SaaS Pricing Extraction Report

Version 1.0

No Author Given

ICSOC 2024

22nd International Conference on
Service-Oriented Computing

October 7, 2024

INDEX

Chapter 1

Introduction

The Software as a Service (SaaS) model has revolutionized software distribution by offering cloud-based access to features and technical support services through recurring subscriptions [7]. These subscriptions are organized through a *pricing*, i.e., a structure arranged into various plans with optional add-ons. This approach caters to different customer needs and usage levels, allowing users to contract a version of the software based on their requirements and budgets. By offering tailored experiences, SaaS providers can enhance user satisfaction and drive revenue growth through broader market reach.

However, a recent study [4] reveal a dramatic increase in the *configuration space*, defined as the set of all possible subscription combinations within a pricing. For instance, GitHub's pricing[1] saw an exponential increase (81,354.55% [5]) in potential subscription combinations over the last six years, reaching up to 8,960 distinct combinations in 2024 with just 3 plans, 81 features, 9 usage limits, and 14 add-ons. This complexity underscores the urgent need for robust tools to efficiently manage and model SaaS pricing.

Drawing inspiration from *intelligent contracts*—self-adaptive agreements in digital ecosystems—we propose the notion of *intelligent pricing*[2], i.e. a dynamic, machine-readable pricing that behaves as a software artifact, following the same processes of design, development, and maintenance as other software components. Currently, metamodels like Pricing4SaaS [3] lay the groundwork for modeling SaaS pricing, yet manual processes dominate this space. Relying on Natural Intelligence (NI) to model and update SaaS pricing is inefficient and prone to human errors. Furthermore, the lack of standardized guidelines for representing SaaS pricing on websites implies the development of ad hoc solutions for each SaaS, hindering the transformation to iPricing.

This paper introduces an AI-driven approach to automate this transformation. By leveraging LLMs to automate the extraction and modeling of SaaS pricing elements, we reduce manual intervention, minimize errors, and provide a scalable framework. Our solution builds on the Pricing4SaaS metamodel, transforming static pricing information from SaaS websites into an intelligent pricing. As the SaaS configuration space grows exponentially (as seen in [4]), the adoption of intelligent pricing will become a critical asset for both development and operations teams, facilitating competitive analysis, enhancing operational decision-making, and paving the way for fully automated pricing operations.

In this paper, we present the following key contributions:

1. The introduction of the concept intelligent pricing, a promising solution for Pricing-driven Development and Operation of Saas [2].

[1] Pricing available here.
[2] For the sake of brevity, we may also refer to this concept as iPricing.

2. A novel AI-driven approach to automate the transformation of iPricing, reducing manual intervention and ensuring consistency across diverse SaaS.

3. Validation of the previous approach through a working prototype, demonstrating the system's ability to extract and model all the necessary pricing elements across 30 commercial SaaS pricing websites.

Chapter 2

Methodology and implementation

2.1 Proposed automatic intelligent pricing modeler

This subsection outlines the minimum requirements for developing an extractor capable of automatically transforming a web static pricing into an iPricing by just receiving the URL where the HTML pricing is hosted. As illustrated in Fig. 2.1, the final system is divided into three main components:

- *Information Extractor:* This component is responsible for retrieving pricing data from a given URL using web scraping technologies. After extracting the data, it leverages an LLM to parse and organize the relevant information, filtering out noise and focusing on essential elements for pricing modeling. These key elements include various plans, features, usage limits, and add-ons.

- *Process Engine:* The Process Engine processes the extracted data, validating and verifying the information to correct inconsistencies and mitigate hallucinations commonly found in LLM-generated content (e.g., ensuring there are no duplicate elements). It not only refines the output from the Information Extractor but also generates warnings and errors (e.g., checking discrepancies between annual and monthly pricing) to assist developers in reviewing the resulting YAML file of the subsequent stage.

- *Results Modeler:* Finally, the Results Modeler converts the extracted pricing data into a structured file, using a syntax like Pricing2Yaml[1] [3], transforming static pricing into an intelligent one. Additionally, it generates a log file with warnings and errors to help identify potential risks and inaccuracies in the LLM-generated content.

2.2 AI4Pricing2Yaml: a basic implementation

For this implementation [1], Python was selected due to its popularity and suitability for the project. Python's versatility, particularly its ease of integration with LLMs and its rich ecosystem of AI libraries, makes it an ideal choice for rapid development and experimentation. This thriving ecosystem allows for straightforward model invocation, while also providing the potential to offer the system's functionality through an API. This makes Python highly adaptable to different deployment scenarios, such as a nanoservice, which would be the ideal setup for deploying this system.

In our implementation, we employed an existing LLM model. We initially adopted an externally guided approach by applying basic prompt engineering. This approach has yielded surprisingly effective results, as demonstrated in Section 3. This success suggests that further refinement

[1] Previously referred to in the literature as Yaml4SaaS.

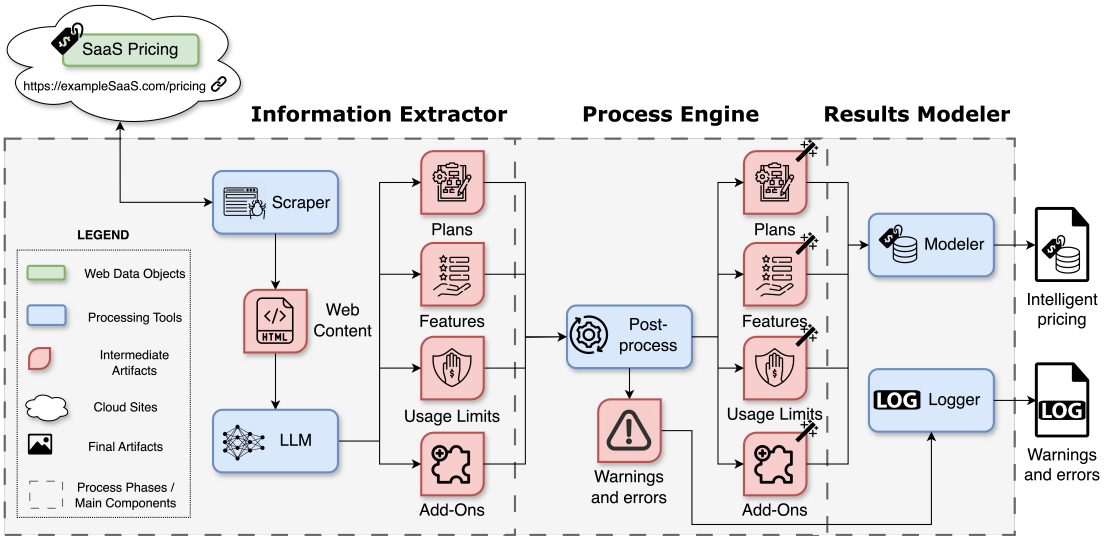

Figure 2.1: An overview of the proposed automatic intelligent pricing modeler.

of the prompts and the use of improved LLM versions could lead to even better outcomes, potentially avoiding the need for fine-tuning.

Considering the three main components of the proposed automatic intelligent pricing modeler, described in the previous subsection, we have fully implemented the first one, the Information Extractor, as it serves as the core component. All the information used by the other components is integrated into the workflow by this component. In other words, the feasibility of the approach largely depends on the performance of the Information Extractor.

The LLM chosen was Gemini 1.5 Flash[2], mainly selected for its context window of up to 10^6 tokens, allowing the entire HTML content of the SaaS pricing page to be fully processed. It also features a free tier, making it ideal for research without incurring costs. Additionally, this LLM is available through an API, with a pay-as-you-go system for the paid tier (Model as a Service).

For extracting pricing information, Selenium was utilized. Selenium's ability to handle JavaScript-rendered content provides an advantage over more traditional tools like BeautifulSoup. This capability allows web scripts to be processed by the LLM. However, dynamic content, such as elements that require user interaction (e.g., clicking a button or opening a modal), can pose challenges in extracting all relevant information accurately.

Finally, an important consideration for the AI4Pricing2Yaml implementation is its design to extract information from a single webpage. If the webpage does not contain all the necessary information, it may be challenging (not to say impossible) to model accurate pricing. Additionally, the current version of the algorithm is optimized for URLs that include a comparison table with all plans and features info. Applying it to different-structured SaaS webpages may result in unexpected outcomes in both execution and the result files.

[2]This model is part of the Gemini family. More info is available here.

Chapter 3

Results and validation

In this chapter, we evaluate the performance of the implemented Information Extractor using the dataset from [5], which includes up to 30 commercial SaaS to assess accuracy across multiple domains. The evaluation [1] focuses on four tasks: extracting plans, features, usage limits, and add-ons.

3.1 Methodology

We utilized the implementation detailed in Chapter 2.2 across 30 SaaS to extract various metrics and evaluate the system's performance [1]. For each SaaS, we extracted their pricing plans and categorized the results as follows:

- **True Positives (TP):** Correct extractions where relevant information (e.g., pricing features) was successfully retrieved.
- **False Positives (FP):** Instances of hallucinations, where the system returned incorrect or non-existent data as part of the response.
- **False Negatives (FN):** Information that should have been extracted but was missed, despite being present in the scraped data.
- **True Negatives (TN):** Information not extracted because it wasn't present in the scraped data, although it could potentially be retrieved through dynamic interactions (e.g., clicking a button to load and display the data).

We introduced a point-based scoring system to quantify the accuracy of our extractions. A full point was awarded for each correct extraction, whether it was a TP or TN, and for each incorrect extraction, either a FP or FN. In cases where partial or incomplete information was retrieved, half points were assigned. For instance, when a feature was correctly extracted but overestimated the number of plans it offered, half of the point was awarded to the TP, and the other half to the FP. The results of these extractions are detailed in Tables 3.1, 3.2, 3.3 and 3.4, which further break down these metrics by SaaS.

Additionally, a set of metrics has been applied to all the tables, where values are provided (except for cases where the values are zero). These metrics are:

- **Accuracy:** The percentage of correct extractions (TP and TN) relative to the total number of extractions.
- **Precision:** The percentage of correct positive extractions (TP) out of all positive extractions (TP and FP).

- **Recall (Sensitivity):** The percentage of correct positive extractions (TP) out of the total expected positive extractions (TP and FN).

- **Specificity:** The percentage of correct negative extractions (TN) out of the total expected negative extractions (TN and FP).

- **False Negative Rate (FNR):** The proportion of FN compared to the total expected positives (FN and TP). It is the complementary of the recall.

- **False Discovery Rate (FDR)**: It represents the proportion of FP out of all positive extractions (FP and TP). It is the complementary of the precision.

Notably, only 15 out of the 30 SaaS platforms had their features, usage limits, and add-ons successfully extracted. This limitation is primarily due to the absence of structured HTML tables or reliance on dynamically generated content, which hindered the extraction process.

3.2 Plans extraction

In our evaluation of the LLM-based system for extracting SaaS pricing plans, we achieved 100% accuracy in 13 out of 30 SaaS. However, the overall mean accuracy was 0.643, with a median of 0.75. The main challenges in correctly extracting plans were twofold: the system often mistakenly included add-ons alongside plans, and it sometimes extracted plans from other products when multiple offerings were present on the same pricing page like what happened with Figma and FigJam plans[1].

On a positive note, there were no FN (perfect recall), meaning that all relevant plans were successfully extracted. This highlights the system's thoroughness in capturing all necessary information.

To provide further insights, we measured performance across the SaaS. SaaS like Box, Buffer, Canva, and ClickUp achieved perfect precision and recall (1.00), indicating flawless extraction. However, others like Salesforce (0.156 accuracy) and Zapier (0.269 accuracy) showed much lower accuracy due to a higher number of FP, reflecting the difficulty of distinguishing plans from other elements.

It is also worth noting that the system was unable to extract dynamically loaded prices, which appear only after pressing a button in the web page.

3.3 Features extraction

Our feature extraction system demonstrated strong overall performance, with a mean accuracy of 0.882 and a median of 0.963 across tested SaaS platforms. Precision was 0.911, and recall was 0.964, indicating effective extraction of relevant features while minimizing hallucinations and other mistakes.

Several SaaS, achieved perfect precision and recall, correctly identifying all relevant features without any FP. Furthermore, other SaaS like Buffer and Figma also showed near-perfect accuracy (0.985 and 0.994, respectively), confirming the system's efficacy with well-structured feature lists.

However, challenges arose with some SaaS, which had lower accuracy scores due to higher FDR. For example, Dropbox had 16 FP out of 78 features (FDR of 0.208), while Mailchimp had 26 FP out of 106 features (FDR of 0.254). These errors were often due to misclassifications and "hallucinations," where the model confused non-features (e.g. add-ons) with actual features.

[1] You can find both pricings in https://www.figma.com/pricing/.

Table 3.1: Plans Extraction Metrics

SaaS	TP	FP	FN	TN	Total	Accuracy	Precision	Recall	FDR	Specificity
Box	7	0	0	0	7	1	1	1	0	0
Buffer	2	0	0	2	4	1	1	1	0	1
Canva	2	0	0	2	4	1	1	1	0	1
Clickup	2	0	0	2	4	1	1	1	0	1
Clockify	8	3	0	0	11	0.727	0.727	1	0.273	0
Crowdcast	2	1	0	0	3	0.667	0.667	1	0.333	0
Databox	3	2	0	0	5	0.6	0.6	1	0.4	0
Deskera	3	0	0	0	3	1	1	1	0	0
Dropbox	3.5	0.5	0	2	6	0.917	0.875	1	0.125	0.8
Evernote	2	2	0	0	4	0.5	0.5	1	0.5	0
Figma	3.5	4.5	0	0	8	0.438	0.438	1	0.563	0
GitHub	1.5	5.5	0	0	7	0.214	0.214	1	0.786	0
Hypercontext	2	2	0	0	4	0.5	0.5	1	0.5	0
Jira	2	1	0	1	4	0.75	0.667	1	0.333	0.5
Mailchimp	4	0	0	0	4	1	1	1	0	0
Microsoft 365	2	0	0	2	4	1	1	1	0	1
Notion	2	3	0	1	6	0.5	0.4	1	0.6	0.25
Openphone	3	3	0	0	6	0.5	0.5	1	0.5	0
Overleaf	7	0	0	0	7	1	1	1	0	0
Planable	3	1	0	0	4	0.75	0.75	1	0.25	0
Postman	4	9	0	0	13	0.308	0.308	1	0.692	0
Pumble	5	0	0	0	5	1	1	1	0	0
Quip	3	0	0	0	3	1	1	1	0	0
Salesforce	2.5	13.5	0	0	16	0.156	0.156	1	0.844	0
Slack	4	0	0	0	4	1	1	1	0	0
Tableau	3	4	0	0	7	0.429	0.429	1	0.571	0
Trustmary	2.5	7	0	1.5	11	0.364	0.263	1	0.737	0.176
UserGuiding	3	0	0	0	3	1	1	1	0	0
Wrike	5	0	0	0	5	1	1	1	0	0
Zapier	1.5	9.5	0	2	13	0.269	0.136	1	0.864	0.174
Mean						0.643	0.614	1		0.526
Median						0.75	0.739	1		0.8

On the other hand, there were SaaS like Jira, with a recall of 0.8 due to missed features, that also posed difficulties. This issue stemmed from the model skipping rows in HTML tables, a problem that needs further investigation.

Table 3.2: Features Extraction Metrics

SaaS	TP	FP	FN	TN	Total	Accuracy	Precision	Recall	FNR	FDR
Buffer	64	1	0	0	65	0.985	0.985	1	0	0.015
Clockify	68	0	0	0	68	1	1	1	0	0
Dropbox	61	16	1	0	78	0.782	0.792	0.984	0.016	0.208
Evernote	28.5	0	0.5	0	29	0.983	1	0.983	0.017	0
Figma	88.5	0.5	0	0	89	0.994	0.994	1	0	0.006
Hypercontext	52	0	0	0	52	1	1	1	0	0
Jira	38	10.5	9.5	0	58	0.655	0.784	0.8	0.2	0.216
Mailchimp	76.5	26	3.5	0	106	0.722	0.746	0.956	0.044	0.254
Microsoft 365	39.5	0	5.5	0	45	0.878	1	0.878	0.122	0
Overleaf	7.5	4.5	0	0	12	0.625	0.625	1	0	0.375
Postman	72	3.5	2.5	0	78	0.923	0.954	0.966	0.034	0.046
Quip	10	0	0	0	10	1	1	1	0	0
Slack	38.5	2	3.5	0	44	0.875	0.951	0.917	0.083	0.049
Tableau	40	0	0	0	40	1	1	1	0	0
Wrike	78	3	0	0	81	0.963	0.963	1	0	0.037
Mean						0.882	0.911	0.964		
Median						0.963	0.985	1		

3.4 Usage Limits extraction

In the usage limits extraction task, system performance varied more than in features extraction, with a mean accuracy of 0.67 and a median of 0.75 across tested SaaS. Despite this, the model achieved a mean precision of 0.838 and recall of 0.778, showing its ability to identify them, although there are still challenges.

SaaS like Evernote, Jira, and Postman performed perfectly, with recall and accuracy scores of 1.0, extracting all usage limits without errors. Mailchimp and Slack also performed well, with recall scores of 0.857 and 0.789, respectively.

However, SaaS like Buffer (0.563 accuracy) and Clockify (0.5 accuracy) struggled due to missed limits and some hallucinations. Microsoft 365 had the worst performance, with 0.125 accuracy and an extremely high FDR of 0.87.

The primary challenge in this extraction task lies in the difficulty for the LLM to accurately interpret and condense usage limit information, further complicated by the diverse ways in which each SaaS platform presents these elements.

Table 3.3: Usage Limits Extraction Metrics

SaaS	**TP**	**FP**	**FN**	**TN**	**Total**	**Accuracy**	**Precision**	**Recall**	**FNR**	**FDR**
Buffer	4.5	1	2.5	0	8	0.563	0.818	0.643	0.357	0.182
Clockify	1.5	0.5	1	0	3	0.5	0.75	0.6	0.4	0.25
Dropbox	8	0.5	4.5	0	13	0.615	0.941	0.64	0.36	0.059
Evernote	6	0	0	0	6	1	1	1	0	0
Figma	1.5	0	0.5	0	2	0.75	1	0.75	0.25	0
Hypercontext	0	0	0	0	0	1	1	1	0	0
Jira	5	0	0	0	5	1	1	1	0	0
Mailchimp	6	0	1	0	7	0.857	1	0.857	0.143	0
Microsoft 365	3.5	23.5	1	0	28	0.125	0.13	0.778	0.222	0.87
Overleaf	2	0	1	0	3	0.667	1	0.667	0.333	0
Postman	8	0	1	0	9	0.889	1	0.889	0.111	0
Quip	0	0	0	0	0	1	1	1	0	0
Slack	7.5	0.5	2	0	10	0.75	0.938	0.789	0.211	0.063
Tableau	0	0	0	0	0	1	1	1	0	0
Wrike	2	0	3	0	5	0.4	1	0.4	0.6	0
Mean						0.67	0.838	0.778		
Median						0.75	1	0.789		

3.5 Add-Ons extraction

The add-ons extraction task showed mixed results, with a mean accuracy of 0.535 and a median of 0.5, reflecting moderate success in identifying SaaS add-ons. Despite lower accuracy, the system achieved a strong mean recall of 0.81, indicating it captured most relevant ones, though mean precision results (0.63).

High performers included SaaS like Dropbox, Postman, Slack, and Mailchimp. Dropbox and Slack achieved perfect recall, successfully extracting all add-ons. Postman showed an accuracy of 0.733 and precision of 0.917, with just 1 missing, while Mailchimp had solid precision and recall (0.667) without missing any.

However, the LLM struggled with SaaS like Jira, Clockify, Overleaf, and Evernote. Jira, Clockify and Overleaf had 0 TP, while they had several FP. Similarly, Microsoft 365 showed low accuracy at 0.2, with 4 FP and only 1 TP. Both Overleaf and Clockify extracted add-ons that did not exist, as neither SaaS offers any add-ons. Consequently, the recall (and the FNR) could not be calculated for these cases.

Two prompts were used in this task. The first prompt leveraged the previous extraction to identify add-ons that appear alongside usage limits in the features table. The second focused on extracting the remaining add-ons from the HTML.

Table 3.4: Add-Ons Extraction Metrics

SaaS	TP	FP	FN	TN	Total	Accuracy	Precision	Recall	FNR	FDR
Buffer	1	1	0	0	2	0.5	0.5	1	0	0.5
Clockify	0	1	0	0	1	0	0	-	-	1
Dropbox	1	0	0	0	1	1	1	1	0	0
Evernote	0.5	3	0.5	0	4	0.125	0.143	0.5	0.5	0.857
Figma	0	0	0	0	0	1	1	1	0	0
Hypercontext	0	0	0	0	0	1	1	1	0	0
Jira	0	20	1	0	21	0	0	0	1	1
Mailchimp	2	1	0	0	3	0.667	0.667	1	0	0.333
Microsoft 365	1	4	0	0	5	0.2	0.2	1	0	0.8
Overleaf	0	3	0	0	3	0	0	-	-	1
Postman	11	1	3	0	15	0.733	0.917	0.786	0.214	0.083
Quip	0	0	0	0	0	1	1	1	0	0
Slack	4	1	0	0	5	0.8	0.8	1	0	0.2
Tableau	3	0	3	0	6	0.5	1	0.5	0.5	0
Wrike	3	2	1	0	6	0.5	0.6	0.75	0.25	0.4
Mean						0.535	0.63	0.81		
Median						0.5	0.733	1		

3.5.1 Ideal extraction

SaaS pricing information is typically presented on websites in unstructured formats, making automated extraction challenging. Drawing on the results of the four extraction tasks and a visual analysis of SaaS pricing webpages over the last six years [4], we propose that an "ideal SaaS pricing webpage" should be organized into the following three clearly defined sections:

1. *Plans:* This section should outline the different subscription options in a simple and structured format like in UserGuiding's pricing.

2. *Comparison table:* All features should be presented in a structured HTML table, preferably as boolean values without any unnecessary details or complexities (e.g. Quip's pricing).

3. *Add-ons:* The add-ons section should mirror the structure of the plans, detailing each add-on's name, price, and unit, as well as specifying which plans are required for subscribing to it (e.g. Postman's pricing).

If usage limits are present, they should follow a consistent format, such as Evernote's pricing, and be included within the features table. Any features unique to add-ons should be listed separately in their own HTML table, without overlapping with the main features table of the plans such as Postman's pricing. This structured approach will improve clarity, facilitate automated extraction, and streamline future transformations, although it may not guarantee perfect results. Moreover, all data should be static, ensuring no additional actions (such as clicking to load hidden information like discounts) are required for complete information.

3.6 Challenges

The extraction system delivers robust performance across several SaaS, particularly in its ability to capture relevant elements as evidenced by high mean recall values. As illustrated in Fig. 3.1, the mean accuracy, precision, and recall for the plans, features, usage limits, and add-ons, demonstrate the system's strong potential. Despite this, the system encounters specific difficulties with certain elements. While plans and add-ons are generally extracted with reasonable recall, they tend to exhibit lower precision when the system processes entire HTML blocks with-

out adequate contextual cues. This often results in noisy or incomplete extractions, particularly when structured elements are inconsistently applied across different websites.

For usage limits and add-ons, there is a noticeable decline in recall due to their complexity and inconsistent representation within pricing structures. Unlike plans and features, usage limits are frequently hidden behind interactive elements or require more advanced reasoning to interpret correctly. This inconsistency leads to challenges in reliably identifying and extracting these elements, even as the system improves in other areas.

In contrast, the extraction of features remains the most stable and reliable, exhibiting minimal fluctuations in both recall and precision. This stability stems from the more clearly defined and consistently presented nature of features across SaaS pricing pages, making them less prone to errors during extraction.

Moreover, other challenges remain, such as dealing with dynamic content. These challenges and the ones aforementioned can be categorized into two main areas:

- *Internal:* These challenges stem from the system's difficulties in accurately modeling extracted elements. Issues such as hallucinations, incorrect modeling of usage limits and add-ons, and occasionally missing features fall under this category. To address these issues, we propose exploring more advanced models (e.g., Gemini 1.5 Pro [6]), enhancing prompt engineering, and leveraging advanced functions like tool calling[2] or structured outputs[3].

- *External:* These challenges relate to the data extraction process itself, such as difficulties in recognizing tables without HTML tags or interacting with dynamic content (e.g., requiring clicks to load additional data). Implementing an LLM agent could significantly improve this, allowing the system to intelligently navigate and interact with complex or hidden content, ensuring comprehensive and accurate data extraction while minimizing noise. Moreover, the integration of knowledge graphs could provide a more structured approach to capturing relationships between pricing elements, enabling to reduce the noise and size of the original HTML content.

In summary, addressing both internal and external challenges will lead to substantial improvements in the system's output. While internal solutions focus on enhancing the system's processing and modeling capabilities, external solutions aim to optimize the quality of input data.

[2]More info about this for the Gemini API at https://ai.google.dev/gemini-api/docs/function-calling.
[3]More info about this for the Gemini API at https://ai.google.dev/gemini-api/docs/structured-output.

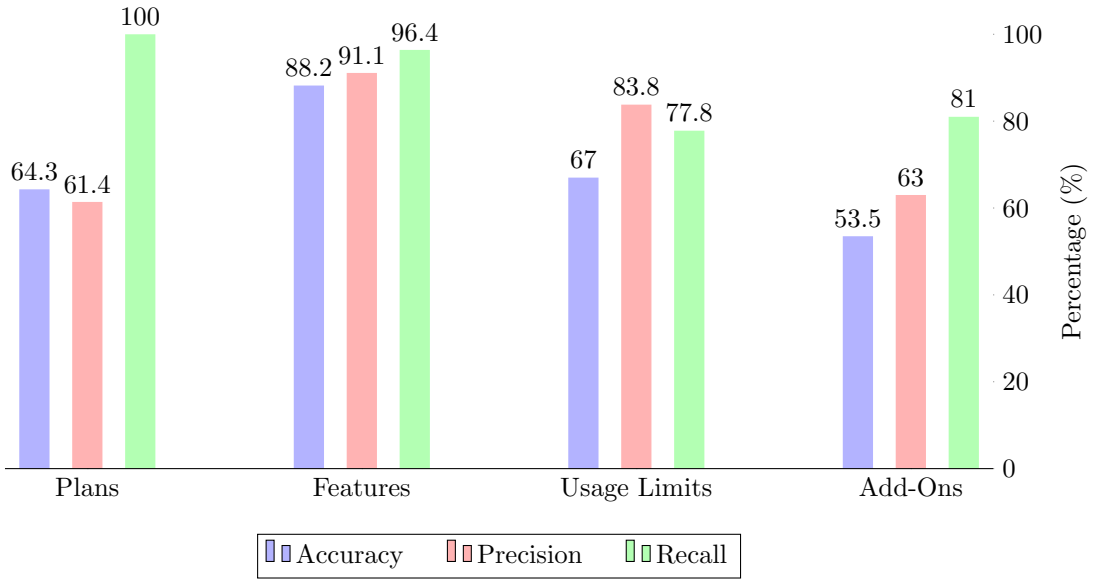

Figure 3.1: Summary of accuracy, precision, and recall metrics per extracted element.

Bibliography

[1] Anonymous: AI4Pricing2Yaml - Supplementary Material (2024), https://anonymous.4open.science/r/icsoc24-AI4Pricing2Yaml/

[2] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing-driven Development and Operation of SaaS: Challenges and Opportunities. In: Actas de las XIX Jornadas de Ciencia e Ingeniería de Servicios (JCIS). SISTEDES (2024), https://hdl.handle.net/11705/JCIS/2024/37

[3] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing4SaaS: Towards a Pricing Model to Drive the Operation of SaaS. In: Intelligent Information Systems. CAiSE 2024. Lecture Notes in Business Information Processing, vol. 520. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-61000-4$_6$

[4] García-Fernández, A., Parejo, J.A., Cavero, F.J., Ruiz-Cortés, A.: Racing the market: An industry support analysis for pricing-driven devops in SaaS. In: Proceeding of the International Conference on Service Oriented Computing, (ICSOC). (In Press) (2024). https://doi.org/10.48550/arXiv.2409.15150

[5] García-Fernández, A., Parejo, J.A., Cavero, F.J., Ruiz-Cortés, A.: SaaS Analysis - Supplementary Material (2024). https://doi.org/10.5281/zenodo.13857484

[6] Gemini Team: Gemini 1.5: Unlocking multimodal understanding across millions of tokens of contex. arXiv (2024). https://doi.org/10.48550/arXiv.2403.05530

[7] Jiang, Z., Sun, W., Tang, K., Snowdon, J.L., Zhang, X.: A pattern-based design approach for subscription management of software as a service. In: 2009 Congress on Services - I (2009). https://doi.org/10.1109/SERVICES-I.2009.40