"How Can I Be of Service?"—A Comprehensive Analysis of Web Service Integration Practices

Siddhi Baravkar*, Olivia Pellegrini*, Pratiksha Gaikwad*, Zheng Song*, and Eli Tilevich†
*Department of Computer and Information Science, University of Michigan at Dearborn
Email: {siddhib, opelle, prati, zhesong}@umich.edu
†Dept. of Computer Science, Virginia Tech
Email: tilevich@cs.vt.edu

***Abstract*—Despite the widespread adoption of Web services in modern computing applications, there remains a lack of a systematic approach that can guide service developers in creating appealing services. This paper addresses this gap by presenting findings from a comprehensive study of RapidAPI web services, the largest service marketplace, and their integration into GitHub-hosted applications. We collected data on over 16K RapidAPI services and 19K corresponding GitHub repositories invoking these services, evaluating each service based on metrics such as latency, reliability, pricing, community support, and provider support. Our analysis examines how these metrics influence service popularity and usage patterns on GitHub. We manually analyzed 800 GitHub repositories and identified developers' service selection preferences and integration patterns, considering alternative services and their features. Additionally, we classified GitHub developers based on proficiency levels to understand how developers' levels of proficiency impact their service selection and integration strategies. Our findings offer insights for service marketplaces to recommend integration-friendly services and for service developers to create offerings tailored to real-world application needs.**

***Index Terms*—Service marketplace, Service selection, RapidAPI, GitHub, Service metrics**

I. Introduction

In the realm of web services, service selection represents a fundamental and enduring problem. This problem can be articulated as follows: when confronted with a collection of equivalent web services offering similar functionalities and usable interchangeably, how to identify and select the service that most effectively fulfills an application's functional requirements. Existing approaches solve this problem based on various facets, including latency, cost, reliability, community support, and trust [1], [2], [3]. These approaches frequently operate under the assumption that integration developers prioritize specific service facets when selecting services. However, the extent to which integration developers prioritize these facets remains unclear, leaving service providers uncertain about which aspects of their services require improvement.

In this work, we take an empirical approach to this problem by conducting a large-scale study. Rather than surveying and interviewing integration developers, our study derives actionable insights by studying how they integrate services in their source code. To execute our empirical study at a sufficiently large scale to derive meaningful insights, we had to overcome the following challenges: 1) how to identify service invocations within the source code, as developers often utilize various libraries with diverse functionalities; 2) how to distinguish the invoked services in the absence of a standardized format, such as differentiating service from other HTTP requests; 3) how to identify and extract relevant metadata associated with services.

In the service-oriented architecture (SOA), a service registry enables providers to register their services and integration developers to find the services they need [4], [5]. Recent years have seen the rapid growth of a new type of service registry that we call *Marketplace*, which provides a one-stop solution for integrating services throughout application lifecycle: 1) Integration developers can use keywords to search for services; 2) when invoking a service, the request is sent to the marketplace, which authenticates the credentials of the requests and serves as a middleman delegating the requests; 3) the marketplace further measures the amount of requests, the latency and successful status of each request, and displays such metadata on each service's web page; 4) based on the billing strategies specified by service providers, the marketplace handles the billing for integration developers' service subscriptions; 5) the marketplace also provides discussion forums for integration developers and service providers to communicate directly.

Due to their convenience and utility, the SOA ecosystem makes extensive use of marketplaces, such as RapidAPI, and BaiduAPI, which attract 16K services, over 4 million users [1], and 400 billion invocations per month [6]. These emerging platforms also enable large-scale empirical studies. We take advantage of marketplaces as the source of relevant service metadata, while using open-source software repositories as the source of service invocation statistics. Juxtaposing the obtained information provides a powerful avenue for better understanding integration developers' rationale for selecting and integrating services.

This paper reports on the results of a large-scale empirical study that focuses specifically on how developers select and integrate services. We collected 16K services with their metadata from RapidAPI, and 19K GitHub repositories that invoke these services. We first analyzed how different service facets impact their popularity as measured by RapidAPI and as measured

[1]https://rapidapi.com/company/

by us from mining GitHub repositories, identifying the root causes for the differences. To pinpoint the exact reasons why developers select particular services, we randomly picked 800 services used in different repositories. We manually labeled whether alternate services could replace them by inspecting their call sites. We also divided integration developers into categories based on their proficiency levels and studied how they differed in service selection and integration.

The paper makes two primary contributions. Firstly, it introduces a novel methodology for empirically studying how integration developers select services, accompanied by the public release [2] of all collected data for use by fellow researchers and practitioners. Secondly, through analysis of the gathered data, the paper addresses the following research questions:

- **RQ1:** What are the common characteristics shared by services managed through RapidAPI, as well as the GitHub repositories and developers who utilize these services?
- **RQ2:** How do various service facets correlate with their usage patterns, as observed on both RapidAPI and GitHub platforms?
- **RQ3:** What factors primarily influence integration developers in selecting a service from a set of equivalent options with similar functionalities?
- **RQ4:** How does the proficiency level of developers influence their practices in selecting and integrating services?

The rest of this paper is organized as follows. Section II introduces the background and related work. Section III details how we collected data from RapidAPI and Github, and answers RQ1. Section IV analyzes data and provides answers to the other RQs. Sections V and VI introduce the threats to validity and implications of our research. Section VII concludes this paper.

II. Background and Related Work

In this section, we introduce background knowledge about using RapidAPI as a marketplace, existing work on service selection, and empirical study approaches for service integration and available datasets.

A. Service Registry and How RapidAPI is Different

Traditional service registries serve as a bridge for integration developers to find services. RapidAPI, initially founded in 2015, is now the largest service registry. It takes a brand new role in service-oriented architecture as a service marketplace, which acts as a delegation between service providers and consumers. All service requests are sent to the delegation servers of RapidAPI, which further query the actual servers managed by service providers to obtain the results and return them to integration developers.

In particular, *the marketplace* acts as a *'One-stop-solution'* providing Service consumers with features like service discovery, QoS monitoring, API documentation, API service subscription, and billing all within a Platform as a Service (PaaS) offering, supporting the entire software development life cycle. [7]. *The service providers* share with the platform the invocation URL of their services, along with the documentation and example code for using the services and their price. *The integration developers* can discover the service and utilize it by subscribing to the service and managing billing on a platform itself [8].

RapidAPI consists of 49 different categories [3] of services like Sports, Finance, Data, Entertainment, etc., and 522 *collections* [4] represent a group of APIs sharing common characteristics. RapidAPI editors manually pick the services in each collection.

In RapidAPI, each service contains a cluster of endpoints that provide distinct functionalities. All the endpoints in a service share the same invocation URL and RapidAPI measures the QoS and the usages by services, not endpoints. Hence, in our data collection and analysis, we carry on the definition of services from RapidAPI and measure the performance and usages for each service, or say, a cluster of endpoints.

B. Service Selection Criteria

Selecting the appropriate service that aligns with the desired requirements is essential for developers to meet both functional and performance expectations. Numerous research studies [9]–[12] have been conducted to assist developers in recommending services or selecting optimal services, focusing primarily on evaluating or predicting QoS parameters such as availability, latency, reliability, etc. Other than QoS, cost [13] for invoking services, the level of community support for services [14], [15], and the quality of documentations [16] are also considered in service selection. Our empirical study considers all of the aforementioned criteria.

There are also other criteria for selecting services. For example, services' QoI (quality of information) [17], i.e., accuracy, updated frequency, and data completeness might be considered for service selection; the QoS perceived by end users may also be considered [18], [19]; the design patterns and anti-patterns of services may also be considered [20]. However, this paper excludes these criteria from our study, as the QoI, end-user QoS, and design anti-patterns are hard to measure on a large scale.

C. Empirical Study of Service Integration

Understanding integration developers' preferences has been an important problem. There have already been multiple small-scale studies on this topic. For example, [21] surveyed less than 300 users on their opinions towards selecting one SMS service from 92 candidate services. Its result indicates that the service facets, ranking by their impact on the selection results, are 1) functionality of the service; 2) reliability; 3) cost; 3) developer support; and 4) latency. However, this rating for SMS service is ad-hoc and may not represent the general developer's preferences. Similarly, [20] interviewed 40 developers about their preferences over 3 sets of services,

[2]https://anonymous.4open.science/r/Web_Service-_Research-66B4/

[3]https://rapidapi.com/categories

[4]https://rapidapi.com/collections

which still provides limited insights and might be biased. Due to the lack of proper methodology, large-scale studies on developers' preferences for selecting and integrating services in the wild are yet to be conducted.

III. Data Collection

This section delves into our methodology for collecting and labeling data from RapidAPI and Github. Fig. 1 shows the flow for the data collection, in which *Selenium web driver* [22] was used to crawling and MySQL database for storage. We repeated this data collection process for two rounds over 8 months, to further understand how services' performance and usage change.

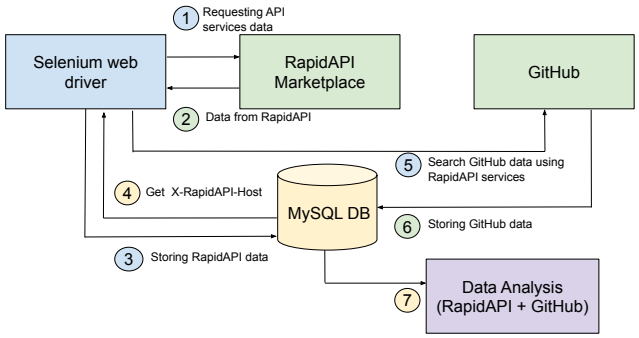


Fig. 1: RapidAPI and GitHub data collection workflow

A. Collecting Service Metadata from RapidAPI

Initially, we obtained the full list of web services from all 49 RapidAPI categories. For each web service in the list, we crawled its detail page. We obtained metrics that include popularity, latency, service level, pricing, hostname, guiding documents, and discussion forums, as indicated by Fig. 2. The popularity, latency, and service level with hostname (in Fig. 2a) are directly measured by RapidAPI, as it delegates service requests. The guiding documents (in Fig. 2b) are also specified by service providers to guide integration developers. For the discussion forums (in Fig. 2c), we collected pages of discussions and the number of replies. The pricing (in Fig. 2d) is set by service providers and may have different formats. For example, developers can set the price for the monthly subscription, the quota (hourly, daily, or monthly request limits), and the per-request cost if the total requests exceed the quota. We further collected the collections, which are similar services manually grouped by editors.

B. Collecting Services Usages from Github

Invoking services from RapidAPI always requires specifying *X-RapidAPI-Host* of the service, which has a similar format of "service name"+"p.rapidapi.com," with "service name" being unique to each service. For example, the RapidAPI host for "Bitcointy" service is "community-bitcointy.p.rapidapi.com." Hence, we searched on Github for repositories using the *X-RapidAPI-Host* of services as the keyword, as specified by steps 4 and 5 in Fig. 1. We collected each repository's metadata, including repository URLs, stars, forks, watchers (Fig. 2e) and its owner (Fig. 2f). We also collected each owner's repository count and followers to measure their proficiency.

C. Statistics of Collected Data:

Platforms / Data collection	Round 1	Round 2
RapidAPI services	16,613	16,395
RapidAPI services used in GitHub	1,500	1,677
Repositories using RapidAPI services	10,317	19,733

TABLE I: Data collection from RapidAPI and GitHub

We conducted two rounds of data collection, in July 2023 and March 2024, to understand how RapidAPI services and their usage changed in 8 months. Table I displays the data collection statistics gathered from RapidAPI and GitHub across two rounds. We observe that 31.60% services available in the first round were removed from RapidAPI at the time of our second round collection, with similar amount of services being added. The utilization of RapidAPI services on GitHub experienced a surprising increase of 93.6%, from 10K repositories to 19K repositories.

Figure 3a illustrates the statistical overview of RapidAPI services. In round 1, we gathered data from 16,613 services, the metrics of 41.96% of which turned undefined. Undefined metrics occur whenever the number of developers using a service is insufficient for RapidAPI to collect enough information about the service's popularity, latency, and service level. Similarly, in round 2, data from 16,395 services were collected, with 47.89% of them featuring undefined metrics. Round 2 shows a 5.94% increase in services with undefined metrics, as compared to round 1. Among the 31.60% services removed from round 2, 67.90% exhibit undefined metrics.

Observation 1: RapidAPI Services are Vibrant

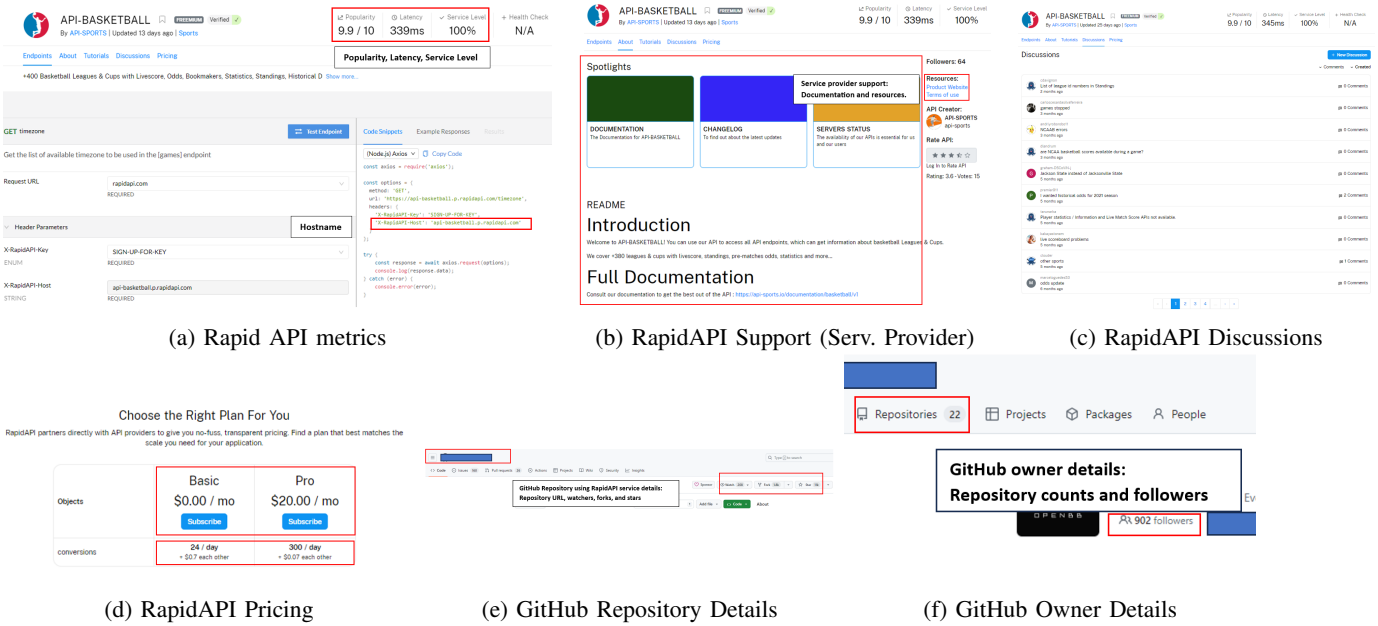
Services on RapidAPI are actively updated (with 31.6% removed in the past year and more added) and widely used by (open source) software.

D. Labeling RapidAPI Service Facets

The service metadata provided by RapidAPI provides numeric values for each service's latency, service level, and popularity [5]. We further define how to label three additional facets, including pricing, support from service developers, and support from the community, as detailed below.

- **Popularity:** The popularity metric is associated with the invocations for a particular service. It is depicted by the RapidAPI platform on a scale of 0 to 10 with a granularity of 0.1, where a higher score indicates greater service utilization. The value is calculated by RapidAPI using two factors: how many developers are using each service and how many requests are sent within a certain period. However, RapidAPI's calculation algorithm is not publicly accessible.

[5]https://docs.rapidapi.com/docs/faqs



(a) Rapid API metrics (b) RapidAPI Support (Serv. Provider) (c) RapidAPI Discussions

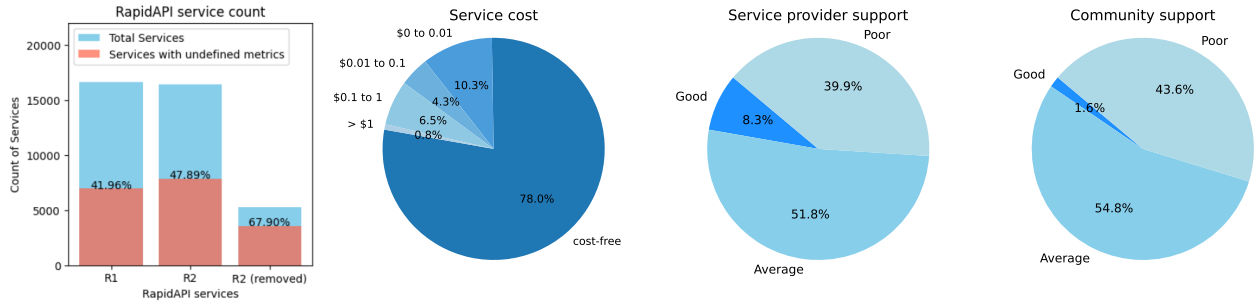
(d) RapidAPI Pricing (e) GitHub Repository Details (f) GitHub Owner Details

Fig. 2: RapidAPI and GitHub Data Collection

- **Latency:** Noting that RapidAPI delegates all service requests, latency is measured as the average time (ms) it takes for a delegation server to receive a response from the API server for all calls within the last 30 days.
- **Service Level:** The service level metric indicates server reliability through successful service calls. On the RapidAPI platform, the service level is depicted on a scale of 0% to 100% with a granularity of 1%, where a higher score indicates greater service reliability.
- **Pricing:** RapidAPI enables service providers to specify various price models, including basic, pro, ultra, and mega, each with a distinctive pricing structure for every service. Initially, we classified the services into three categories: cost-free (services with more than 100 requests per day), limited cost-free (services with fewer than 100 requests per day), and paid services. For the limited cost-free and paid categories, we calculated a "cost per request" by 1) choosing the price model with the lowest cost, as most integration developers may choose the lowest cost; 2) dividing the cost by number of requests allowed per period. As an illustration, in the case of the service depicted in Figure 2d, we opt for the basic model due to its lower cost. However, this model only allows for 24 requests per day, falling short of our set threshold of 100 requests per day. Consequently, we will consider the cost per request as \$0.7. In cases where the cost per request parameter is unspecified, we have evaluated it by $\frac{Requests per day}{cost}$.
- **Support from service providers:** This metric involves information about the documentation support provided by the service providers on the marketplace to assist the developers in utilizing their services. Using documentation content and resource links (see Fig. 2b) as parameters, we set a threshold value of documentation word count of at least 100 words, as a minimum of 100 words can adequately convey necessary details. In particular, we categorized the service providers' support as: 1) *good*, if a service has at least a 100-word count for the documentation as well as a resource link; 2) *average*, if a service has either one but not both; and 3) *poor*, if neither is available.
- **Support from the community:** This metric provides information regarding whether the integration developers of a service are forming a community to help each other. For each service, we consider how many discussion threads have at least one reply, as answering questions forms a healthier community than purely asking questions. In particular, we categorized the service community support as: 1) *good*, if 70% of discussions have replies; 2) *average* for 70% to 10%, and 3) poor for 10% to 0%.

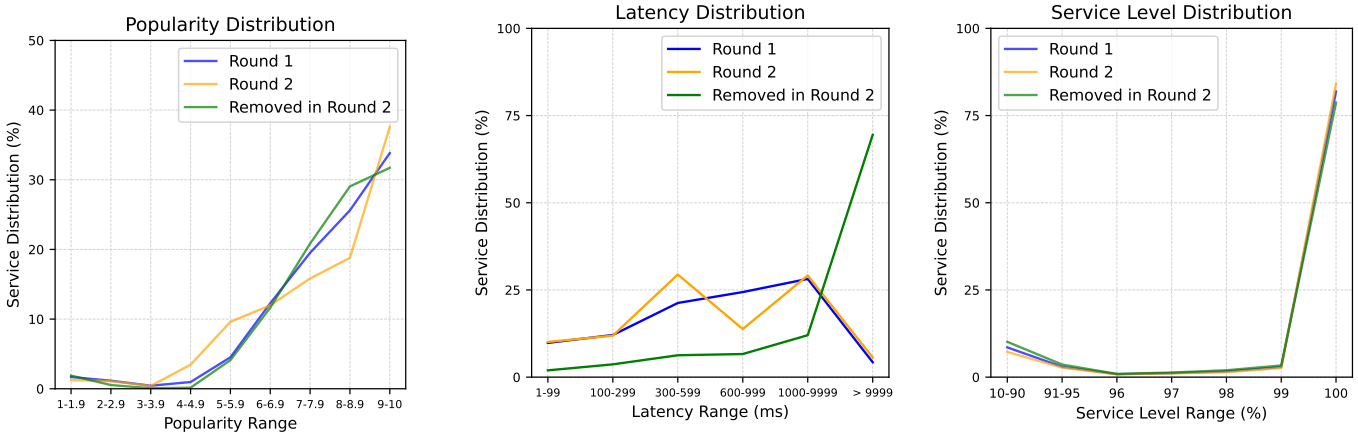
Fig. 3 shows the service distributions based on our customized metrics, i.e., cost per request, service developer's support, and community support. We observe that: 1) most services fall into the cost-free category, but still more than 10% services charge for more than 1 cent per request; 2) over 60% service developers provide good or average support for their integration developers; and 3) only very few services have an activity community.

Fig. 4 shows the service distribution based on popularity, latency, and service level, with the blue line denoting services collected in round 1, yellow for round 2, and green for services included in round 1 but removed in round 2. Note that here we excluded the services with undefined metrics. We observe that 1) the service level and popularity of services in round 1, round 2, and removed are almost identical; 2) over 70% of services removed from round 2 have high latency, making the average latency of services in round 2 smaller than round 2; 4) however, many services still suffer from poor QoS, i.e.,



(a) RapidAPI service statistics (b) Pricing Categorization (c) Service provider's support (d) Community support

Fig. 3: RapidAPI Service Statistics and Distributions based on customized metrics



(a) Popularity Distribution (b) Latency Distribution (c) Service Level Distribution

Fig. 4: RapiAPI Service Distribution

over 20% services with reliability lower than 98% and over 25% of services with latency higher than 1,000 *ms*.

One of our unexpected findings was that many services provided insufficient QoS. A commonly shared understanding is that the QoS of services is bound by SLA (a service-level agreement). Hence, we further studied the documentation of all services, looking for the keywords "SLA" and "service level agreement." To our surprise, among all the services collected in round 2, only 15 services mentioned SLA variations in their documentation, and only 3 with legitimate SLA documents.

Observation 2: RapidAPI Services Characteristics

Developers can use over 3/4 of services offered on RapidAPI for free, but many services suffer from poor QoS and inadequate support from service providers and the community.

E. Labeling GitHub Developers' Proficiency

The developer proficiency level may impact the service selection as per their preferences. In particular, to measure the proficiency of a developer who owns a GitHub repository that integrates RapidAPI services, we consider the following three aspects: i) the number of repositories owned, ii) the number of forks on the repository that invoke RapidAPI services, and iii) the number of followers.

As the resulting metric has multiple dimensions, we applied the following rules to classify developers into three levels:

- **Skilled Developers:** If all three aspects fall in the top 20% of their respective counts.
- **Average Developers:** If any 2 aspects fall in the top 20% of their respective counts.
- **Novice Developers:** If only one or no metric falls in the top 20% of their respective counts.

Fig. 5 shows the distribution of developers' proficiency levels under our definition. We observe that less than 25% of developers that integrate services in their software are classified as skilled and average, with a majority of users fall into the novice category.

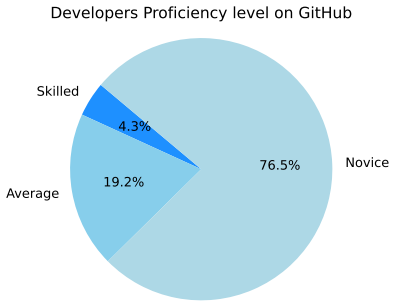


Fig. 5: Distribution of Developer's Proficiency Levels

F. Labeling GitHub Service Usages

We further measured the main purposes of the repositories using services. We analyzed the primary programming language used in each repository, which is provided by Github, and categorized the purposes of the repository by the primary language, as listed below:

- Web Development, with languages being Javascript, Typescript, HTML, CSS, Jupyter Notebook, EJS, Svelte, Vue, ASP.NET, etc.
- General Purpose Scripting, with languages being Python, Ruby, PHP, Perl, Groovy, Lua, etc.
- Mobile App Development, with languages being Swift, Kotlin, and Dart.
- Backend Systems, with languages being C Sharp, Java, Go, Rust, C++, Scala, Haskell, etc.
- Databased, with languages being SQL, HCL, and PLSQL.
- Document and Template Languages, with languages being Tex, XSLT, and Roff.
- Others, with languages not in the above categories.

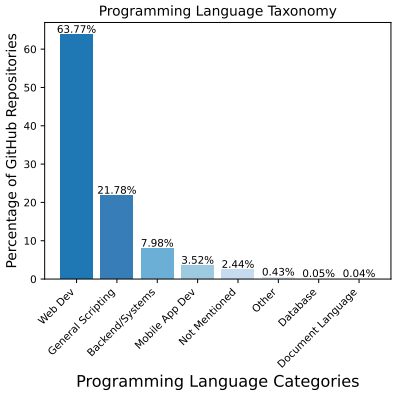


Fig. 6: Distribution of Developer's Proficiency Levels

Observation 3: RapidAPI Services Usage

Services are widely used in web development, but less used in mobile Apps.

IV. Data Analysis

This section details the data analysis we employed to address Research Questions 2 to 4.

A. Correlation between Service Facets and Usages

First, we aim to answer *RQ2*. The popularity of RapidAPI is indicated by both the number of developers utilizing the service and the frequency of invocations made to those services. We correlated this metric by examining the number of repositories utilizing each service on GitHub, terming it as the average repositories per service to evaluate the usage of each RapidAPI service on GitHub.

Fig. 7f illustrates the relationship between the average RapidAPI popularity of all services and the average number of repositories on GitHub utilizing these RapidAPI services. We observe that as RapidAPI popularity increases, so does the number of average repositories on GitHub. This correlation indicates that GitHub can serve as a suitable platform for studying the RapidAPI usage patterns. However, for the services used in more than 40 repositories, we observe a slight decrease in their average popularity, which may be caused by not so many end users using these repositories and sending fewer requests, as RapidAPI popularity is measured by both the number of subscribers and the invocation frequency.

Further in Fig. 7 We compared various RapidAPI service facets and analyzed their impact on service usage. As a general trend, we observe that higher service level (reliability), lower cost, and better support from service providers and the community attract more developers, as indicated by Figs. 7b, 7c, 7d, and 7e, which is not surprising.

Observation 4: Service Usage Characteristics

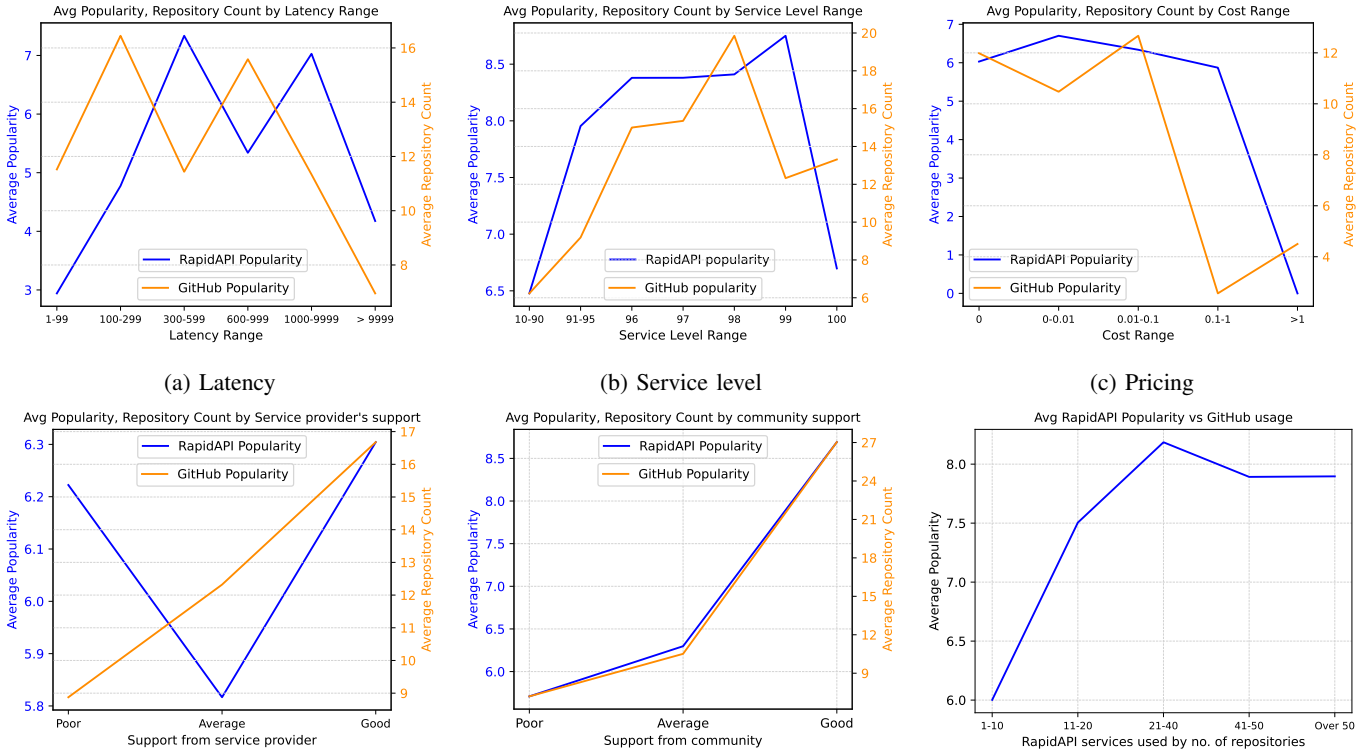
Developers prioritize reliability and latency within acceptable thresholds. Nevertheless, robust support mechanisms, from both the developer and the community, remain crucial.

Other observed insights are as follows:

- Latency (see Fig. 7a): 1) extremely low latencies of 1 to 99 *ms* may not increase the service's popularity, as confirmed by both RapidAPI and Github; 2) different impacts of latency on service usages are observed, i.e., Github developers favor services with less than 1s of latency, while RapidAPI indicates developers favor services with less than 10s of latency.
- Reliability (see Fig. 7b): as confirmed by both RapidAPI and Github, developers tend to prefer services that offer good reliability, typically in the range of 97% to 99%, rather than those with perfect reliability of 100%, which is against the common assumption that services need to guarantee the reliability of several nines. One possible reason causing this phenomenon could be that, as more developers use a service, sending their requests, the service's reliability decreases. Or we can say that instead of developers choosing services with lower reliability, developers' choices cause services' reliability to decrease.
- Support from Service Providers (see Fig. 7d): the correlations between support from service providers and service usages measured by GitHub and RapidAPI differ. Services with little or no documentation and no resource URLs may be more popular than services with either of these properties present, as observed on RapidAPI. Finding a plausible reason that explains this phenomenon remains an open research question.

B. Selecting Services from Similar Options

Here we delve into a comprehensive comparison between the services selected by developers through RapidAPI and potential alternatives that were not chosen. The aim is to examine the facets of RapidAPI that significantly influence developers' choices when choosing a service among alternatives with similar functionalities.



(a) Latency (b) Service level (c) Pricing

(d) Supp (Serv. provider) (e) Supp (Community) (f) RapidAPI vs. GitHub

Fig. 7: Service facets Vs Service usages

Three co-authors manually inspected how services are integrated into Github repositories. By calling RapidAPI-provided functions, they randomly selected 800 GitHub repositories, from which they identified approximately 382 unique RapidAPI services, all of which they inspected manually.

To identify alternative service options for services utilized within each GitHub repository, we adopted the following procedure: 1) we inspected the input/output parameters for each service in use; 2) to help with manually labeling alternative services, we filtered out candidate services that fall into the same collection with the service in use, and appended the candidate service set using keyword search on RapidAPI; 3) if at least two of the three manual inspectors agreed that a service in the candidate set can take the inputs of the service in use and generate outputs with similar physical meaning, we marked the service as an alternative for the current service in use. Out of 382 services analyzed, we identified 332 as having potential replacements.

We further analyzed each RapidAPI facet distribution for 332 services with potential replacements and compared them with the possible alternatives. The facets analysis is divided into 3 categories:

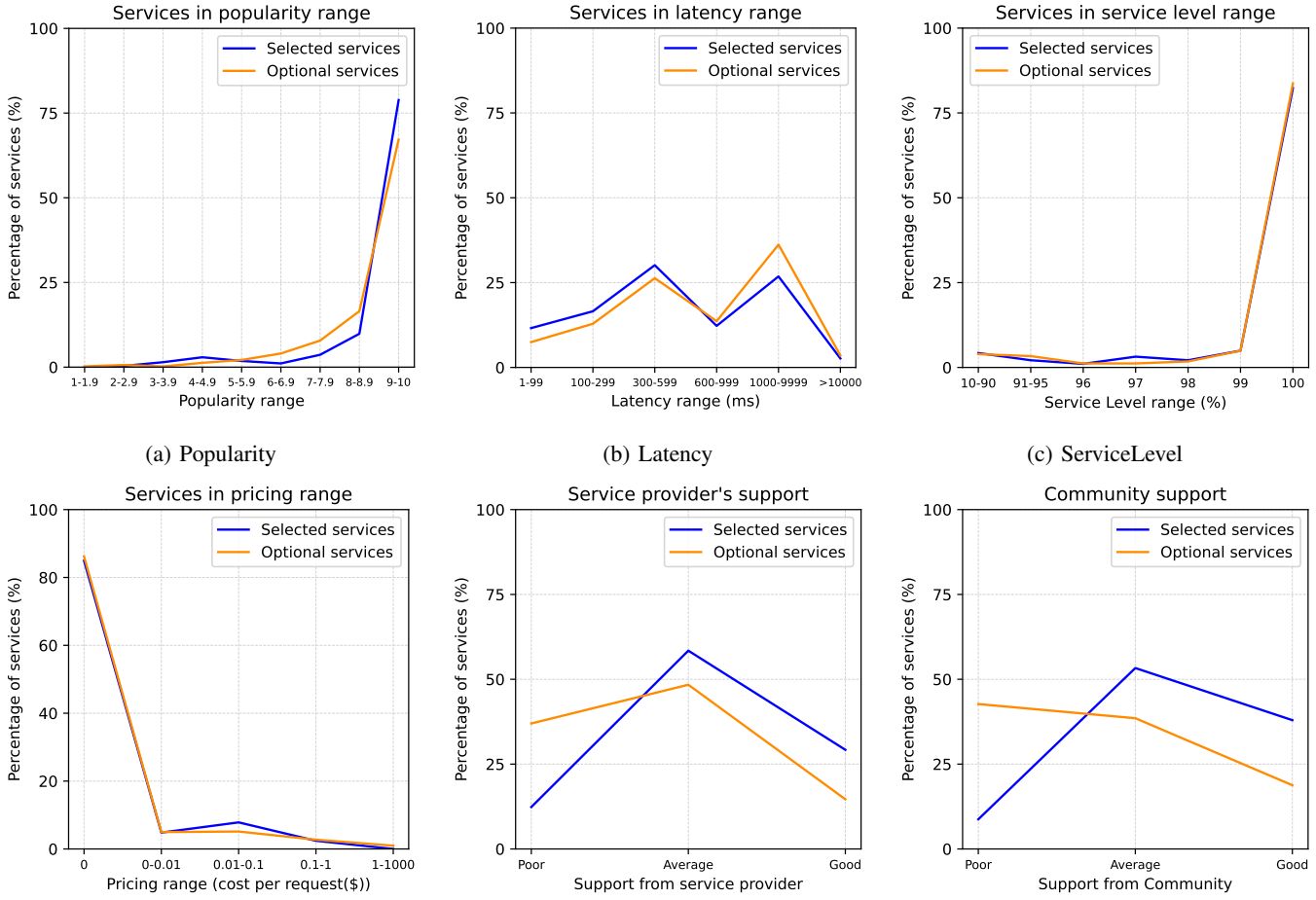
- Facets with no impact: Service level and pricing show similar trends for the selected and alternative services as in Fig. 8c 8d. For pricing, Adam Smith's "invisible hand" [23] can influence the pricing strategies of competing services, as each market consistently provides more appealing pricing to attract customers. The pricing graph indicates a similar outcome, with the selected and alternative services reaching nearly identical pricing.
- Facets with minor impact: The popularity and latency as shown in Fig. 8a 8b may have a minor impact while selecting services from a set of similar options. Selected services exhibit comparatively higher popularity, alongside low-latency performance, specifically those with latencies below 1 second.
- Facets with significant impact: The support provided by service providers and the community significantly influences developers, as illustrated in Fig. 8e and 8f, where it is evident that most selected services enjoy better support from both service providers and the community compared to the alternative services.

Observation 5: How Do Developers Select Services

When selecting services from equivalent options, developers are impacted by the following factors ranked from the highest to the lowest: 1) Service provider support and Community support; 2) Latency and Popularity; and 3) Service level and Pricing.

C. How Developer Proficiency Impacts Service Integration

We analyzed the developers' categories as per their proficiency levels in Sec. III-E labeled as as "Skilled", "Average", and "Novice" developers. In particular, we studied how developer proficiency impacts two aspects of service



(a) Popularity (b) Latency (c) ServiceLevel

(d) Pricing (e) Support from Serv. Provider (f) Support from Community

Fig. 8: RapidAPI facets comparison for selected services and their alternatives

integration: 1) service selection; and 2) error handling. We paid special attention to error handling, driven by the finding of many popular services being afflicted by high latency and low reliability.

1) **Service Selection**: Although services selected by skilled developers slightly outperformed as compared to those selected by average and novice developers in terms of service level, pricing, and community support (Fig. 9c, 9d and 9f) the difference between them was not particularly evident.

As per Figure 9b, the latency aspect reveals that skilled developers prefer integrating services with lower latencies, typically ranging between 1 millisecond to 1 second, compared to average and novice developers. Moreover, beyond the 1s threshold, the usage of services by skilled developers declines, compared to their average and novice developers. This finding indicates that skilled developers recognize the importance of fast responsiveness when selecting services.

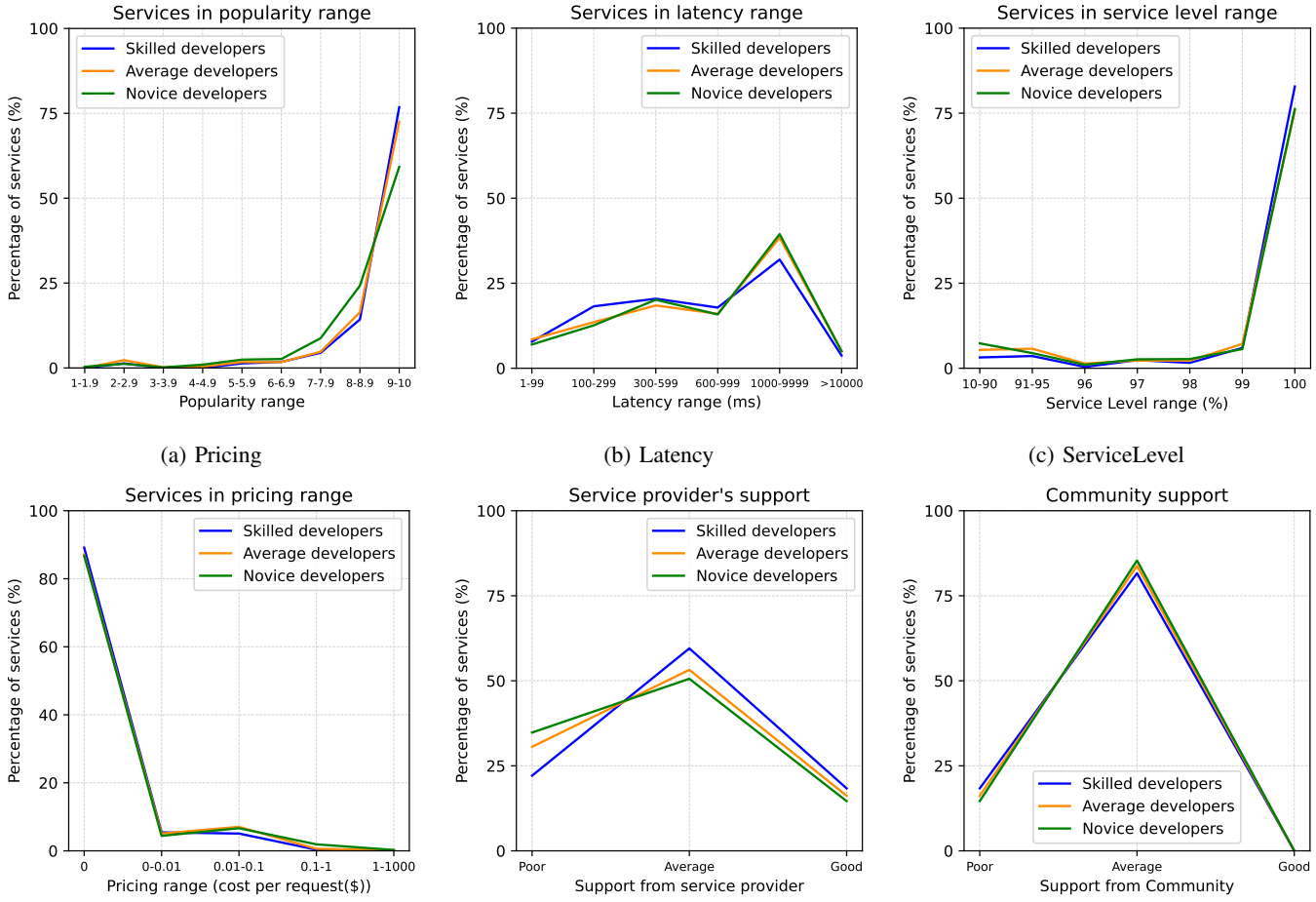
Fig. 9e illustrates both skilled and average developers consider service provider's support like good documentation details, resource links, tutorials, and similar resources when selecting services. It might seem surprising, but skilled developers highly value documentation quality. Experienced developers might view good documentation as a sign of solid software engineering practices. Essentially, well-made software often has detailed documentation. While experienced developers may not always need the documentation for integration, they like to see it as evidence of well-crafted services.

Skilled and average developers also prefer the popularity facet distribution, as shown in Fig. 9a. One explanation for this finding is that services gain popularity as skilled and average developers select them rather than some services possessing high popularity upfront.

Observation 6: Developer Proficiency Characteristics

Compared with novice developers, skilled developers are more concerned about services' popularity, latency, and documentation support from service providers.

2) **Error handling:** Our analysis, as depicted in Fig. 9b and 9c, indicates that many services are impacted by high latency and low reliability. Without proper error handling, an App may be frozen, waiting for services' responses or even crash if the request is not successful. Many HTTP libraries have a default timeout (e.g., 5 minutes for Python *requests*). The default timeout may be inappropriate for services, as some of which may take an unusually long time to execute, while others



(a) Pricing (b) Latency (c) ServiceLevel

(d) Pricing (e) Support from Serv. Provider (f) Support from Community

Fig. 9: RapidAPI facets as per developers' proficiency

need to be retried as soon as their execution fails to meet a shorter latency. Hence, developers must realize the importance of error handling in service invocations and implement proper strategies such as retries after failures and customized timeout.

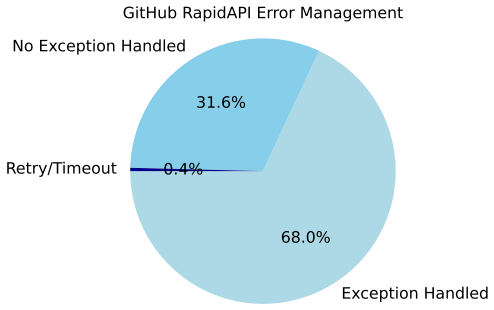


Fig. 10: Error Handling Code Patterns

To understand the error handling patterns followed by the developers, we manually inspected the call site of the 800 repositories in GitHub, which we selected randomly to study developers' service selection preferences. We observed the following distribution in error handling, as shown in Fig. 10:

- No exception handling, 31.6% of repositories.
- Basic exception handling, such as usage of try-catch blocks or providing error details by logging HTTP status codes (e.g., 404, 500) or other customized messages, which comprises 68% of repositories.
- Proper exception handling, such as timeout and retries, which comprises only 0.4% repositories.

Observation 7: Coding Patterns Characteristics

Many service developers are unaware of the pressing need for proper error handling when integrating services into their code.

V. Threats to Validity

Internal validity: When labeling alternative services (Sec. IV-B), we determined whether a service could serve as a replacement at given call sites based on our judgment, which presents an internal validity threat. To mitigate it, three co-authors participated in the manual labeling process, so in instances of disagreement among their opinions, a majority voting mechanism was employed to reach a consensus.

External validity: We focused our analysis on the 53% data subset, which contained QoS metrics measured by RapidAPI,

as illustrated in Fig. 3a. Consequently, this analysis focus may have limited its scope and depth. How we define community support, service provider support, and cost per request may also impact the analysis results. Although more factors may be considered in measuring these service facets, our metrics include the most important factors for each facet. Another potential external threat to the validity of this study lies in the GitHub data collection process related to RapidAPI services, as it relies solely on open-source repositories. This data selection thus excludes any insights that could have been obtained from private repositories inaccessible to us, potentially introducing bias or incomplete representation.

VI. Implications

The study we have carried out discovered 1) the characteristics of RapidAPI-managed services by examining their various facets, 2) how these services are used on GitHub with respect to their various facets, 3) how developers choose services over alternative options; and 4) how developers' proficiency levels impact their choices. Our findings can have useful implications for different stakeholders:

- **For Integration Developers:** The insights into RapidAPI's multifaceted nature can guide developers in systematically evaluating service facets, while also highlighting the importance of robust error-handling strategies.
- **For Service Providers:** These insights can also inform service providers about developer expectations, guiding efforts in improving service usability.
- **For Marketplace Platforms:** Understanding developer proficiency and preferences enables platforms to tailor recommendations, improving satisfaction and platform engagement across varying expertise levels.
- **For Researchers:** This study identified the need for novel approaches for locating services with subpar QoS or documentation, as a way to enhance service provider feedback and platform trustworthiness.

VII. Conclusion

This paper has described a large-scale empirical study on how integration developers select web services. Our findings highlight that numerous services plagued by poor QoS suffer from lackluster support from service providers and the developer communities. Factors, including service level, cost, and support from service providers and communities significantly influence the utilization of RapidAPI services. Among these factors, support emerges as the one that impacts most which services end up being selected. Additionally, when classifying repositories by the developers' proficiency levels, we discovered that skilled developers focus on factors that include popularity, latency, and provider support. Finally, service integration can benefit from better error-handling strategies.

Acknowledgement

This research is supported by NSF through the grants #2104337 and # 2232565.

References

[1] D. Pudasaini and C. Ding, "Service selection in a cloud marketplace: a multi-perspective solution," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 576–583.

[2] A. A. Olu, "Modelling the performance of web services in cloud e-marketplaces based on consumer waiting time and provider cost," Ph.D. dissertation, University of Zululand, 2016.

[3] G. Gu and F. Zhu, "Trust and disintermediation: Evidence from an online freelance marketplace," *Management Science*, vol. 67, no. 2, pp. 794–807, 2021.

[4] M. R. Azmy, Suhardi, and W. Muhamad, "Advanced technologies to support service discovery in service-oriented systems," in *2020 International Conference on Information Technology Systems and Innovation (ICITSI)*, 2020, pp. 300–305.

[5] C. Manchanda, W. Hussain, L. Rabhi, and F. Rabhi, "Towards an api marketplace for an e-invoicing ecosystem," in *Enterprise Applications, Markets and Services in the Finance Industry*, J. van Hillegersberg, J. Osterrieder, F. Rabhi, A. Abhishta, V. Marisetty, and X. Huang, Eds. Cham: Springer International Publishing, 2023, pp. 82–96.

[6] L. Dahlander, D. M. Gann, and M. W. Wallin, "How open is innovation? a retrospective and ideas forward," *Research Policy*, vol. 50, no. 4, p. 104218, 2021.

[7] A. Menychtas, S. G. Gomez, A. Giessmann, A. Gatzioura, K. Stanoevska, J. Vogel, and V. Moulos, "A marketplace framework for trading cloud-based services," in *Economics of Grids, Clouds, Systems, and Services: 8th International Workshop, GECON 2011, Paphos, Cyprus, December 5, 2011, Revised Selected Papers 8*. Springer, 2012, pp. 76–89.

[8] E. Zeydan, L. Blanco, S. Barrachina-Muñoz, F. Rezazadeh, L. Vettori, and J. Mangues, "A marketplace solution for distributed network management and orchestration of slices," in *2023 19th International Conference on Network and Service Management (CNSM)*, 2023, pp. 1–6.

[9] S.-Y. Hwang, C.-C. Hsu, and C.-H. Lee, "Service selection for web services with probabilistic qos," *IEEE transactions on services computing*, vol. 8, no. 3, pp. 467–480, 2014.

[10] Y. Ma, S. Wang, F. Yang, and R. N. Chang, "Predicting qos values via multi-dimensional qos data for web service recommendations," in *2015 IEEE International Conference on Web Services*, 2015, pp. 249–256.

[11] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and qos information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1913–1924, 2014.

[12] D. Pudasaini and C. Ding, "Service selection in a cloud marketplace: A multi-perspective solution," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 576–583.

[13] R. Ramacher and L. Mönch, "Cost-minimizing service selection in the presence of end-to-end qos constraints and complex charging models," in *2012 IEEE Ninth International Conference on Services Computing*. IEEE, 2012, pp. 154–161.

[14] Y. Wang, J. Zhang, and J. Vassileva, "Effective web service selection via communities formed by super-agents," in *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 1. IEEE, 2010, pp. 549–556.

[15] F. Binzagr, H. Labbaci, and B. Medjahed, "Fame: An influencer model for service-oriented environments," in *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*. Springer, 2019, pp. 216–230.

[16] A. Owrak, A. Namoun, and N. Mehandjiev, "Quality evaluation within service-oriented software: a multi-perspective approach," in *2012 IEEE Ninth International Conference on Services Computing*. IEEE, 2012, pp. 594–601.

[17] Z. Song, O. Rowader, Z. Li, M. Tello, and E. Tilevich, "Quality of information matters: Recommending web services for performance and utility," in *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2022, pp. 41–48.

[18] N. C. Mendonca, J. A. F. Silva, and R. O. Anido, "Client-side selection of replicated web services: An empirical assessment," *Journal of Systems and Software*, vol. 81, no. 8, pp. 1346–1363, 2008.

[19] E. Jawabreh and A. Taweel, "Time-aware qos web service selection using collaborative filtering: A literature review," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2023, pp. 55–69.

[20] M. Daaji, A. Ouni, M. M. Gammoudi, S. Bouktif, and M. W. Mkaouer, "Multi-criteria web services selection: Balancing the quality of design and quality of service," *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 1, pp. 1–31, 2021.

[21] M. Bano and D. Zowghi, "Users' voice and service selection: An empirical study," in *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE, 2014, pp. 76–79.

[22] R. Khankhoje, "Web page element identification using selenium and cnn: A novel approach," *Journal of Software*, vol. 1, no. 1, 2023.

[23] J. C. Panzar, *Competition and Efficiency*. London: Palgrave Macmillan UK, 2016, pp. 1–4. [Online]. Available: https://doi.org/10.1057/978-1-349-95121-5_669-1