.007
One-month Lagged 5-star Ratings	5.365	0.575
Pre-Install	−0.196	0.635
Full Controls	YES	
Observations	53,245	

Notes: The other interaction terms between Apple and Month indicators are included in the model but not shown here. Appendix B.2 lists variables included in the full controls.

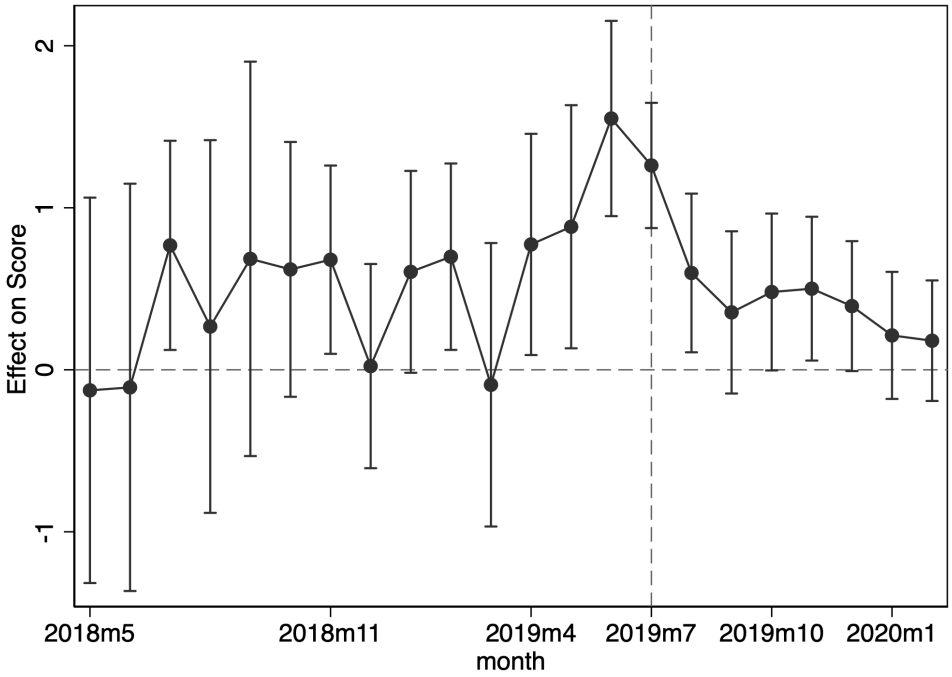
Figure 3 presents the coefficients on the Apple-ownership indicator in each month. It illustrates that the self-preferencing evolves through four periods: i) before April 2019, it was weak if any; ii) from April 2019 to June 2019, it starts to increase and reaches the peak; iii) from July 2019 to August 2019, it starts to decrease; iv) after August 2019, it basically disappears.

To understand the four periods of the self-preferencing, I start with the latest two periods: from July 2019 to February 2020. The results are consistent with the search algorithm change studied in Section 3. In fact, when normalizing the self-preferencing parameter ($\theta_{1,\tau}$) in July 2019 as zero, Figure F.7 shows that the self-preferencing parameters in and after August 2019 are significantly lower than those in June and July 2019. Therefore, the search algorithm change in July 2019 provides good cross-validation for the structural estimates.

The first two periods seem to be inconsistent with the presented average search rankings of Apple's apps in Figure 1. Specifically, it suggests that Apple's apps deserve the observed high search rankings during the first period, and as independent apps become stronger competitors for Apple's apps, the self-preferencing kicks in during the second period. This interpretation can be shown by comparing the observed rankings to rankings of residual downloads. In particular, suppose the self-preferencing was stronger in the first period compared to the second period, as suggested by Figure 1. In that case, the gap between the observed rankings and rankings of residual downloads should be larger in the first period compared to the second

[62]Figure F.10 presents the histogram of static developer-fixed effects across all developers. To estimate the developer-fixed effects, I replace the interaction terms between the Apple-ownership indicator and month indicators in Equation (8) with developer-fixed effects, where the single-product developers are normalized as the reference group. Figure F.10 shows that, even when allowing all developers to have their own advantage or disadvantage in the search ranking algorithm, Apple ownership still generates larger advantages than most developers.

Figure 3: Apple self-preferencing Parameters across Months



Notes. The figure presents point estimates of the effects of Apple ownership on ranking score in each month during the sample period.Error bars indicate 90% confidence interval using standard errors clustered at the category-month level.

period. However, Figure F.8 shows the opposite. It supports the above interpretation and explains the seemingly inconsistent trends. This discussion suggests the importance of identifying self-preferencing rigorously and the incompleteness of information from only observed rankings.

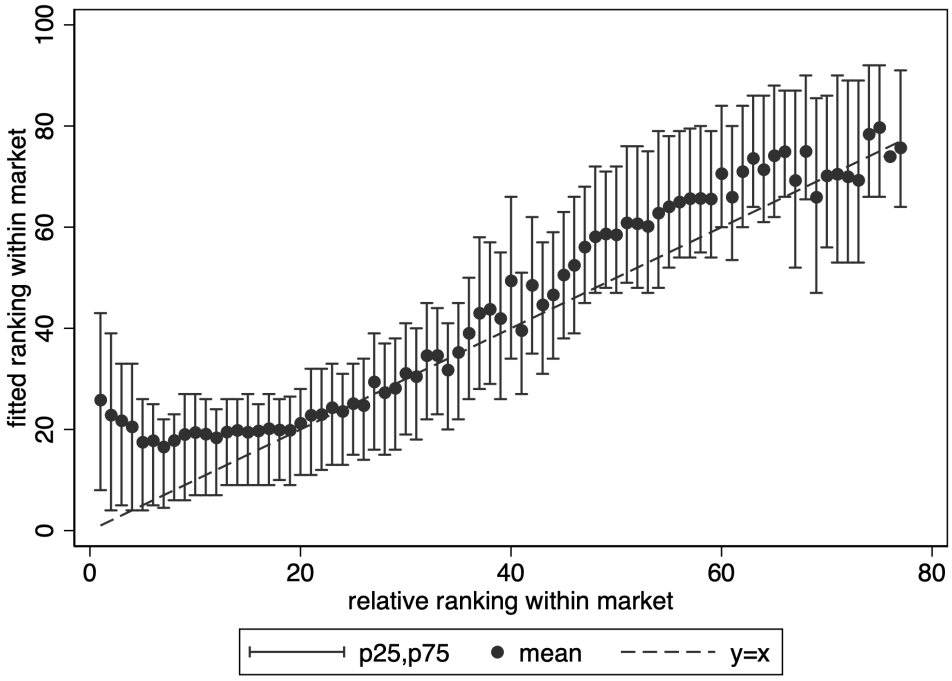
Figure 4 shows the fitness of the search ranking model. It plots the model-fitted most likely within-market relative rankings against the observed within-market relative rankings.[63] It shows that the model fits the data relatively well. For example, for most of the relative rankings, the average fitted relative rankings are close to the observed ones; and the interval between the first and the third quartiles of the fitted relative rankings covers the observed ones. In addition, Figure F.11 particularly shows the fitness for Apple's apps. App developers' beliefs on possible search rankings are constructed based on the estimated search ranking model.

5.4 Estimates of Revenues and Update Costs

Table 8 reports the estimates for parameters of the supply model. First, the estimation results for in-app-purchase and in-app-subscription revenues show intuitive results: i) revenues increase with downloads concavely, and ii) update levels directly contribute to revenues, especially for game apps. The estimates imply that an average app receives $4.3 from in-app-purchase and in-app-subscription with each new download (consumer).

[63]For example, the search ranking of an app might be 33.5 in a market where there are nine apps whose search rankings are strictly higher (smaller) than 33.5. Then, the within-market relative ranking of this app is 10. Ties are considered in the search ranking model.

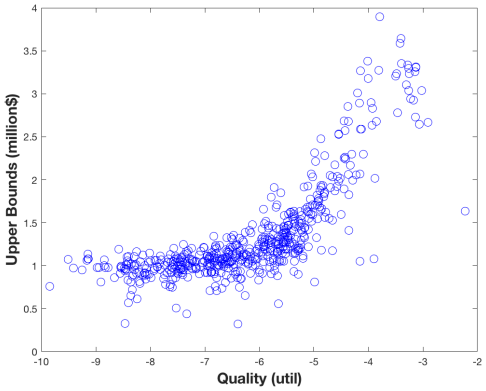
Figure 4: Rank-ordered Logistic Regression Model Fitness



Notes. The figure presents predicted most-likely within-market ranking (y-axis) against observed within-market ranking (x-axis) across markets. Bars indicate the 25 percentile and 75 percentile of fitted within-market rankings across apps that have ranked at the given observed within-market ranking.

Figure 5: Bounds of Fixed Costs of Updates (million $) *v.s.* Quality

Panel A. Upper Bound



Panel B. Lower Bound

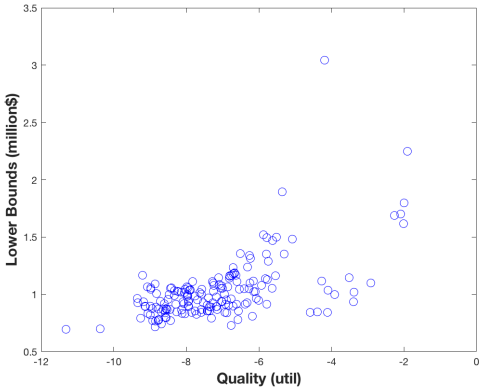


Table 8: Supply Model Estimates

Variables	Parameter	Standard error
In-App-Purchase and In-App-Subscription Revenue Parameters		
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 ($)	4.298	
Observations	37,382	
In-App-Advertising Profit Parameters		
Downloads	0.068	0.429
Squared downloads	−0.003	0.085
Marginal Update Costs Parameters		
Update Frequency	1.631	0.052
Age	−0.002	0.000
Constant	−3.731	0.146
Month-FE	YES	
Average Marginal In-App-Advertising Profit ($)	0.067	
Average Marginal Cost (million$)	0.323	
Observations	25,325	

Notes: Update level is $\log(1 + \text{update frequency})$. Marginal revenue and marginal profit are with respect to downloads. Marginal update cost is with respect to update levels.

Second, the estimation results for the update-level first-order conditions show that i) in-app-advertising profits insignificantly increase with downloads; and ii) marginal update costs are larger for apps with higher update levels and/or fewer experiences. The results imply that an average app earns \$0.07 from in-app advertising with each new download. The resulting average marginal update cost is \$0.32 million. To give some contexts for the magnitudes, I do the following back-of-envelope calculation. One weighted update is equivalent to $\log(2)$ update levels, and the average salary for a computer engineer is about \$97,000 per year in California. Then, the average marginal update cost is equivalent to hiring 28 ($\approx 12 \times \log(2) \times 0.32/0.097$) computer engineers in a month.

As for fixed costs of updates, I obtain upper bounds for each updated app and lower bounds for each app that is not updated in each market. Figure 5 plots the estimated upper bounds (Panel A) and lower bounds (Panel B) on the vertical axes against app quality on the horizontal axis. The average upper bound in Panel A is \$1.34 million, and the average lower bound in Panel B is \$1.23 million.

In addition, as cross-validation of the model fitness, Appendix C.1 compares structural and reduced-form estimates of the average treatment effect of the search algorithm change in July 2019. It shows that both methods generate the same direction and similar magnitudes of changes in updates and search rankings. The structural estimate of the average treatment effect has the same sign as the DiD estimate but with smaller magnitudes, which is explained in the appendix.

6 Counterfactual Simulations

This section uses the estimates from the demand, search ranking, and supply models to examine counterfactual simulations. Specifically, I focus on changes in market equilibrium when shutting down the identified self-preferencing. The changes are used to quantify the effects of self-preferencing. In particular, I first examine the effects on independent apps' qualities, then examine the equilibrium welfare effects on consumers and independent developers.

The counterfactual simulations cover eight markets with identified self-preferencing. The markets are the entertainment category, health & fitness category, music category, and utilities category in June and July in 2019. The two months are estimated to have the strongest self-preferencing, i.e., the largest significant coefficients on the Apple-ownership indicator, during the sample period. When shutting down the self-preferencing, the coefficients on the Apple-ownership indicator are set to be zero in the search ranking model.

6.1 Effects of self-preferencing on Update Frequency and App Quality

App quality is positively affected by update frequency on average, as estimated in the demand model. Changes in update frequencies then lead to changes in app qualities. I take two main steps to find counterfactual update frequencies and app qualities. First, I take simulation draws of fixed costs of update from a range that is consistent with the identified bounds. Second, for each simulation draw of fixed costs, an equilibrium set of update portfolios across developers is computed based on best-response iterations. Given

each equilibrium of update portfolios, I compute the second-stage equilibrium positive update frequency with estimated variable update cost shocks ($\hat{\omega}$) whenever possible. [64]

In the first step, the sunk-cost draws are drawn in a way following Fan and Yang (2020*a*). Specifically, for each updated app in the data, I have obtained an upper bound for the fixed cost of updating it, $\overline{C}_{jm}$. For such an app, I uniformly draw five sunk-cost draws from the range $[0.5\overline{C}_{jm}, \overline{C}_{jm}]$. 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, $\underline{C}_{jm}$. For such an app, I uniformly draw five sunk-cost draws from the range $[\underline{C}_{jm}, 5\underline{C}_{jm}]$.

In the second step, there are two layers of best-response iterations. The top layer is the best-response iterations of update portfolios, which could be challenging with multiple-product firms. For example, one affected developer has up to 12 apps, which creates a choice among $2^{12}(= 4096)$ update portfolios. To alleviate the computational burden, I apply the heuristic algorithm for finding the best-response product portfolio from Fan and Yang (2020*a*) to find the best-response update portfolio. The bottom layer is the best-response iterations of positive update levels for apps chosen to be updated. [65]

As explained in Section 5.1, only the fixed costs of apps owned by the category-specific top5 independent developers are computed. Therefore, in the counterfactual simulations, only these top5 developers are active players in the update competition game, and the updates of the other apps are holding fixed. As a result, in an average market for the counterfactual simulations, there are ten apps whose updates are subject to changes, while there are 90.4 independent apps in total. On average, the active apps account for 45.1% of total downloads in a market. In Appendix C.5, I allow all independent developers to choose update levels but holding update portfolios fixed. The results are robust.

Additionally, I fix the search costs of outside options throughout the counterfactual simulations. In the model, outside option lumps i) not using any app, ii) using previously downloaded apps, and iii) using pre-installed apps. In the first two cases, there is no search costs. In the last case, search costs for pre-installed apps change with their search rankings. However, the search algorithm change in July 2019 is mainly driven by changes of non-preinstalled Apple's apps' search rankings, which implies the rigidity of pre-installed apps' search rankings.[66] Therefore, I fix the search rankings of pre-installed apps when eliminating the identified self-preferencing. Given that the empirical model for estimation does not fix the search rankings of pre-installed apps, for consistency, the reported status-quo market outcomes with self-preferencing are the results from fixing pre-installed apps' search rankings. It turns out that the changes of equilibrium due to fixing pre-installed apps' search rankings are small. For details on the process and effect of fixing the pre-installed apps' search rankings, please see Appendix C.2.

[64] Therefore, as a rigorous interpretation, the simulated outcomes are the expected outcomes of the simulated second-stage update competition game, given the simulated first-stage equilibrium update portfolios and the identified variable cost shocks($\hat{\omega}$). The expectation is over draws of variable cost shocks for apps that are not updated in the data but simulated to be updated in the counterfactual. These variable cost shocks are drawn from the empirical distribution of $\hat{\omega}$. This step ensures that the difference between the simulated update levels and observed update levels is not due to different variable cost shocks. Finally, these update levels are transformed into update frequencies using the definition equation that update level equals $\log(1+\text{update frequency})$ and fed into the definition Equation (**??**) to calculate app qualities.

[65] In the bottom-layer best-response iteration, the variable update cost shocks(ω) are the same as those in the computation of bounds on fixed costs.

[66] Figure F.3 shows the average search ranking of non-preinstalled Apple's apps around July 2019, which sees a more significant change after July 2019 than the average search ranking of all Apple's apps in Figure 1.

Table 9: Effects of Self-preferencing on Update Levels

	Status-quo	Shut-down	Percentage Change(%)			
	mean	mean	mean	min	max	std
Market-Level Results						
Number of Updated Apps	6.98	7.03	0.71	0	2.86	1.26
Average Update Frequency	1.170	1.174	0.44	−0.02	1.25	0.51
Average App Quality	−6.0833	−6.0829	0.01	−0.001	0.02	0.01
Product-Level Results						
Probability of Update[a]	0.706	0.711	0.01	−0.2	0.2	0.05
Expected Update Frequency[b]	1.14	1.15	0.28	−19.97	25.14	4.35
Expected App Quality	−6.270	−6.269	0.01	−0.28	0.30	0.06

Notes: Update frequencies are monthly number of updates weighted by length of release notes. Variables are computed for apps owned by the category-specific top5 independent app developers. For the status-quo case, there is identified self-preferencing. For the shut-down case, there is no self-preferencing. [a]For probability of update, the reported values in the last four columns are summary statistics on changes instead of percentage changes. [b]For expected update frequency, the summary statistics on percentage change are conditional on upgrading in status-quo. The expectation is over draws of sunk costs.

Table 9 shows the counterfactual simulation results for update frequencies and app qualities of independent apps with and without the identified self-preferencing. Overall, I find small average effects of eliminating self-preferencing. The top panel shows that, in an average market, eliminating self-preferencing leads to a 0.7 percent increase in the number of updated independent apps and a 0.4 percent increase in average update frequencies. These changes in updates result in a 0.01 percent increase in average app qualities. The bottom panel shows that, for an average independent app, after eliminating self-preferencing, the probability of update increases by 0.01, and the expected app quality increases by 0.01%. Conditional on updating before eliminating self-preferencing, an average independent app increases expected update frequency by 0.3% after the elimination. [67]

However, large but heterogeneous effects might be hidden behind the small average effects when updates may increase or decrease after eliminating the self-preferencing. Recall that theory predicts ambiguous effects of self-preferencing on product quality, which implies the possibility of heterogeneous effect of self-preferencing on updates across products. Therefore, I examine the heterogeneous effects in the last three columns of Table 9. Specifically, compared to the small average effects, the most astonishing result shows up in changes of expected update frequencies, where I find that an independent app's expected update frequency might decrease by 20% and might as well increase by 25%.[68] These effects are large relative

[67]Notice that these counterfactual simulations fix pre-installed apps' search rankings, which limits the magnitude of effects. Therefore, the reported effects here are smaller than those estimated from the difference-in-differences analysis(Table3), even though the self-preferencing is completely shut down (which is not achieved by the search algorithm).

[68]Moreover, 38 percent of the estimated effects on expected update frequency are positive, and 19 percent of the estimated effects are negative, which constitutes 57 percent of observations that see changes of expected update frequency and quality in the counterfactual simulations. Furthermore, conditional on updating before the elimination, the average positive effect is a 1.3 percent increase of expected update frequency, the average negative effect is a 1.4 percent decrease of expected update frequency. Both of

to the average effects. Moreover, the standard deviation of the effects on the expected update frequencies is 15.5 times its mean, again indicating sizable heterogeneity in the effects on updates. Similarly, I find both negative and positive effects on the probability of update and expected app quality. Not only does the heterogeneity exists at the product level, but I also find both negative and positive effects on average update frequencies and average app qualities at the market level, except that no market has a decrease in the number of updated apps.[69] Overall, there is sizable heterogeneity in the direction and magnitude of the effects of self-preferencing on update frequency and app quality.[70]

6.2 Effects of self-preferencing on Welfare

Given the counterfactual update frequencies and app qualities, I calculate counterfactual search rankings, installations, consumer surplus, and developer profits. The welfare of a consumer is defined as the utility of the app chosen by the consumer net of the total search costs incurred by the consumer to find the app in dollars.[71] Specifically, expected consumer surplus in category g and month t is given by[72],

$$CS_{gt} := M_{gt} \cdot \mathbb{E}_{(\boldsymbol{\varepsilon},\boldsymbol{c},\boldsymbol{\sigma},\boldsymbol{y})} \left[-\left(u_{ij(i)gt} - \sum_{l \in \mathscr{S}_i} c_{ilgt}\right) / \alpha \right] \tag{17}$$

where M_{gt} is the market size of market gt, $j(i)$ is the chosen app by consumer i, $\mathscr{S}_i$ is the set of apps that are searched by consumer i, and α is the identified price(income) coefficient in the demand model. The expectation is over i) the vector of consumer-app-specific unobserved match-values ($\boldsymbol{\varepsilon}$), ii) the vector of consumer-app-specific search costs ($\boldsymbol{c}$), iii) consumer-specific random coefficients over updates ($\boldsymbol{\sigma}$), and iv) the vector of app-specific search rankings ($\boldsymbol{y}$). Changing self-preferencing potentially changes both the chosen app ($j(i)$) and the set of searched apps ($\mathscr{J}_i$). For more details on the computation, please see Appendix C.3.

To measure effects on independent developers, I calculate changes in variable profits and profits. Specifically, variable profit is measured as the total revenues from installation, in-app purchase and in-app subscription, plus in-app-advertising profits and minus variable update costs. Profit is variable profit minus fixed update costs.

Table 10 shows the effects of eliminating self-preferencing on search rankings, installations, and welfare. The first row confirms that eliminating self-preferencings does not change average search rankings; it only re-allocates the positions among apps in the same market.[73] The second and third rows show that, in an

the average positive effect and the average negative effect are larger than the average effect.

[69] All markets see changes in average update frequencies and average app qualities. 62.5 percent of the markets see positive changes, while 37.5 percent of the markets see negative changes. Additionally, one-quarter of the markets see changes in the number of updated apps, and 5 percent of observations see changes in expected update-or-not choice.

[70] Appendix C.4 explains why some independent apps increase update frequency after eliminating the identified self-preferencing while others decrease update frequency.

[71] Kim, Albuquerque and Bronnenberg (2010) and Ursu (2018) define consumer welfare in the same way.

[72] All expressions for expected consumer surplus are up to a constant. See Small and Rosen (1981).

[73] Therefore, eliminating self-preferencing is a differential change in search costs across products instead of a universal reduction in search costs. In particular, eliminating the identified self-preferencing decreases the search costs for some independent apps while increasing the search costs for Apple's apps. It makes the research question different from most papers in the literature on information frictions and product competition.

average market, eliminating the self-preferencing lowers down Apple's apps by 17.5 positions and boosts up independent apps by 0.6 positions in search. It corresponds to a 142 percent decline of search rankings for Apple's apps and a 1.6 percent rise of search rankings for independent apps.[74]

Table 10: Effects of Self-preferencing on Search Rankings, Installations and Welfare

	Variable	Status-quo	Shut-down	Mean Δ	Mean $\%\Delta$
(1)	Average Search Rankings	38.92	38.92	0	0
(2)	- Independent apps	39.86	39.23	-0.63	-1.56
(3)	- Apple's apps	14.35	31.85	17.51	142.08
(4)	Total Installations (million)	15.16	15.40	0.24	1.56
(5)	- Independent apps	15.06	15.32	0.26	1.75
(6)	- Apple's apps	0.10	0.08	-0.03	-26.58
(7)	Consumer Surplus (million \$)	321.40	321.95	0.548	0.172
(8)	Variable Profits (million \$)	76.60	77.04	0.448	0.678
(9)	Profits (million \$)	69.27	69.66	0.396	0.680
(10)	Total Search Costs (million \$)	15.39	15.35	-0.04	-1.53
(11)	Total Realized Utility (million \$)	336.79	337.31	0.51	0.15

Notes: For the status-quo case, there is identified self-preferencing. For the shut-down case, there is no self-preferencing. Variable profits and profits are calculated based on independent developers.

From the fourth row to the ninth row, Table 10 shows the effects of eliminating self-preferencing on installations and welfare. Specifically, after the elimination, in an average market, the total installations of independent apps increase by 1.8%, while the total installations of Apple's apps decrease by 26.6%. Because Apple's apps only account for a small portion of the products in the markets, the total installation in an average market increase by 1.6%. These changes lead to a \$0.55 million increase in consumer surplus and a \$0.40 million increase in profits of independent developers in an average market.[75] In each month, four categories are identified to be affected by self-preferencing. Thus, multiplying the per-market effects by four, it gives a per-month consumer surplus gain of \$2.2 million and a per-month profit gain of \$1.6 million for independent developers. To give some context of the magnitudes of the welfare effects, the per-month total installation payments across the affected categories are \$1.32 million, and the per-month total consumer payments (installation payments plus in-app purchase and subscription payments) are \$160.5 million. Therefore, the total surplus gain in a month from the elimination is equivalent to saving three-month consumers' installation payments or 2 percent of per-month total consumer payments.

Consumers may be better off due to reduced search costs or increased match values. To understand which channel dominates the gains in consumer surplus, the last two rows of Table 10 decompose the consumer welfare. It shows that most of the gains come from more efficient consumer-app matches, which generates 92.7% ($\approx$51/55) gains in consumer surplus. On the other hand, the reduced search costs only

[74] The smaller changes in the search rankings of independent apps relative to Apple's apps are due to the large number of independent apps compared to the number of Apple's apps in the markets.

[75] Profits of Apple's apps decrease by \$0.001 million in an average market.

account for 7.3% (≈4/55) gains in consumer surplus.[76]

To understand the contribution of the increased quality to the welfare gain, I simulate the case of shutting down self-preferencing but holding update levels fixed. The results of this non-game simulation are shown in the third column of Table 11. The first column of Table 11 copies the corresponding values in Table 10. Comparing the two columns shows that update adjustment accounts for small welfare gains. Specifically, consumer surplus increases by $0.53 million after eliminating the self-preferencing with fixed updates. Therefore, update adjustment accounts for 3.6% (≈0.02/0.55) gains of equilibrium consumer surplus with update adjustment. On the supply side, with fixed updates, variable profits of independent developers increase by $0.41 million after the elimination. This is $0.14 million more increase than that in the first column due to increased update costs with update adjustment. Therefore, update adjustment accounts for 3.5% (≈0.14/0.396) gains of equilibrium variable profits for independent developers with update adjustment. Overall, increased quality weakly contributes to the welfare gain from eliminating the identified self-preferencing.

Table 11: Welfare Effects with and without Update Adjustment

	With Update Portfolio and Level Adjustment	With Conditional Update Level Adjustment	Without Update Adjustment
Mean ΔCS (million $)	0.548	0.551	0.531
Mean %ΔCS (%)	0.172	0.173	0.167
Mean ΔVariable Profits (million $)	0.448	0.408	0.410
Mean %ΔVariable Profits (%)	0.678	0.618	0.621

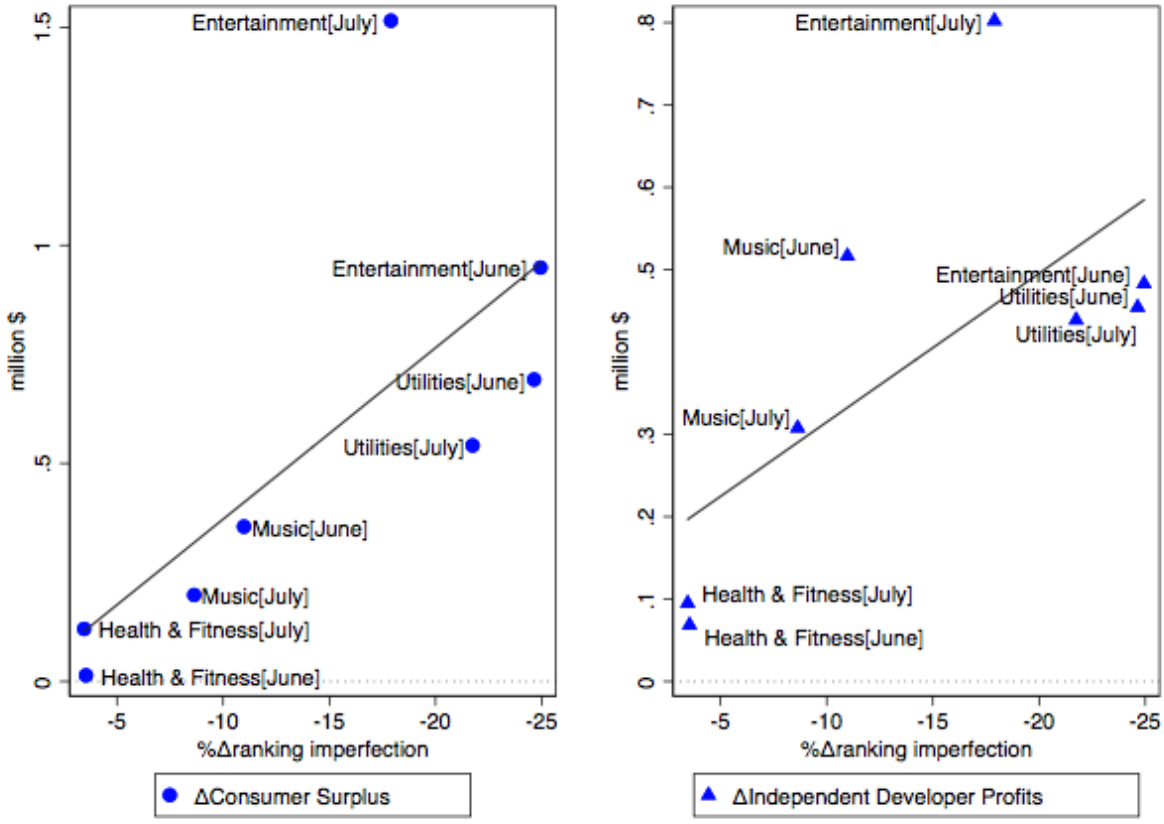
Notes: Update portfolio and level adjustment holds updates of non-top5 independent developers' apps fixed. Conditional update level adjustments allow all independent developers to adjust update levels of apps that are updated in status-quo. Variable profits are calculated based on independent developers.

The second column of Table 11 reports the welfare effects of eliminating the identified self-preferencing in a partial-game simulation. Specifically, in the partial-game simulation, I allow all independent app developers to adjust their positive update frequency but holding update portfolios fixed. Similar to the previous results, comparing the second column to the third column shows that allowing partial update adjustments increases the gains in consumer surplus. Moreover, comparing the second column to the first column shows that allowing more independent apps to adjust update frequency modestly further increase the gain in consumer surplus, and decrease the gains in variable profits for independent developers. I interpret such a difference as the effect of competition between developers because there are more active developers in the partial-game simulation than the baseline simulation in the first column. In particular, the more intensive competition between developers increases consumer surplus gain and shrinks gains in variable profits.[77]

[76] Allen, Clark and Houde (2019) finds that 50 percent of consumer surplus gain from reduced search frictions in mortgage markets is associated with reduced search costs, while 22 percent gain is associated with inefficient matching.

[77] Notice that, without update portfolio adjustment, the change in variable profits is equal to change in profits. Therefore, average gain in profits with update portfolio adjustment ($0.396 million in Table 10) is lower than the average gain in profits without update portfolio adjustments reported in column (2) of Table 11. One reason is the increased fixed update costs with more apps to be updated.

Figure 6: Welfare Effects and Ranking Imperfection due to Apple's Self-preferencing



All together, the results from this section show that platform's self-preferencing decreases both consumer surplus and independent developer profits, which is exacerbated when accounting for quality adjustments. This conclusion is further confirmed by the heterogeneity of the welfare gains across markets. In particular, Figure 6 compares the welfare gains across markets with different extents of ranking imperfection induced by the identified self-preferencing.[78] As a correlation relationship, Figure 6 shows that the larger the ranking imperfection due to self-preferencing, the more the gains in consumer surplus and independent developer profits, again suggesting that platform's self-preferencing leads to welfare losses.

7 Conclusion

This paper identifies self-preferencing and quantifies its equilibrium welfare effects in the Apple App Store. I contribute to the literature by developing an empirical model of consumer search and update competition in the mobile application industry that is identifiable with rich aggregate data. To motivate the model, I exploit an unanticipated search algorithm change on the Apple App Store that drops some of Apple's apps from top search results. The existence and degree of self-preferencing are identified by comparing search rankings of

[78] Appendix C.6 explains how the ranking imperfections are calculated.

Apple's apps and independent apps, conditional on app quality, app price, ratings, and text relevance. I then use the estimates to examine the equilibrium effects of self-preferencing on update frequency, app quality, and welfare.

The results show that self-preferencing was present on the Apple App Store in the U.S. market from April to August 2019. It implies that the observed high search rankings of Apple's apps during the self-preferencing period were not purely due to the qualities of the apps; instead, Apple ownership gave them an advantage over independent apps in the search results.

I also find that eliminating the identified self-preferencing increases consumer surplus by $2.2 million and independent developer profits by $1.6 million per month. The resulting total surplus gain is equivalent to saving three-month consumers' installation payments or 2 percent of per-month total consumer payments, including payments for installation and in-app purchase and subscription. Additionally, as predicted by theory, I find sizable heterogeneity in both the direction and the magnitude of the effects on update frequency and app quality. For example, after eliminating the identified self-preferencing, an independent app's update frequency might decrease up to 20 percent and might as well increase up to 25 percent. The rich heterogeneity leads to small average effects on update frequency and app quality. Overall, these results imply that self-preferencing hurt both consumers and producers while negatively but modestly affecting quality provision on average. The approach can be applied to other settings where there might be self-preferencing and the effect of self-preferencing may be different.

Lastly, I argue that these estimated welfare gains from eliminating self-preferencing are likely to be lower bounds of real effects. Due to the lack of data on the number of users for pre-installed apps, the qualities of pre-installed apps are over-estimated. Therefore, I fix pre-installed apps' search rankings when quantifying the effects of self-preferencing. However, there might be further improved search efficiency and total welfare from eliminating self-preferencing when the search rankings of pre-installed apps are subject to changes. Moreover, the paper abstracts away from dynamic incentives of update and finds modest effects on update frequency and app quality in the short-run. However, the effects on updates are potentially larger in the long-run given that app quality is cumulative and updates positively affects the accumulation process. As for future research, this paper assumes that consumers know all the observable product characteristics before search. While I find some reduced-form evidence supporting the assumption to some extent, it is generally possible that consumers have limited information on some product characteristics before search. Relaxing this assumption could be studied in the future.

References

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*.

Anderson, Simon P, and Régis Renault. 2018. "Firm pricing with consumer search." In *Handbook of Game Theory and Industrial Organization, Volume II*. Edward Elgar Publishing.

Armstrong, Mark, and Jidong Zhou. 2011. "Paying for prominence." *The Economic Journal*, 121(556): F368–F395.

Arnosti, Nick, Ramesh Johari, and Yash Kanoria. 2014. "Managing congestion in dynamic matching markets." 451–451.

Bar-Isaac, Heski, Guillermo Caruana, and Vicente Cuñat. 2012. "Search, design, and market structure." *American Economic Review*, 102(2): 1140–60.

Baye, Michael R, John Morgan, and Patrick Scholten. 2004. "Price dispersion in the small and in the large: Evidence from an internet price comparison site." *The Journal of Industrial Economics*, 52(4): 463–496.

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

Bresnahan, Timothy, Joe Orsini, and Pai-Ling Yin. 2014. "Platform choice by mobile app developers." *NBER working paper*.

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

Choi, Hana, and Carl F Mela. 2019. "Monetizing online marketplaces." *Marketing Science*, 38(6): 948–972.

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.

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.

Dranove, David, Daniel Kessler, Mark McClellan, and Mark Satterthwaite. 2003. "Is more information better? The effects of "report cards" on health care providers." *Journal of political Economy*, 111(3): 555–588.

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*a*. "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. 2020*b*. "Merger, product variety and firm entry: The retail craft beer market in California."

Fershtman, Chaim, Arthur Fishman, and Jidong Zhou. 2018. "Search and categorization." *International Journal of Industrial Organization*, 57: 225–254.

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

Fradkin, Andrey. 2017. "Search, matching, and the role of digital marketplace design in enabling trade: Evidence from airbnb."

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

Ghose, Anindya, Panagiotis G Ipeirotis, and Beibei Li. 2014. "Examining the impact of ranking on consumer behavior and search engine revenue." *Management Science*, 60(7): 1632–1654.

Goeree, Michelle Sovinsky. 2008. "Limited information and advertising in the US personal computer industry." *Econometrica*, 76(5): 1017–1074.

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

Hausman, Jerry A, and Timothy F Bresnahan. 2008. *5. Valuation of New Goods under Perfect and Imperfect Competition.* 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. 2019. "Vertical contracts with endogenous product selection: An empirical analysis of vendor-allowance contracts." *Available at SSRN 3506265*.

Huang, Justin T. 2018. "Visibility Policy, Seller Incentives, and Pricing Dynamics in a Digital Goods Marketplace."

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

Kim, Jun B, Paulo Albuquerque, and Bart J Bronnenberg. 2010. "Online demand under limited consumer search." *Marketing science*, 29(6): 1001–1023.

Kim, Jun B, Paulo Albuquerque, and Bart J Bronnenberg. 2017. "The probit choice model under sequential search with an application to online retailing." *Management Science*, 63(11): 3911–3929.

Koulayev, Sergei. 2014. "Search for differentiated products: identification and estimation." *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.

Leyden, Benjamin T. 2021. "Platform Design and Innovation Incentives: Evidence from the Product Ratings System on Apple's App Store."

Li, Xing, Timothy Bresnahan, and Pai-Ling Yin. 2016. "Paying incumbents and customers to enter an industry: Buying downloads." *Available at SSRN 2834564*.

Mendelson, Haim, and Ken Moon. 2016. "Growth and customer loyalty: Evidence from the app economy."

Mendelson, Haim, and Ken Moon. 2018. "Modeling success and engagement for the app economy." 569–578.

Moraga-González, José Luis, Zsolt Sándor, and Matthijs R Wildenbeest. 2022. "Consumer Search and Prices in the Automobile Market."

Nosko, Chris. 2010. "Competition and quality choice in the cpu market."

Nosko, Chris, and Steven Tadelis. 2015. "The limits of reputation in platform markets: An empirical analysis and field experiment." National Bureau of Economic Research.

Orhun, A Yeşim, Sriram Venkataraman, and Pradeep K Chintagunta. 2016. "Impact of competition on product decisions: Movie choices of exhibitors." *Marketing Science*, 35(1): 73–92.

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.

Ratchford, Brian T. 2009. "Consumer search behavior and its effect on markets."

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

Santos, Babur De Los, Ali Hortaçsu, and Matthijs R Wildenbeest. 2017. "Search with learning for differentiated products: Evidence from e-commerce." *Journal of Business & Economic Statistics*, 35(4): 626–641.

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.

Tadelis, Steven, and Florian Zettelmeyer. 2015. "Information disclosure as a matching mechanism: Theory and evidence from a field experiment." *American Economic Review*, 105(2): 886–905.

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.

Wang, Peichun. 2017. "Innovation is the new competition: Product portfolio choices with product life cycles."

Wang, Quan, Beibei Li, and Param Vir Singh. 2018. "Copycats vs. original mobile apps: A machine learning copycat-detection method and empirical analysis." *Information Systems Research*, 29(2): 273–291.

Watson, Randal. 2009. "Product variety and competition in the retail market for eyeglasses." *The Journal of Industrial Economics*, 57(2): 217–251.

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

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

Yao, Song, and Carl F Mela. 2011. "A dynamic model of sponsored search advertising." *Marketing Science*, 30(3): 447–468.

Appendix A Details on Data

A.1 Sample Selection: Criteria and Process

Here I describe the details in sample selection. A category is in the sample if it has benchmark conversion rates data from AppTweak. The sample selection processes for apps and keywords are listed below.

App Selection Process:

1. Find apps that have ever-ranked top50 in the top-grossing charts in the selected categories during Apr.2019 to Sep.2019.
2. Conditional on selection into step 1, find the 50 mostly downloaded apps in 2019 in each category.
3. Repeat the above steps for top-paid charts, in order to ensure enough price variation for identifying price elasticity.[79]
4. Drop an app-month observations if i) that app has zero download in that month; or ii) that app has unobserved file size or ratings in that month.

Keyword Selection Process:

1. Find keywords that have ever entered the list of recently used keywords of a selected app, based on AppTweak's keyword suggestions. The "recency" on AppTweak is last 3 month, which corresponds to March 2020 to June 2020. The suggested keywords are keywords that either i) the app has ranked in top100 in the keyword recently; ii) the app's title, or subtitle, or descriptions contains the keyword.
2. For each app-keyword, track the app's historical search ranking in the keyword on each day during 2019. For each category, find the 60 keywords that have the most apps showing up in top500 search results in the category.

A.2 Variable Construction

Update Frequency is weighted number of updates(i.e.,released versions) in a month. An update is weighted by the length of release notes. I calculate the three quartiles of release note length for each category. If an update of an app in a category has a release note shorter than the first quartile of release note length for the category, the weight on this update is 0.25. If an update of an app in a category has a release note longer than the first quartile but shorter than the second quartile of release note length for the category, the weight on this update is 0.50. If an update of an app in a category has a release note longer than the second quartile but shorter than the third quartile of release note length for the category, the weight on this update is 0.75. If an update of an app in a category has a release note larger than the third quartile of release note length for the category, the weight on this update is 1. Then, the weighted number of updates of an app in a month is the sum of the weighted updates of the app in the month. **Update level** is $\log(1+n_{jt})$, where n_{jt} is the update frequency of app j in month t.

Search Ranking of an app in a category in a month is average positions of the app in the search results of the popular keywords in the category in the month, conditional on the position is in top50. If an app has

[79] Indeed, the sample does not include apps that only show up in top-free charts. For example, apps that monetize only through in-app advertising. Their objective functions are likely to be very different from the ones in my sample, who rely on the app store to earn profits.

never reached top50 in any keyword on any day in a category in a month, that app does not have a search ranking in that category in that month. Such situation is tracked with an indicator.

Number of Star Ratings of an app/category/month combination is the average number of ratings the given star level across days in the month. Star levels are 1 to 5 in integers.

Average Rating is the weighted average of the app/category/month's number of star ratings, with the weight equating the level of the star.

Title (subtitle) Match is the weighted average number of keywords that contains any word in the title (subtitle) of the app in a given month, with the weight equating the search volume of the keywords.

Ratio of Adopted Keywords for an app in an category on a day is the number of popular keywords that are adopted by the app on that day over the number of keywords selected for that category. Monthly level measurement is average of that ratio across days. An app's adoption of a keyword on a day is approximated by the appearance of that app in the keyword's top 500 search results on that day.

Appendix B Details on Estimation

B.1 Descriptive Evidence on Product Characteristics Known by Consumers Before Search

Here I provide descriptive test results on the assumption that consumers know all the observable app characteristics before search. I test the assumption with 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:

$$\begin{aligned} SearchVolume_{kgt} =& \beta_1^V AllPreinstall_{kgt} + AllPreinstall_{kgt} \times \{\bar{\boldsymbol{x}}_{kgt}^V \cdot \boldsymbol{\beta}_2^V + \beta_3^V \bar{p}_{kgt}\} \\ &+ \beta_4^V \overline{Apple}_{kgt} + \beta_5^V \overline{Preinstall}_{kgt} + \beta_5^V brand_k + \lambda_{gt}^V + \varepsilon_{kgt}^V \end{aligned} \tag{18}$$

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 E.16 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 E.17 presents the estimation results of Equation 18. 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 indicates the that assumption that consumers know the observable app characteristics before search is not too crazy.

B.2 Details in Estimation of Search Ranking Model

Here I list the control variables in the vector $\boldsymbol{z}^s_{jm}$ in Equation 8.

- $1\{j \textit{ is pre-installed}\} * 1\{t(m) = \tau\}$
- price($p_{jt(m)}$)
- paid($z^s_{1jt(m)}$)
- title-match($z^s_{2jt(m)}$)
- subtitle-match($z^s_{3jt(m)}$)
- last-period number of 5-star ratings($z^s_{4jt(m)}$)
- last-period number of 3-star ratings($z^s_{5jt(m)}$)
- last-period number of 1-star ratings($z^s_{6jt(m)}$)
- ever show up in top 500 search results in any selected keyword for the category ($z^s_{7jt(m)}$).
- brand ratio: the ratio of brand keywords among the keywords where the app has an observed ranking, interacted with z_{7ht}. ($z^s_{8jt(m)}$)

- $z_{jt} * 1\{g = h\}$ for each category $h \neq$ *Business* and $z_{jt} \in \{p_{jt(m)}, z^s_{1jt(m)}, z^s_{2jt(m)}, z^s_{3jt(m)}, z^s_{4jt(m)}, z^s_{5jt(m)}, z^s_{6jt(m)}, z^s_{7jt(m)}, z^s_{8jt(m)}\}$. Here, to save computation time, all game apps are indexed into one category: Game.
- $z_{jt} * 1\{t = \tau\}$ for each $t = 1, 2, \cdots, 23$ and $z_{jt} \in \{p_{jt(m)}, z^s_{1jt(m)}, z^s_{2jt(m)}, z^s_{3jt(m)}, z^s_{4jt(m)}, z^s_{5jt(m)}, z^s_{6jt(m)}, z^s_{7jt(m)}, z^s_{8jt(m)}\}$.

B.3 Truncated Set of Possible Orders

For markets with strictly more than 5 products, I use truncated set of possible orders $\mathscr{B}_{\boldsymbol{a}}$ to construct firms' beliefs on search rankings given update levels $\boldsymbol{a}$. Now I explain how $\mathscr{B}_{\boldsymbol{a}}$ is constructed. First recall that in Equation 9, the probability for an order $\boldsymbol{y}$, denoted by $\mathbb{P}[\boldsymbol{y}]$, is given by

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

$\mathscr{B}_{\boldsymbol{a}}$ are constructed based on the most-likely order, $\boldsymbol{y}^*$, predicted from the estimated rank-ordered logistic model, given a vector of updates $\boldsymbol{a}$. Notice that the numerator of $\mathbb{P}[\boldsymbol{y}]$ is unchanged with $\boldsymbol{y}$. Inspired by that observation, $\mathscr{B}_{\boldsymbol{a}}$ includes the following types of permutations of $\boldsymbol{y}^*$, found by enlarging the denominator of $\mathbb{P}[\boldsymbol{y}]$[80]. For all the examples below, I use 12345 to denote the ordering of 5 products in the most-likely ordering in a market with at least 5 products.[81]

1. first-layer enlargement-1: alter the positions of two nearby products. For example, 12345 $\to$ 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 $\to$ 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 $\to$ 13245 = 13245 $\to$ 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 $\to$ 21345 = 21345 $\to$ 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 $\to$ 21345 $\to$ 21435. $\forall j \in [1, J_m^{[1]} - 3]$.

Conduct the above permutations until position 30, whenever it applies. The resulting size of $\mathscr{B}_{\boldsymbol{a}}$ is

$$5 \times \min\{J, 30\} - 9 \leq 141$$

[80] Another type of permutation that also marginally enlarges the denominator is to alter the positions of two products with 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.

[81] In rank-ordered logistic regression model, the most-likely ordering is ordering of products by the mean score, which maximizes the $\mathbb{P}[\boldsymbol{y}]$.

B.4 First-Stage Results of Instruments in the Demand Model and Supply Model

Table E.18 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 E.19 reports the coefficients and standard errors in the first-stage estimation.

B.5 Constraints on Supply Model Estimation

The following constraints are applied on estimation of Equation 10 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 following constraints are applied on estimation of Equation 13 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 wrt downloads: $\psi_1 > 0$, $\psi_2 < 0$

B.6 Algorithm to Find Equilibrium Positive Update Levels

Given a draw of update cost slope shifters, $\boldsymbol{\omega}$, and an update portfolio $\boldsymbol{D}$, find equilibrium update levels (including zeros), $\boldsymbol{a}^*(\boldsymbol{\omega}; \boldsymbol{D})$ in the following steps:

1. Starting update levels: $\boldsymbol{a}_0 = (0, \boldsymbol{a}_{-j})$ if $D_j = 0$; $\boldsymbol{a}_0 = (a_j, \boldsymbol{a}_{-j})$ if $D_j = 1$.
   - 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 set of likely orders $\mathscr{B}_0$ following these steps:
   (a) predict probabilities to be ranked first, $\boldsymbol{p}_0$ based on the rank-ordered logit model, given $\boldsymbol{a}_0$.
   (b) use the probabilities, $\boldsymbol{p}_0$, to find the set of likely rankings, $\mathscr{B}_0 := \mathscr{B}_{\boldsymbol{a}_0}$.
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 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-change market outcomes if there were not the algorithm change. Because only categories with Apple's apps, i.e., the treatment group, will have different market outcomes, the simulation is conducted in categories with Apple's apps during the post-change periods from August to November in 2019. Because only bounds on fixed-costs of updates are identified, I follow the specification in 6.1 to draw fixed costs and report average effects across the draws. Moreover, because there is unobserved randomness in search rankings from the unobserved ranking score shocks, which feeds into randomness of installations, when comparing no-algorithm-change search rankings(installations) to with-algorithm-change search rankings(installations), I compare the expected values of search rankings(installations). Specifically, I simulate no-algorithm-change *expected* search rankings(installations) based on simulated no-algorithm-change update levels, and with-algorithm-change *expected* search rankings(installations) based on observed post-change update levels, where the expectation is over the truncated sets of possible search rankings. The structural estimates of average treatment effect on search rankings(installations) is difference between expected search rankings(installations) without the search algorithm change and expected search rankings(installations) with the search algorithm change.

Table E.20 reports the simulation results. It shows that the structural estimate of the average treatment effect on update levels is 1.7% while the difference-in-differences estimate is 2.1%. They are reasonably close. The smaller structural estimate also makes sense, because only the top5 firms are allowed to change update levels in the simulations, which potentially restricts the magnitudes of the effect. The structural estimate of the average treatment effect on search rankings is 1.5% while the difference-in-differences estimate is 3.6%. The discrepancy can be explained by the imperfect fitness of the search ranking model and the fact that the expectation is over the truncated set of possible search rankings instead of the complete set. The structural estimate of the average treatment effect on installation is 3.4% while the difference-in-differences estimate is 22.1%. They have the same signs but different magnitudes. The reasons for the discrepancy are two-fold: i) the discrepancy in effects on updates and search rankings feeds into the discrepancy in effects on downloads; ii) in the structural estimation, apart from update levels, all the other product characteristics and market features are fixed; while in the difference-in-differences, when comparing pre-change outcomes to post-change outcomes across treatment and control groups, there might be changes in other product characteristics (for example, file size and average ratings) and market features (for example, number of products). These changes lead to additional changes in installations that are not captured by the structural estimates. Overall, the structural estimates of ATE have the same signs as the difference-in-differences estimates, and the discrepancy have reasonable explanations.

C.2 Fix Pre-installed Apps' Search Rankings

Here I explain how I fix the search rankings of pre-installed apps and present its effects on market equilibrium. Following the idea that, search rankings are not perfectly known by developers in the stage of update choices, I fix the beliefs on search rankings of pre-installed apps (instead of observed search rankings of pre-installed apps). Specifically, following the estimated search ranking model and the construction of truncated set of possible search rankings as developers' beliefs on search rankings, I calculate the marginal distribution of search rankings of pre-installed apps in the status-quo. Then, I fix this marginal distribution through all simulations, with or without changes in preferential treatment on platform-owned products. Therefore, the welfare and product competition effects presented in Section 6 are immune from the effects of fixing pre-installed apps' search rankings. To complete the picture, I explain and present the effects of fixing the marginal distribution of pre-installed apps' search rankings on market outcomes below.

Depending on the extent of update adjustment that's allowed in the counterfactual simulation, the market outcomes with fixed marginal distributions of the search rankings of pre-installed apps may be different. Specifically, I calculate the market outcomes in three cases: i) no-game simulation: app developers are not allowed to change update levels; ii) partial-game simulation: app developers are only allowed to change positive update levels; iii) full-game simulation: only top5 app developers in each category are allowed to change update levels, including update or not and positive update levels. It is relatively easy to understand why the latter two cases see different market outcomes when fixing the marginal distribution of pre-installed apps' search rankings: equilibrium update levels could be different due to different beliefs of developers. When pre-installed apps' search rankings are not fixed, developers may believe that they can replace pre-installed apps in the positions that are occupied by pre-installed apps. When pre-installed apps' search rankings are fixed, they won't believe so.

In the first case (no-game simulation), the market outcome changes with fixing pre-installed apps' search rankings because of truncation of the set of possible search rankings. To illustrate the idea, let's consider an example where there are three products, 1-2-3, with the first product as the pre-installed app. As a benchmark, let's firstly consider the case of no truncation of the set of possible search rankings. The goal is to compare the probability to observe product-2 ranked above product-3 with and without fixing pre-installed apps' search rankings. The complete set of possible rankings of the three products is $\{123, 132, 213, 231, 312, 321\}$. Then, based on Equation 4.2 and 9, the probability to observe product-2 ranked above product-3 is given by $\mathbb{P}[23] = \mathbb{P}[123] + \mathbb{P}[213] + \mathbb{P}[231] = \exp(\overline{score}_2)/[\exp(\overline{score}_2) + \exp(\overline{score}_3)]$. After fixing the rankings of the pre-installed product-1, there are only two products that will be ranked in the search ranking model: product-2 and product-3. Therefore, the probability to observe product-2 ranked above product-3 after fixing the pre-installed product's ranking is also $\exp(\overline{score}_2)/[\exp(\overline{score}_2) + \exp(\overline{score}_3)]$. Therefore, without truncation of the set of possible search rankings, market outcome may not change with fixing pre-installed apps' search rankings.

Now, let's consider the case of truncation of the set of possible search rankings. As described in Appendix B.3, one type of permutation included in the truncated set is alternating the positions of two near-by products in the most-likely ordering of products. For simplicity, we only consider this type of permuta-

tion here.[82] Then, let us suppose the most-likely ordering of products is $\{123\}$, which gives the following truncation set of orderings: $\{123, 132, 213\}$. Notice that the order $\{231\}$ is not captured by the truncation set. Then, the probability of observing product-2 ranked above product-3 within the truncation set is $\tilde{\mathbb{P}}[23] = \exp(\overline{score}_2)/[\exp(\overline{score}_2) + \exp(\overline{score}_3)(\exp(\overline{score}_1) + \exp(\overline{score}_3))]$. When fixing the pre-installed product-1's position, the truncation set becomes $\{23, 32\}$, which is also the complete set for the two products. Then, the probability of observing product-2 ranked about product-3 within the truncation set is $\exp(\overline{score}_2)/[\exp(\overline{score}_2) + \exp(\overline{score}_3)]$, which is different from the probability without fixing pre-installed apps' search rankings. Therefore, with truncation of the set of possible search rankings, market outcomes will change with fixing pre-installed apps' search rankings.

To show the effects of fixing pre-installed apps' search rankings on market outcomes, I compare the simulated market outcomes with fixed marginal distribution of pre-installed apps' search rankings(denoted by $y_{jgt}^{[fix]}$) to market outcomes without fixed marginal distribution of pre-installed apps' search rankings(denoted by $y_{jgt}^{[0]}$). The difference captures the effect of fixing marginal distribution of pre-installed apps' search rankings. To measure the difference, I use relative L1-Norm in percentage : $100 \times [\sum_{jgt}(|y_{jgt}^{[fix]} - y_{jgt}^{[0]}|)]/[\sum_{jgt}|y_{jgt}^{[0]}|]\%$.

Table E.21 reports the effect of fixing marginal distribution of pre-installed apps' search rankings. All of the changes are smaller than 0.1%. Although the changes are small, they are taken into account during counterfactual simulations, i.e., all reported effects in Section 6 are comparing the market outcomes without preferential treatment effects on platform-owned products in search algorithm and market outcomes with preferential treatment effects on platform-owned products in search algorithm, while the marginal distribution of pre-installed apps' search rankings are fixed in both cases.

C.3 Calculation of Consumer Surplus

This appendix gives details on computing the expected consumer welfare in Equation 17. To compute the consumer welfare, I take 10,000 draws of the idiosyncratic unobserved match values and consumer search costs. The match values are drawn from the Type-I Extreme Value distribution. The search costs are drawn from the app/category/month-specific distribution given in Equation 5.[83] On top of these random shocks, there are i) 4 sparse-grid nodes for the one-dimension random coefficient on update levels generated from the sparse-integration method in Heiss and Winschel (2008) with accuracy level of 4; and ii) the truncated set of possible search rankings. Given 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 (2022), i.e., $r_{ij} = \delta_{ij} +$

[82] Considering all five types of permutation listed in Appendix B.3 will cause the truncation set equal to the complete set for just three products.

[83] 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 the computational time.

$H_0^{-1}(c_{ij})$, where δ_{ij} is the known utility before search, equating $u_{ij\cdot} - \varepsilon_{ij\cdot}$ in Equation 2; c_{ij} is the consumer-app specific search costs; and the function $H^0(\cdot)$ is given in Equation 5.

2. Check 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. Check 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 4 gives the consumer welfare in dollars.

Then I take average of simulated consumer welfare and multiply it with market size to get market-level consumer surplus. Notice that, by simulating the optimal sequential search problem, I have computational market shares derived from the discrete choices of simulated consumers. Therefore, there might be 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 E.24 presents the computational error during computation of consumer surplus. It shows that all computational errors are smaller than 0.4%.

C.4 Explain Heterogeneous Effects on Update Frequency

The understand why some independent apps increase their update frequency while others decrease, I summarize the patterns between the percentage change of update frequency and some explanatory variables with the following linear regression. It is only a correlation results, instead of causal relationship. But it is helpful for summary.

$$\begin{array}{lllll} \%\Delta update_{jm} = & -0.65 + & 0.66\ \%\Delta Q_{jm} - & 0.27\ nop_{jm} - & 0.17\ June_m + \rho_{jm} \\ & (0.96) & (0.36) & (0.14) & (1.19) \end{array}$$

where $\%\Delta update_{jm}$ is the percentage change of update frequency of app j in market m after eliminating the identified self-preferencing, $\%\Delta Q_{jm}$ is the percentage change of downloads of app j in market m after the eliminating without update adjustment, nop_{jm} is the number of apps owned by app j's developer in market m, and $June_m$ indicates whether the market m is in June 2019 (the month with stronger identified self-preferencing).

The regression results indicate that reduction of update frequency after the elimination is related to reduction of downloads without update adjustment, stronger cannibalization concern (captured by larger nop_{jm}), and aggressive reduction in self-preferencing (captured by $June_m$). In particular, looking into the simulation data, I find that apps whose downloads decrease after the elimination without update adjustment are typically apps who are always above or below Apple's apps with or without self-preferencing. Then, their reduction in downloads are due to business stealing from boosted-up independent apps who replace

Apple's apps. It implies that the boosted-up independent apps are stronger competitors and more attractive for consumers than Apple's apps.

C.5 Counterfactual Simulation Results without Update Portfolio Adjustment

Here I report the simulation results with changes in positive update level but without update portfolio adjustments. I call such simulation as partial-game simulation. Because all independent apps with positive update levels are allowed to make a change, there more simulation data points than those in the case of full update adjustment by top5 developers.[84] It motivates me to use these results to discuss patterns with respect to heterogeneous effects of self-preferencing on update levels. I firstly show the quantified effects of self-preferencing on update levels, search rankings, installations and welfare. Then I discuss the heterogeneity in the effects on update levels.

Table E.22 presents the effect of self-preferencing on positive update levels from the partial-game simulation. The effects have the same direction as the main results reported in Table 9, with smaller magnitudes. While we don't see much heterogeneous effects at market-level, the heterogeneity at product-level remains strong. The standard deviation of the percentage change of product-level positive update levels is 4.2 times of the mean. The range of the percentage change of product-level positive update levels is from -0.2% to 4.4%.

Table E.23 presents the effects of self-preferencing on search rankings, installations and welfare in the partial-game simulation. The effects are quite similar to the main results reported in Table 10. The smaller increase in producer surplus compared to that in the main result can be explained by restricted adjustment: the developers are not allowed to adjust update portfolios in the partial game.

C.6 Ranking Imperfection due to self-preferencing

Here I explain how the ranking imperfection due to self-preferencing is calculated. Ideally, the rankings should be based on average scores equal to consumer values on the apps (relative mean-utility). However, this is not true neither with or without the self-preferencing. Therefore, I firstly calculate i) within-market rankings that are based on app values, ii) the within-market rankings that are based on the mean scores with self-preferencing, and iii) the within-market rankings that are based on the mean scores without self-preferencing. Then I calculate the gap between the first and the second within-market rankings, and the gap between the first and the third within-market rankings. The gaps are the ranking imperfections across products. To aggregate the product-level ranking imperfections to market-level, I firstly construct a weight. Because higher positions are more important for welfare, I use $1/\log(1+ranking)$ as the weight on the gaps across products. Then, I use the weighted average gaps in a market to measure the ranking imperfection in the market. Lastly, I calculate the percentage change in the market-level ranking imperfection after eliminating self-preferencing to measure the ranking imperfection due to self-preferencing.

[84] There are 337 app/category/month combinations whose update levels are subject to changes in counterfactual simulations with endogenous positive update levels. That number is 79 app/category/month combinations in counterfactual simulations with endogenous update levels of top5 developers in each category.

Appendix D Ambiguous Supply-side Effect of self-preferencing

Here I explain the ambiguity of supply-side responses to change in self-preferencings: it is theoretically ambiguous that whether developers will update more or less with less self-preferencing of Apple's apps. The ambiguity roots in two offsetting economic forces: i) strategic complementarity of the virtual mean relative utility (which roots in concavity of revenue curves); ii) larger marginal quantity with respect to update with higher search rankings. I firstly illustrate the intuitions in a two-product market, then discuss the case in more complex markets.

Imagine a market with only two apps: i) app-1, an Apple's app; ii) app-2, an independent app. Then, there are two possible orders of the apps: 12 or 21. Then, the marginal benefit from update for app-2 is given by

$$MB_2 = \sum_{\boldsymbol{y}\in\{12,21\}} \left(\frac{\partial \pi_2^I(\boldsymbol{y})}{\partial a_2} \tilde{\mathbb{P}}[\boldsymbol{y}] + \pi_2^I(\boldsymbol{y}) \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}]}{\partial a_2} \right)$$

A decrease in self-preferencing of Apple's apps is a decrease in θ_{Apple}, which only changes the function $\tilde{\mathbb{P}}[\boldsymbol{y}]$, but not the profit function output π_2^I. In particular, denote the new probability function with lower θ_{Apple} as $\tilde{\mathbb{P}}'[\boldsymbol{y}]$. Then, the post-change marginal benefit from updates for app-2 is given by

$$MB_2' = \sum_{\boldsymbol{y}\in\{12,21\}} \left(\frac{\partial \pi_2^I(\boldsymbol{y})}{\partial a_2} \tilde{\mathbb{P}}'[\boldsymbol{y}] + \pi_2^I(\boldsymbol{y}) \frac{\partial \tilde{\mathbb{P}}'[\boldsymbol{y}]}{\partial a_2} \right)$$

Denote the difference between the post-change values and pre-change values with $\Delta x := x' - x$. Notice that $\tilde{\mathbb{P}}[\mathbf{12}] + \tilde{\mathbb{P}}[\mathbf{21}] \equiv 1$. Therefore, $\Delta\tilde{\mathbb{P}}[\mathbf{12}] + \Delta\tilde{\mathbb{P}}[\mathbf{21}] = 0$, and $\frac{\partial \tilde{\mathbb{P}}[\mathbf{12}]}{\partial a_2} = -\frac{\partial \tilde{\mathbb{P}}[\mathbf{21}]}{\partial a_2}$. It is also useful to notice that i) a decrease in θ_{Apple} will cause $\Delta\tilde{\mathbb{P}}[\mathbf{21}] > 0$; ii) $\pi_2^I(\mathbf{21}) > \pi_2^I(\mathbf{12})$ since only rankings are different, and it's identified and estimated that $\lambda_2 > 0$ and thus $Q_2(\mathbf{21}) > Q_2(\mathbf{12})$, and marginal profits from quantities are estimated as positive[85]. I use these observations to simplify ΔMB_2 and get the following expression:

$$\Delta MB_2 = \underbrace{\Delta\tilde{\mathbb{P}}[\mathbf{21}]}_{>0} \underbrace{\left(\frac{\partial \pi_2^I(\mathbf{21})}{\partial a_2} - \frac{\partial \pi_2^I(\mathbf{12})}{\partial a_2} \right)}_{I \text{ (unclear sign)}} + \underbrace{\Delta \frac{\partial \tilde{\mathbb{P}}[\mathbf{21}]}{\partial a_2}}_{II \text{ (unclear sign)}} \underbrace{\left(\pi_2^I(\mathbf{21}) - \pi_2^I(\mathbf{12}) \right)}_{>0} \tag{19}$$

I analyze term I first, then term II. The term I is given by:

$$\begin{aligned} \frac{\partial \pi_2^I(\mathbf{21})}{\partial a_2} - \frac{\partial \pi_2^I(\mathbf{12})}{\partial a_2} = & \underbrace{\left[0.7 \underbrace{\left(\tau_1 + 2\tau_2 Q_2(\mathbf{21}) \right)}_{R_2^{(q)}} + \underbrace{\phi_1 + 2\phi_2 Q_2(\mathbf{21})}_{F_2^{(q)}} \right] \underbrace{\frac{\partial Q_2(\mathbf{21})}{\partial a_2}}_{\text{probably large}}}_{\text{small}} \\ & - \underbrace{\left[0.7 \underbrace{\left(\tau_1 + 2\tau_2 Q_2(\mathbf{12}) \right)}_{R_2^{(q)\prime}} + \underbrace{\phi_1 + 2\phi_2 Q_2(\mathbf{12})}_{F_2^{(q)\prime}} \right] \underbrace{\frac{\partial Q_2(\mathbf{12})}{\partial a_2}}_{\text{probably small}}}_{\text{large}} \end{aligned} \tag{20}$$

[85] Marginal "revenue" from quantities is given by $0.7p_j + 0.7(\tau_1 + 2\tau_2 Q_j) + (\phi_1 + 2\phi_2 Q_j)$ for each product j.It's estimated that $\tau_1 > 0$, $\tau_2 < 0$, $\phi_1 > 0$, $\phi_2 < 0$. Therefore, the "revenue" curve is increasing and concave with respect to quantities.

Because i)$Q_2(\mathbf{21}) > Q_2(\mathbf{12})$, ii) it's estimated that $\tau_2 < 0$ and $\phi_2 < 0$, I have lower marginal "revenues" wrt quantities from the order $\mathbf{21}$ than those from the order $\mathbf{12}$. This is just saying that when ranked higher, the app-2's downloads increase. Given the concave revenue curve, this decreases the marginal benefit from updates. An alternative explanation is strategic complementarity of the virtual mean relative utility (product value net of search costs): app-2's competitor, app-1, is ranked lower, and thus has a lower virtual mean relative utility; which decreases app-2's marginal benefits from increasing the virtual mean relative utility because their virtual mean relative utilities are strategic complements. The strategic complementarity roots in the increasing and concave revenue curve wrt quantities.

Now I compare $\frac{\partial Q_2(\mathbf{21})}{\partial a_2}$ and $\frac{\partial Q_2(\mathbf{12})}{\partial a_2}$, and argue that the former is likely to be larger in this paper's empirical context. Their difference is given by:

$$\frac{\partial Q_2(\mathbf{21})}{\partial a_2} - \frac{\partial Q_2(\mathbf{12})}{\partial a_2} = \underbrace{M \int \underbrace{[(s_{i2}(\mathbf{21}) - s_{i2}^2(\mathbf{21})) - (s_{i2}(\mathbf{12}) - s_{i2}^2(\mathbf{12}))]}_{>0 \text{ if } s_{i2}(\mathbf{21}) \leq 0.5} (\gamma + \sigma v_i) d\Phi(v_i)}_{\text{unclear sign}}$$

The function $x(1-x)$ is increasing and concave, reaching maximum when $x = 0.5$. Moreover, $s_{i2}(\mathbf{21}) > s_{i2}(\mathbf{12})$ since $\lambda < 0$. Therefore, when $s_{i2}(\mathbf{21}) \leq 0.5$, the bracket-term is positive. And this inequality is likely to happen in our empirical context because the largest total market share in the data is smaller than 0.25. On the other hand, v_i may be negative. It turns out that the bracket-term could be increasing or decreasing with v_i[86]. Therefore, ultimately, the difference has an unclear sign. Intuitively, this difference captures how the marginal downloads with respect to updates change with discoverability. I conclude that, while the model is flexible enough to allow both more and less marginal downloads with more discoverability, our data is likely to fit the case of discoverability increasing marginal downloads with respect to updates.

Next I analyze term *II* and argue that it has an unclear sign. Note that $\partial \tilde{\mathbb{P}}[\mathbf{21}] = p_2$, where p_2 is the probability that app-2 is ranked first. I write out term *II* as the following

$$\Delta \frac{\partial \tilde{\mathbb{P}}[\mathbf{21}]}{\partial a_2} = \underbrace{[(p_2' - p_2^2) - (p_2 - p_2^2)]}_{\text{unclear sign}} \underbrace{\gamma}_{>0} \underbrace{\theta_{quality}}_{>0}$$

I only know that $p_2' > p_2$. In fact, $p_2 \in (0,1)$ with $p_2 + p_1 = 1$ while $s_2 + s_1 < 1$ since there is out-side option for consumers to choose but all products are ranked. Therefore, I have no clue for the likely sign of the bracket-term[87]. And term *II* has an unclear sign.

As a summary of the illustrative example. Equation 19 writes out the effect of the self-preferencing on

[86] I derived that $\partial s_{i2} / \partial v_i = (s_{i2} - s_{i2}^2)\sigma a_2 > 0$. I further derived that the partial derivative of the bracket-term with respect to v_i is given by $(s_{i2}(\mathbf{21})(1 - s_{i2}(\mathbf{21}))(1 - 2s_{i2}(\mathbf{21})) - s_{i2}(\mathbf{21})(1 - s_{i2}(\mathbf{21}))(1 - 2s_{i2}(\mathbf{21})))\sigma a_2$. The value of the partial derivative depends on the behavior of the function $x(1-x)(1-2x)$. This function is symmetric through $(0.5, 0)$, with the points left to $(0.5, 0)$ being positive and the points right to $(0.5, 0)$ being negative. And it is reserve U-shape before reaching (0.5,0) and U-shape after the point (0.5,0). This behavior of the function implies that even when $s_{i2}(\mathbf{21} \leq 0.5$, I could have smaller (less positive) bracket-term with larger v_i. However, to have positive difference for sure when $s_{i2}(\mathbf{21} \leq 0.5$, we need the bracket-term to be larger with larger v_i so as to surely offset negative values of v_i with positive v_is.

[87] To fit the idea, the following table is useful:

bracket-term	$p_2 < 0.5$	$p_2 > 0.5$
$p_2' < 0.5$	+	n/a
$p_2' > 0.5$	unclear	-

developers' incentives to update apps. Equation 20 writes out the most important term in Equation 19, which illustrates the two offsetting economic forces. On the one hand, concave revenue curve generates strategic complementarity of product values net of search costs. This decreases independent developers' incentive to update apps with less self-preferencing of Apple's apps: my competitors perform worse, why don't I take a good rest. On the other hand, discoverability is likely to increase the marginal downloads with respect to updates. This increases independent developers' incentive to update apps with less self-preferencing of Apple's apps: my efforts are more visible now, why don't I take the chance.

Now I discuss the case in more complex markets. A general version of Equation 19 is given by

$$\Delta MB_j = \sum_{\boldsymbol{y}' \in \mathscr{B}'_{\boldsymbol{a}}} \left(\frac{\partial \pi_f^I(\boldsymbol{y}')}{\partial a_j} \tilde{\mathbb{P}}'[\boldsymbol{y}'|\boldsymbol{a}] + \pi_f^I(\boldsymbol{y}') \frac{\partial \tilde{\mathbb{P}}'[\boldsymbol{y}'|\boldsymbol{a}]}{\partial a_j} \right) - \sum_{\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}} \left(\frac{\partial \pi_f^I(\boldsymbol{y})}{\partial a_j} \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}] + \pi_f^I(\boldsymbol{y}) \frac{\partial \tilde{\mathbb{P}}[\boldsymbol{y}|\boldsymbol{a}]}{\partial a_j} \right)$$

The difference between the above equation and Equation 19 are mainly two-fold. First, the permutation set $\mathscr{B}$ is a truncated set of permutations, therefore, $\boldsymbol{y} \neq \boldsymbol{y}'$, or, $\mathscr{B}'_{\boldsymbol{a}} \neq \mathscr{B}_{\boldsymbol{a}}$.In particular, independent apps are ranked higher in every $\boldsymbol{y}' \in \mathscr{B}'_{\boldsymbol{a}}$ compared to $\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}$, because the most likely ordering has independent apps ranked higher, and it determines the truncation set. However, in the illustrative example, the permutation set is not changed with θ_{Apple}. This difference add complication into the analysis of ΔMB_j because I cannot take out a common term $\Delta\tilde{\mathbb{P}}[\boldsymbol{y}]$ as what I did in Equation 19. However, because independent apps are ranked higher in every $\boldsymbol{y}' \in \mathscr{B}'_{\boldsymbol{a}}$ compared to $\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}$, the term I in the Equation 19 still exists in the general version. Therefore, it's possible that the intuition still applies. The next job is to check whether we have similar predictions on the sign of term I in the general version.

Analysing the term I in the general version introduces the second difference between Equation 19 and the general case: firms are allowed to be multiple-product firms in the general version, while the illustrative example just has two single-product firms. The corresponding general version of Equation 20 is given by:

$$\begin{aligned} \frac{\partial \pi_f^I(\boldsymbol{y}')}{\partial a_j} - \frac{\partial \pi_f^I(\boldsymbol{y})}{\partial a_j} = & \sum_{l \in \mathscr{J}_f} \underbrace{[0.7 \underbrace{(\tau_1 + 2\tau_2 Q_l(\boldsymbol{y}'))}_{R_l^{(q)}} + \underbrace{\phi_1 + 2\phi_2 Q_l(\boldsymbol{y}')}_{F_l^{(q)}}]}_{\text{unclear relative size}} \underbrace{\frac{\partial Q_l(\boldsymbol{y}')}{\partial a_j}}_{\text{unclear}} \\ & - \sum_{l \in \mathscr{J}_f} \underbrace{[0.7 \underbrace{(\tau_1 + 2\tau_2 Q_l(\boldsymbol{y}))}_{R_l^{(q)'}} + \underbrace{\phi_1 + 2\phi_2 Q_l(\boldsymbol{y})}_{F_l^{(q)'}}]}_{\text{unclear relative size}} \underbrace{\frac{\partial Q_l(\boldsymbol{y})}{\partial a_j}}_{\text{unclear}} \end{aligned}$$

where $\boldsymbol{y}' \in \mathscr{B}'_{\boldsymbol{a}}$ and $\boldsymbol{y} \in \mathscr{B}_{\boldsymbol{a}}$. Now the bracket terms do not have unclear relative sizes, even though independent apps' rankings are ranked higher if not unchanged. This is because apps are competitors, one of the apps that are ranked higher may become so much more attractive to consumers such that it steals business from all the other apps. Similar logic applies to the analysis of relative magnitudes between $\partial Q_l(\boldsymbol{y}')/\partial a_j$ and $\partial Q_l(\boldsymbol{y})/\partial a_j$: it's unclear whether $s_{il}(\boldsymbol{y}')$ will be larger or smaller than $s_{il}(\boldsymbol{y})$, even though the ranking of l is in $\boldsymbol{y}'$ is no lower than that in $\boldsymbol{y}$. Therefore, there are more ambiguity in the general case, and I focus

on the illustrative example for illustrating the fundamental intuitions.

Overall, in the general case where multiple products exist in one market, who replace the platform-owned product in the higher positions matters for the competition outcome. If the boosted-up independent products are more competitive than the platform-owned products, then the strengthened product competition will provide additional incentive to work against the strategic complementarity and encourage quality provision. Otherwise, the product competition is further weakened, providing additional incentive to exacerbate the strategic complementarity and discourage quality provision. There are three possible scenarios: i) the platform-owned products have higher before-change net product values than all of the boosted-up independent products; ii) the platform-owned products have higher before-change net product values than some of the boosted-up independent products; iii) the platform-owned products have lower before-change net product values than all of the boosted-up independent products.

In the scenario i), the product competition is weakened, and strategic complementarity discourages quality provisions of all independent apps, boosted-up or not. In the scenario ii), the product competition is firstly strengthened for the independent apps that were ranked above platform-owned products. However, for the boosted-up independent apps that have lower before-change net product values than the platform-owned products, it's unclear whether their product competition might be strengthened or weakened. On the one hand, the average competitiveness of products ranked before them is higher; on the other hand, the number of products ranked before them is smaller. Lastly, for the boosted-up independent apps that have higher before-change net product values than the platform-owned products, their product competition is weakened due to less products ranked before them. On top of the potentially heterogeneous changes in product competition, there is smaller marginal revenue from downloads due to increased downloads from higher rankings. Therefore, in this scenario, it is unclear whether the combination of heterogeneous changes in product competition and concavity in marginal revenue curves will lead to more or less quality provision. In the scenario iii), the product competition is strengthened for independent products that were ranked before platform-owned products; and weakened for the boosted-up independent products. This again leads to unclear overall changes of quality provision. While the scenario i) is unlikely to happen following the idea of self-preferencing, the other two scenarios both lead to ambiguous changes in the incentives for quality provision.

On top of the changes in product competition and smaller marginal revenues from quantities, there is higher marginal quantity from quality that encourages quality provision. The three forces lead to ambiguous supply-side effects of self-preferencing in the general case.

Appendix E Tables

Table E.12: App Categories in the Sample

	Non-Game App Category	Game App Category
(1)	Book	Games-Action
(2)	Business	Games-Adventure
(3)	Education	Games-Arcade
(4)	Entertainment	Games-Board
(5)	Finance	Games-Card
(6)	Food & Drink	Games-Casino
(7)	Health & Fitness	Games-Family
(8)	Lifestyle	Games-Music
(9)	Medical	Games-Puzzle
(10)	Music	Games-Racing
(11)	Navigation	Games-Role Playing
(12)	News	Games-Simulation
(13)	Newsstand	Games-Sports
(14)	Photo & Video	Games-Strategy
(15)	Productivity	Games-Trivia
(16)	Reference	Games-Word
(17)	Shopping	
(18)	Social Networking	
(19)	Sports	
(20)	Travel	
(21)	Utilities	
(22)	Weather	

Table E.13: Additional Summary Statistics and Data Sources

Variable	Mean	SD	Min	Max	Source
Panel A. App Data					
#5-star ratings(million)	0.048	0.240	0	10.11	AppTweak
#4-star ratings(million)	0.006	0.028	0	1.03	AppTweak
#3-star ratings(million)	0.002	0.009	0	0.34	AppTweak
#2-star ratings(million)	0.001	0.003	0	0.13	AppTweak
#1-star ratings(million)	0.002	0.013	0	0.71	AppTweak
Market Size(million)	107.800	4.691	98.55	117.70	Comscore
Pre-installs[c] :					
#Pre-installs	0.643	1.312	0	7	public info[b,c]
min.Ever Top50	2.756	6.985	0	50	SensorTower[c]
{min.Search Ranking}×Ever Top50	0.300	0.458	0	1	SensorTower[c]
Obs(app/category/month)		56,570			
Panel B. Benchmark Conversion Rates Data					
Type(free/paid apps)	0.35	0.48	0	1	AppTweak
Benchmark Conversion Rate	0.052	0.059	0.001	0.45	AppTweak
Obs(type/category/month)		1,337			
Number of Categories		38			
Number of Months		23			
Panel C. Search Volume Data					
Brand Keyword?	0.36	0.48	0	1	AppTweak
Search Volume	45.03	15.15	5	100	AppTweak
Obs(keyword/day)		1,246,000			
Number of Keywords		1,780			

[a]Constructed based on data on title and subtitle history from AppTweak.
[b]Some pre-installed apps do not have category information, and thus are not counted. For data availability of each pre-installed app, please see Table E.14.
[c]Each app/category/month observation is matched with information about pre-installed apps in the category/month.

Table E.14: 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 E.15: 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 E.16: 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 E.17: Estimation Results: Known App Characteristics before Search

	Top50 Search Results		Top1 Search Results	
	Search Volume			
Variable	(1)	(2)	(1)	(2)
All Pre-install?	1.176	-1.997***	5.861**	-3.135***
	(1.974)	(0.741)	(2.292)	(1.037)
Update Level	1.391***	0.464***	1.131***	0.161**
	(0.275)	(0.0758)	(0.217)	(0.0659)
Average Rating	0.328	0.0593	0.241	0.296*
	(0.333)	(0.114)	(0.286)	(0.169)
Age(month)	0.0120***	-0.0138***	0.0179***	-0.0208***
	(0.00438)	(0.00212)	(0.00370)	(0.00442)
File Size(GB)	-0.949***	0.215	-0.344	0.0588
	(0.367)	(0.165)	(0.262)	(0.199)
#Screenshots	0.785***	0.0492**	0.395***	0.00915
	(0.0692)	(0.0230)	(0.0527)	(0.0301)
Description Length(1,000 characters)	-1.112***	-0.0921*	-0.496***	0.135
	(0.143)	(0.0522)	(0.105)	(0.0865)
Offer In-app-purchase	3.141***	-1.748***	3.659***	-3.002***
	(0.625)	(0.347)	(0.514)	(0.658)
Paid Installation	-12.61***	-0.711**	-8.762***	-2.683***
	(0.572)	(0.344)	(0.456)	(0.845)
Price	-0.209***	0.000665	-0.0887*	0.154***
	(0.0774)	(0.0369)	(0.0457)	(0.0430)
Apple	10.99***	-1.611***	12.31***	-2.347***
	(1.521)	(0.508)	(1.081)	(0.838)
Pre-install	-2.582	1.485**	-10.23***	3.212***
	(1.728)	(0.595)	(2.060)	(0.979)
Brand-name Keywords?	5.327***		5.737***	
	(0.127)		(0.181)	
Constant	40.32***	50.05***	40.36***	51.22***
	(1.666)	(0.671)	(1.380)	(0.911)
Observations	41,974	41,964	18,787	18,708
R-squared	0.262	0.963	0.347	0.965
Category-Month FE	YES		YES	
Keyword-Category FE		YES		YES
Month FE		YES		YES
Mean level	48.29		48.90	

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

Table E.18: First-stage IV Regression Results: F statistics

	Demand					Supply
Variables	(1) Price	(2) Average Rating	(3) Update Level	(4) Search Ranking ×Ever Top50	(5) Ever Top50	(6) Update Level
F^a	545.5	86.53	372.1	1759	2702	1645
Excluded F^b	43.73	58.46	113.7	66.52	239.8	40.47

[a] On the demand side, the reported F is F(93,52865). On the supply side, the reported F is F(73,25252)

[b] On the demand side, the reported excluded F is F(57,52865). On the supply side, the reported excluded F is F(50,25252).

Table E.19: First-stage IV Regression Results

	Demand					Supply
	(1)	(2)	(3)	(4)	(5)	(6)
Variables	Price	Average Rating	Update Level	Search Ranking ×Ever Top50	Ever Top50	Update Level
Price		0.00280***	0.00159***	0.0358***	-0.000258	0.00176**
		(0.000683)	(0.000492)	(0.0102)	(0.000315)	(0.000899)
Average Rating	0.0990***		0.0514***	-0.313***	0.0473***	
	(0.0261)		(0.00292)	(0.0670)	(0.00203)	
Update Level	0.0908***	0.0830***		-0.756***	0.0334***	
	(0.0302)	(0.00465)		(0.102)	(0.00270)	
Search Ranking × Ever Top50	0.00399***	-0.000987***	-0.00147***		0.0200***	
	(0.00109)	(0.000210)	(0.000198)		(8.80e-05)	
Ever Top50	-0.0373	0.193***	0.0842***	25.90***		
	(0.0464)	(0.00819)	(0.00684)	(0.119)		
Pre-installs:Search Ranking × Ever Top50	-0.00234	0.00155*	-5.78e-05	-0.00497	9.25e-05	
	(0.00464)	(0.000928)	(0.000720)	(0.0152)	(0.000419)	
Pre-installs: Ever Top50	0.0633	-0.0499*	0.0176	-0.167	0.0130	
	(0.138)	(0.0269)	(0.0220)	(0.455)	(0.0127)	
Apple	-0.505***	-0.182***	-0.0731**	-5.336***	0.232***	
	(0.137)	(0.0611)	(0.0286)	(0.918)	(0.0264)	
Paid Installation?	4.285***	-0.0290***	-0.228***	-0.0655	-0.188***	-0.0739***
	(0.0542)	(0.00795)	(0.00579)	(0.136)	(0.00387)	(0.00994)
Offer In-app-purchase?	-0.572***	0.0267***	0.0902***	0.367***	0.00522	0.103***
	(0.0557)	(0.00758)	(0.00450)	(0.110)	(0.00344)	(0.0110)
$\log(Age)$[a](month)	0.133***	0.00574	-0.0185***	-0.821***	0.0263***	-9.56e-05
	(0.0197)	(0.00463)	(0.00345)	(0.0745)	(0.00208)	(0.000126)
$\log(FileSize)$(MB)	0.487***	0.0135***	0.0361***	-0.347***	0.0165***	0.0503***
	(0.0175)	(0.00232)	(0.00152)	(0.0358)	(0.00106)	(0.00294)
#Screenshots	-0.0620***	0.0185***	0.0245***	0.0576***	-0.00255***	
	(0.0119)	(0.00122)	(0.00104)	(0.0214)	(0.000587)	
$\log(1+DescriptionLength)$(1,000 characters)	0.290***	0.0308***	0.0112***	-0.0865	0.0175***	
	(0.0266)	(0.00463)	(0.00311)	(0.0574)	(0.00160)	
#Pre-installs	-0.0639***	-0.00305	-0.00528	-0.680***	0.0182***	
	(0.0213)	(0.00425)	(0.00327)	(0.0685)	(0.00194)	
Game Apps?	-1.721***	-0.0182	0.0304	3.448***	-0.168***	
	(0.146)	(0.0343)	(0.0257)	(0.540)	(0.0148)	
PromotionTextLength	0.000355*	-2.00e-05	-6.22e-05*	0.000556	-5.25e-05**	
	(0.000195)	(3.89e-05)	(3.31e-05)	(0.000748)	(2.08e-05)	
#Icon Changes	0.0270	0.00475	0.292***	0.803***	-0.00684**	
	(0.0198)	(0.00481)	(0.00570)	(0.131)	(0.00336)	
Title Match	1.556***	-0.310***	-0.175**	-73.53***	3.568***	0.197**
	(0.519)	(0.0886)	(0.0822)	(1.900)	(0.0451)	(0.0930)
Subtitle Match	0.421	1.128***	0.727***	-11.86***	1.289***	0.531***
	(0.474)	(0.0850)	(0.0811)	(1.853)	(0.0459)	(0.0952)
Apple×Post	0.323**	0.0340	0.00912	7.077***	-0.0952***	
	(0.137)	(0.0925)	(0.0354)	(1.012)	(0.0298)	
AppleCompetitor×Post	0.182***	-0.0101	-0.00491	0.134	-0.00692	
	(0.0560)	(0.00964)	(0.00750)	(0.167)	(0.00462)	
#Category	0.0663***	-0.00828***	-0.00361***	-0.0240	-0.000430	
	(0.00513)	(0.00135)	(0.000916)	(0.0221)	(0.000651)	
Multiple-Category Developer?	0.268	-2.070***	-0.0105	8.622***	-0.281***	0.0434
	(0.238)	(0.0850)	(0.0409)	(1.064)	(0.0344)	(0.0658)
Other-Category Same-Developer Average:						
Price	0.187***	0.00943***	-0.00265**	0.0505*	-0.00192**	-0.0141***
	(0.0176)	(0.00131)	(0.00107)	(0.0285)	(0.000895)	(0.00234)
Update Level	0.103***	-0.0332***	0.144***	0.427***	-0.0192***	
	(0.0175)	(0.00361)	(0.00551)	(0.104)	(0.00280)	
Average Rating	0.0919*	0.453***	-0.0486***	-2.196***	0.0714***	-0.0204
	(0.0499)	(0.0181)	(0.00729)	(0.199)	(0.00648)	(0.0145)
Ever Top50	0.0624	0.0101	0.0628***	3.433***	-0.110***	
	(0.177)	(0.0356)	(0.0202)	(0.557)	(0.0173)	
Keyword Un-adoption Ratio	-1.581***	0.115***	0.0334	-4.461***	0.174***	
	(0.0993)	(0.0185)	(0.0228)	(0.491)	(0.0135)	
Days without Search Results Ratio	-0.552***	0.0115	0.0420**	3.240***	-0.124***	
	(0.153)	(0.0310)	(0.0197)	(0.549)	(0.0167)	
Average of Competing Apps' Other-Category Same-Developer Average:						
Price	-0.486***	0.0166	0.0372*	-0.0674	0.0160	0.0493***
	(0.0882)	(0.0203)	(0.0193)	(0.416)	(0.0110)	(0.0190)
Update Level	-1.154***	0.0222	-0.0819*	-0.0213	0.0130	
	(0.226)	(0.0485)	(0.0461)	(1.052)	(0.0288)	
Average Rating	0.976***	0.185***	-0.275***	0.582	-0.0747**	-0.0449***
	(0.372)	(0.0697)	(0.0588)	(1.347)	(0.0367)	(0.00916)
Ever Top50	-2.911*	-0.464	0.514**	-11.07*	0.850***	
	(1.637)	(0.301)	(0.256)	(5.950)	(0.161)	
Ratio.Keyword Un-adoption	1.808***	-0.436***	0.911***	12.17***	-0.708***	
	(0.677)	(0.163)	(0.155)	(3.190)	(0.0834)	
Ratio.Days without Search Results	-2.487*	-0.690***	0.0483	-7.261	0.590***	
	(1.455)	(0.253)	(0.233)	(5.335)	(0.146)	
Ratio.Paid Apps	-0.0392	0.00181	-0.104**	-3.164***	0.0176	0.123*
	(0.425)	(0.0668)	(0.0522)	(1.181)	(0.0327)	(0.0653)
Ratio.Multiple-Category Developers	0.772**	0.101*	-0.0293	0.660	-0.000627	0.144**
	(0.309)	(0.0593)	(0.0521)	(1.139)	(0.0319)	(0.0663)
Constant	-4.360***	3.723***	-0.00568	7.490***	-0.247***	0.437***
	(0.325)	(0.0564)	(0.0427)	(0.880)	(0.0247)	(0.0428)
Observations	52,959	52,959	52,959	52,959	52,959	25,325
Adjusted R-squared	0.372	0.177	0.335	0.644	0.751	0.072

Notes. Robust standard errors in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$. All columns include category-fixed effects and month-fixed effects.
[a]On the supply side, *age* is an included IV, thus I use *age* instead of $\log(age)$.

Table E.20: Compare Structural Estimates with Difference-in-Differences Estimates of Average Treatment Effects of the Search Algorithm Change

ATE	$\log(1+\text{Update Level})$	$\log(\text{Search Ranking})$	$\log(\text{Downloads})$
Structural Estimates	0.0166	−0.0150	0.0342
DiD Estimates	0.0212	−0.0355	0.2210

Notes. Difference-in-Differences (DiD) estimates are taken from Table 3. Structural estimates are computed from difference 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. The market outcomes are computed without fixing pre-installed apps' search rankings.

Table E.21: Effects of Fixing Pre-installed Apps' Search Rankings

Variable	Update	Search Rankings	Downloads
No-game Simulation(%)		0.0064	0.0210
Partial-game Simulation(%)	0.0002	0.0064	0.0207
Full-game Simulation(%)	0.0606	0.0055	0.0205

Notes. Figures are relative L1-Norm of each column variable y_{jgt} in percentage: $100\times[\sum_{jgt}(|y_{jgt}^{[fix]}-y_{jgt}^{[0]}|)]/[\sum_{jgt}y_{jgt}^{[0]}]\%$, where $y^{[fix]}$ denotes the market outcome with fixing pre-installed apps' search rankings, $y^{[0]}$ denotes the market outcome with flexible pre-installed apps' search rankings as assumed in supply-side estimation. The sample covers affected markets with pre-installed apps: Entertainment category, Music category and Utilities category in the months of June and July in 2019. Search rankings and downloads are evaluated with expectation over the truncated set of possible search rankings. No-game simulation hold update levels fixed. Partial-game simulation endogenizes positive update levels but hold update portfolios fixed. Full-game simulation endogenize both positive update levels and update portfolios of top5 developers in each category.

Table E.22: Effects of Self-preferencing on Positive Update Levels

	Status-quo	Shut-down	Percentage Change(%)			
Positive Update Level	mean	mean	mean	min	max	std
Market-level Average	0.7951	0.7956	0.06	0.01	0.09	0.03
Product-level	0.8060	0.8064	0.07	−0.19	4.37	0.31

Notes. Update level is $\log(1+n_{jt})$, where n_{jt} is weighted number of updates, and the weights are based on the length of release notes. Update portfolios are holding fixed. Figures reported are among independent apps with positive update levels and valid profit functions in the data. For the status-quo case, there is estimated preferential treatment of Apple's apps in the search ranking algorithm. For the shut-down case, there is no self-preferencing. Product-level percentage change of update levels are relative to the product's status-quo update level.

Table E.23: Partial-Game Counterfactual Simulation: Effects of Self-preferencing on Search Rankings, Installations and Welfare

	Variable	Status-quo	Shut-down	Mean Δ	Mean $\%\Delta$
(1)	Average Search Rankings	38.92	38.92	0	0
(2)	- Independent apps	39.86	39.23	-0.63	-1.56
(3)	- Apple's apps	14.35	31.85	17.51	142.07
(4)	Total Installations (million)	15.16	15.40	0.24	1.58
(5)	- Independent apps	15.06	15.33	0.27	1.78
(6)	- Apple's apps	0.10	0.08	-0.03	-26.59
(7)	Consumer Surplus (million \$)	321.41	321.96	0.55	0.17
(8)	Total Search Costs (million \$)	15.39	15.35	-0.04	-1.52
(9)	Total Realized Utility (million \$)	336.79	337.31	0.52	0.15
(10)	Producer Surplus (million \$)	76.60	77.01	0.41	0.62

Notes. In the partial game, all independent apps with valid profit functions and positive update levels are allowed to change their positive update levels, but update portfolios are fixed. For the status-quo case, there is estimated preferential treatment of Apple's apps in the search ranking algorithm. For the shut-down case, there is no self-preferencing. Producer surplus are total revenues net of variable update costs across all independent apps with valid profit functions.

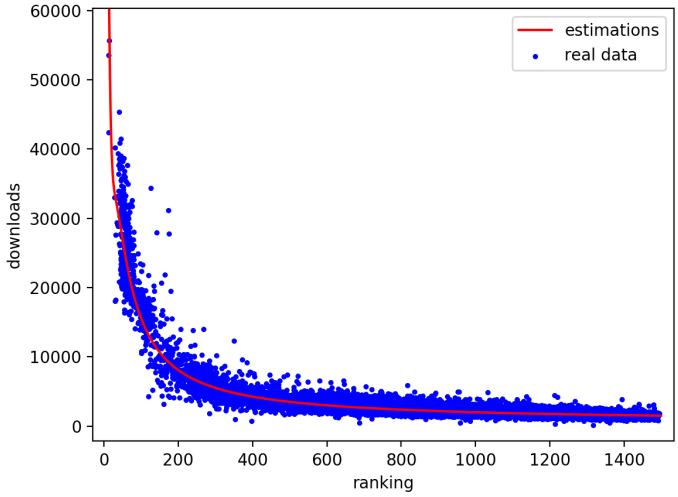
Table E.24: Computational Error in Consumer Surplus

	Max Relative L2 Norm		Max Relative L-infinity Norm	
	Status-quo	Shut down	Status-quo	Shut down
No Game	0.33	0.37	0.26	0.30
Partial Game	0.33	0.36	0.26	0.30
Full Game	0.33	0.37	0.26	0.30

Notes. Figures are 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 are measured by maximum relative L2 norm in the left panel, and maximum relative L-infinity norm in the right panel, across all simulated observations.

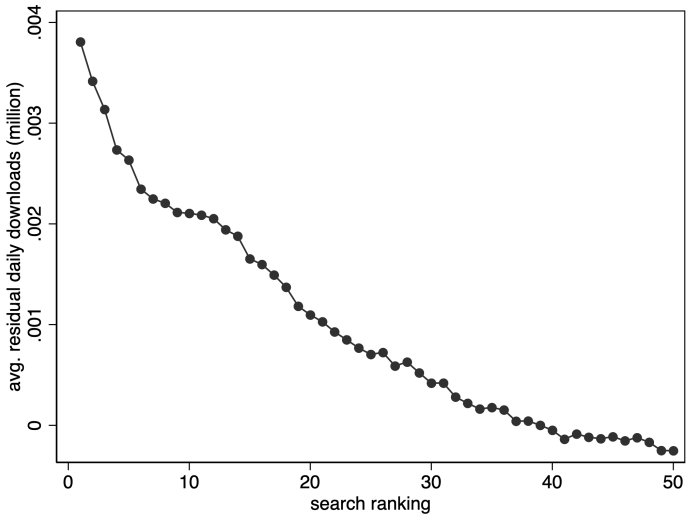
Appendix F Figures

Figure F.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 F.2: Weighted Average Residual Downloads across App/Day/Keyword Given the Search Ranking



Notes. The figure plots weighted average residual daily downloads against search ranking. The average is across app/keyword pairs where the app shows up on the given position in the search results of the keyword. The residuals are from installation price, installation payment type, category-fixed effects, and daily fixed effects. The weights are based on search volume, which is an integer between 5 and 100, constructed by Apple to index how many consumers search for the keyword on a day.

Figure F.3: Average Search Rankings of Non-preinstalled Apple's Apps around the Search Algorithm Change in July 2019.

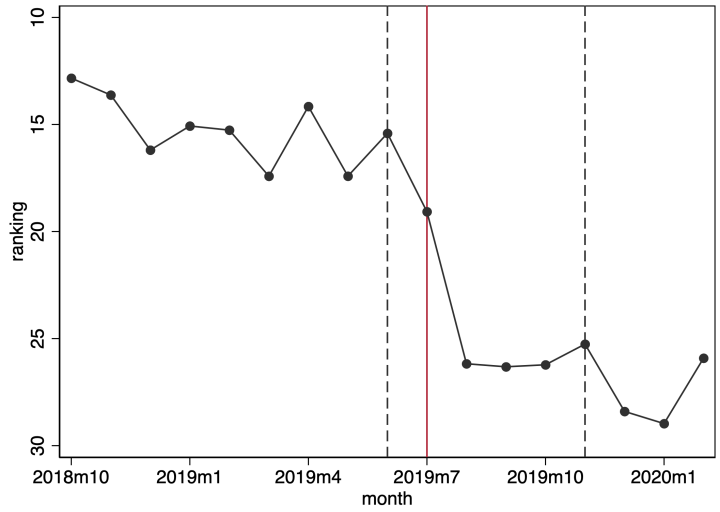
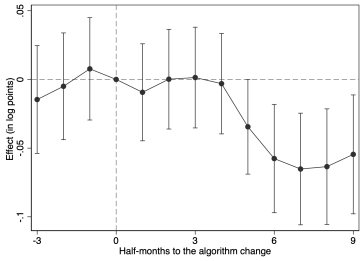
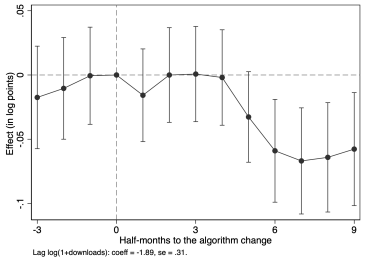


Figure F.4: Effect of Reduced Dominance of Platform-owned Products on Independent Apps, by Half-month from Search Algorithm Change

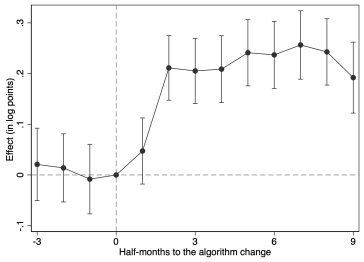
(a) Search Ranking



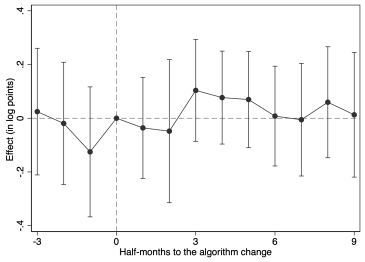
(b) Search Ranking (lag downloads)



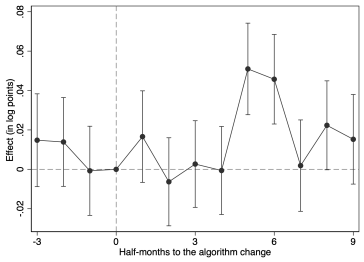
(c) Downloads



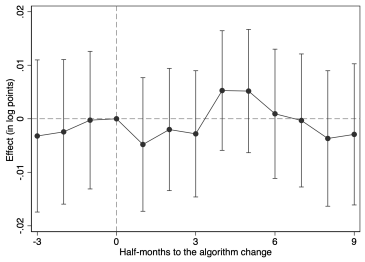
(d) Conversion Rate



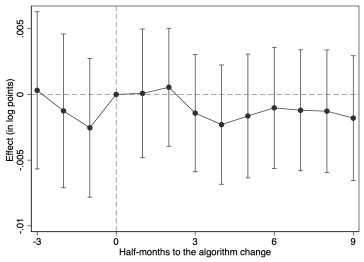
(e) Update Frequency



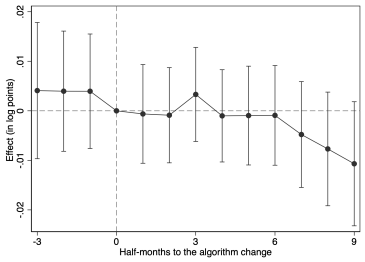
(f) Price



(g) Average Rating



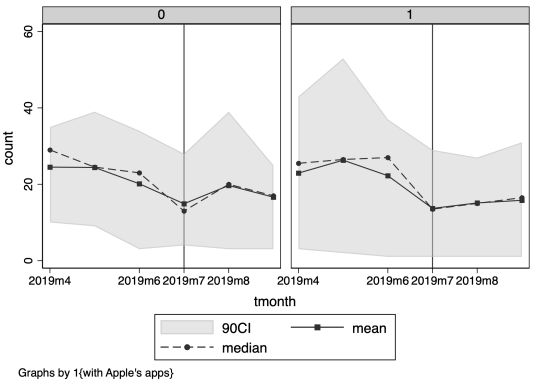
(h) File Size



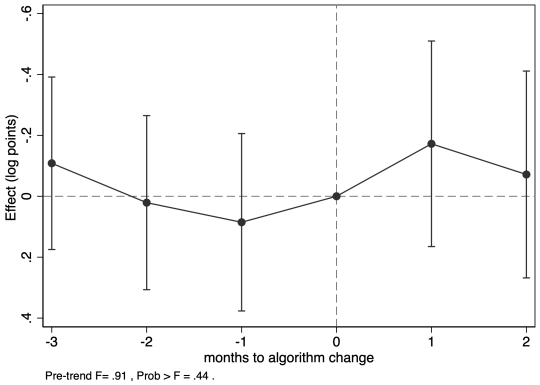
Notes. The charts present point estimates for each half-month using the difference-in-differences specification as specified in Section 3.2. The omitted period is the half-month prior to launch of the search algorithm change. Error bars indicate 95% confidence interval using standard errors robust to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

Figure F.5: Entry Around the Search Algorithm Change

Panel A. Number of New Apps



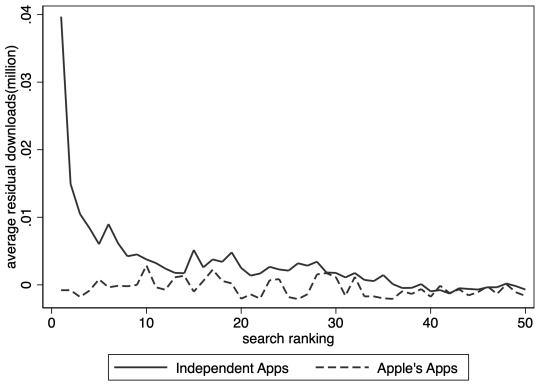
Panel B. DiD Estimates



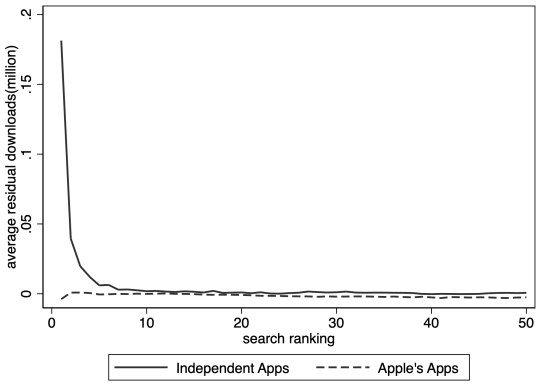
Notes. The sample to draw the figures include independent apps that were ever ranked top50 in category-specific top grossing charts during April - September 2019. Panel A shows the number of new apps in categories with Apple's apps and categories without Apple's apps. To generate Panel B, I regress the 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 F.6: Independent Apps *v.s.* Apple's Apps: Residual Downloads at Each Search Ranking, Before and After the Search Algorithm Change

Panel A. Before

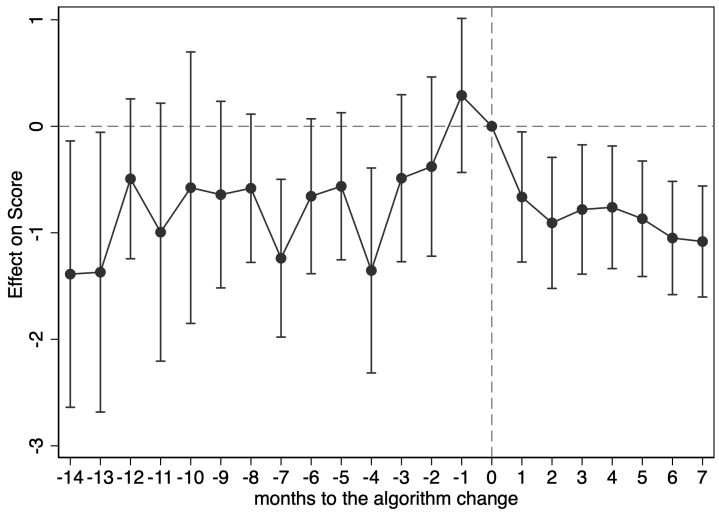


Panel B. After



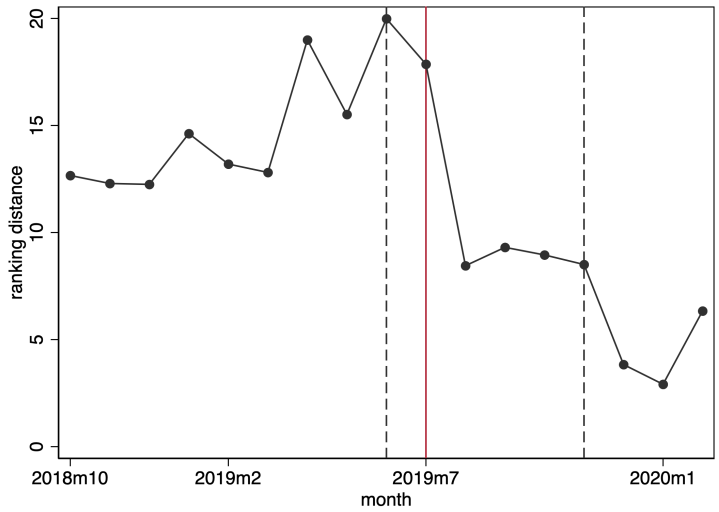
Notes. The figure compares the average residual downloads of independent apps and Apple's apps on the same position in the search results for the same keyword across different days.The residual downloads are residuals from price, installation payment type,category-fixed effects and daily fixed effects. 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 F.7: Dynamic Preferential Treatment Parameters on Apple Relative to July 2019



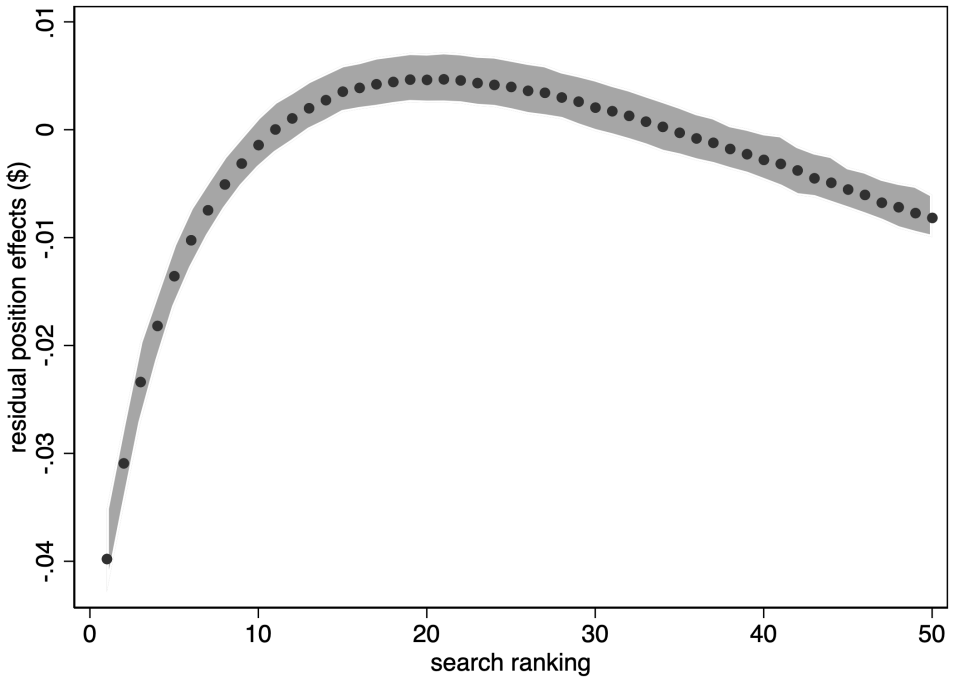
Notes. The figure presents the estimated coefficients on Apple-ownership indicator in each month, while taking the month of the search algorithm change (July 2019) as the reference point. It shows that the degree of self-preferencing significantly decreased after the search algorithm change.

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



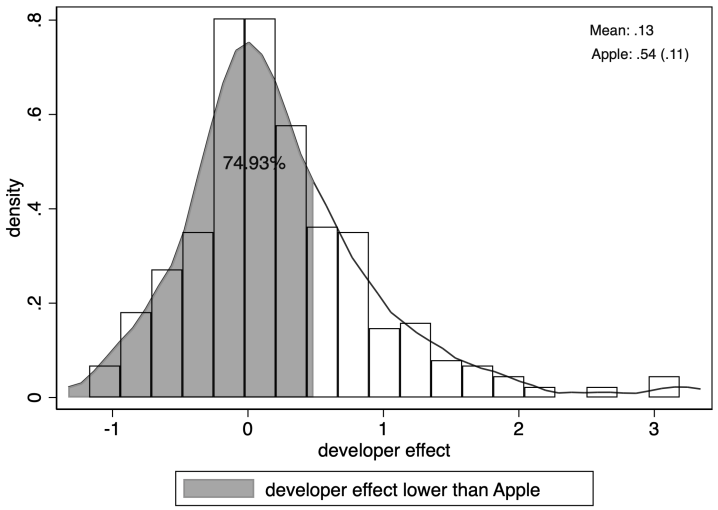
Notes. The figure presents the gap between average observed within-market search rankings of Apple's apps to average within-market rankings of residual downloads of Apple's apps in each month. The residual downloads are residuals from price, installation payment type, category-fixed effects, and daily 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 F.9: Residual Position Effects across Search Rankings



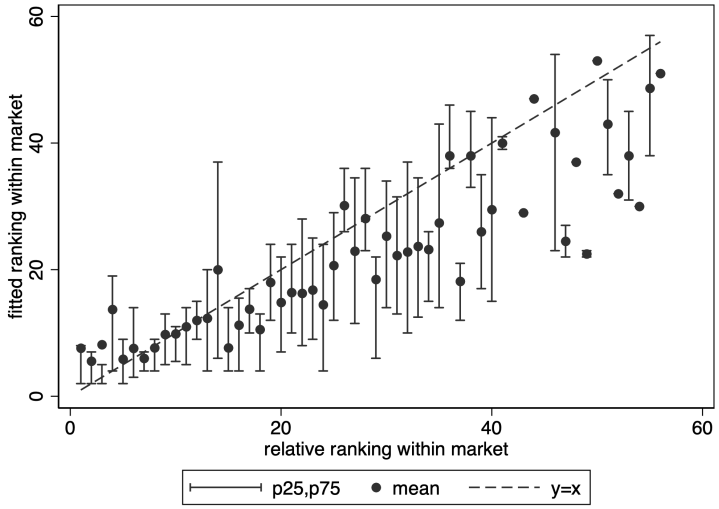
Notes. The points report average residual position effects at each integer search ranking. The residuals are from market-fixed effects. The shaded area covers the 5-th percentile and 95-th percentile of residual position effects. The U-shape curve indicates inelastic demand for apps ranked top in search results and high search costs for apps ranked low in search results.

Figure F.10: Histogram of Static Developer-fixed Effects in Search Ranking



Notes. This figure presents the histogram of static developer-fixed effects across all developers. To estimate the developer-fixed effects, I replace the interaction terms between the Apple-ownership indicator and month indicators in Equation 8 with developer-fixed effects, where the single-product developers are normalized as the reference group. It shows that, even when allowing all developers to have their own advantage or disadvantage in the search ranking algorithm, Apple ownership still generates larger advantages than most developers.

Figure F.11: Rank-ordered Logistic Regression Model Fitness for Apple's Apps



Notes. The figure presents predicted most-likely within-market ranking (y-axis) against observed within-market ranking (x-axis) across markets for Apple's apps. Bars indicate the 25 percentile and 75 percentile of fitted within-market rankings across Apple's apps that have ranked at the given observed within-market ranking.