Faculty of Computer Science and Engineering Skopje

[Department for Intelligent Systems]

Technical Report

Architecting Website Internal Linking Using Advanced Data Processing, Vector Embeddings and Graph-Based Link Prediction Algorithms

sponsored by data & testing partners

September 7, 2024

Author: Emilija Gjorgjevska

Contributors

- PhD Miroslav Mirchev
- PhD Georgina Mircheva

"Koji su dozvolili da sanjam, — "Who let me dream,
koji su omogucili da krenem, — who made it possible for me to go,
koji su virovali da cu stici, — who were waiting for me to arrive,
Vama." — To you."
- Denny Vrandečić

License

History

Version	Date	Comment
1.0	04/09/2024	Open-sourcing v1.0 of the technical solution under Apache 2.0 license

Sponsors: WordLift.io and Kalicube.com

Contents

1 Acknowledgements **4**

2 Introduction **5**
2.1 Aims and objectives 5
2.2 Target audience 6

3 Problem Statement **6**

4 Literature Survey **6**
4.1 Overview of website hierarchies 7

5 The Dataset **13**
5.1 Uniform Resource Locators (URLs) 13
5.2 Raw HTML 14
5.3 HTTP response code 14
5.4 Crawl depth 14
5.5 Inlinks 15
5.6 On the use of first-party data sources in SEO 15
5.6.1 Crawl data 15
5.6.2 Google Search Console (GSC) 16
5.6.3 Google Analytics (GA) 16
5.6.4 Log file data 17
5.6.5 Prioritization and practicality 17

6 System Design and Requirements Engineering **17**
6.1 Design and implementation of the crawler 18

7 Technical Stack and Setup **20**
7.1 ScreamingFrog for web crawling 20
7.2 Python on Google Colab Pro 20
7.3 SQLite for data storage 20
7.4 DuckDB for vector embeddings 21
7.5 Embeddings creation 21
7.6 CSV files for intermediate data storage 21
7.7 GraphSAGE for link prediction 21
7.8 Computer machine details (CPU, GPU, memory) 22
7.9 Subnet verification and IP whitelisting 22
7.9.1 Subnet verification 22
7.9.2 IP whitelisting 22
7.9.3 Collaboration for efficient crawling 23

Sponsors: WordLift.io and Kalicube.com

8 Integration and Workflow **23**
8.1 Data collection and processing workflow tips and tricks 23
8.2 On overloading the target website 23
8.3 Patience during data collection and processing 23
8.4 On saving full HTML files for additional Processing 24
8.5 Ensuring adequate cloud storage 24
8.5.1 Runtime Verification in Google Colab 25

9 Vector Embeddings Analysis **25**

10 GraphSAGE for Link Prediction **26**
10.1 Node embedding generation 26

11 The Final User Interface of the Solution **27**

12 Testing and evaluating results **28**

13 Final Words **28**

Sponsors: WordLift.io and Kalicube.com

1 Acknowledgements

This work would not have been possible without the support and contributions of several individuals and organizations to whom I am deeply grateful.

First and foremost, I would like to express my heartfelt thanks to Jason Barnard, Andrea Volpini, and their teams at Kalicube.com and WordLift.io for providing invaluable data and permission to work with their resources which served as the foundation of this research. Your expertise and willingness to share resources significantly enhanced the scope and depth of this work.

I am also profoundly grateful to Dejan Petrovic (Dan Petrovic) for his insightful suggestions on crawler ideas. Dejan's innovative approaches and practical advice were instrumental in shaping the first technical aspects of this research.

Special thanks go to my esteemed professors, PhD Miroslav Mirchev and PhD Georgina Mircheva, whose guidance and support throughout this process were indispensable. Your ideas for the technical implementations helped bridge theory with practice, ensuring the academic rigor of this work.

To Sara Moccand Sayegh, your tireless efforts in brainstorming, collecting data, and managing technical tasks were invaluable. Our brainstorming discussions, your unwavering encouragement, and your belief in this project pushed it to new heights. I couldn't have done this without your partnership and friendship.

Finally, I would like to thank my family, friends, colleagues, and the international SEO community for their unwavering support and encouragement throughout this journey. Your reality checks kept me grounded and motivated when I needed it most. Your belief in my abilities gave me the strength to persevere and complete this project.

This work is not just mine—it belongs to all of you, and I am deeply appreciative of the unique and irreplaceable role each of you played in bringing it to life.

Sincerely,

Sponsors: WordLift.io and Kalicube.com

2 Introduction

In today's digital landscape, optimizing your website for search engines is essential for driving organic traffic and achieving online success. Internal linking—linking to relevant pages within your site—is a key aspect of SEO. However, traditional internal linking methods can be time-consuming and complex, especially as a website grows in size and depth, requiring more advanced strategies.

Recent advancements in Natural Language Processing (NLP) and Machine Learning (ML) offer new opportunities in SEO, particularly in content analysis and link prediction. These technologies allow SEO professionals to automate and optimize internal linking, ensuring each link has a strategic purpose. Graph-based models, such as GraphSAGE, predict optimal internal links by analyzing the structure and relationships between pages. These models utilize vector embeddings to represent content in a multidimensional space, capturing semantic similarities and relevance.

Our technical report examines the integration of data processing, vector embeddings, and GraphSAGE-like algorithms to refine internal linking strategies for SEO.

2.1 Aims and objectives

This research aims to develop an advanced framework for optimizing internal linking strategies within websites to improve SEO performance, user navigation, and content discoverability. By integrating Natural Language Processing (NLP), Machine Learning (ML), and graph-based algorithms, the research addresses the limitations of traditional internal linking methods, particularly for large-scale websites. Some of our key objectives are provided below.

- Analyze the current limitations: examine existing internal linking practices and identify their shortcomings, particularly in handling complex website architectures with extensive content depth and breadth.
- Explore advanced technologies: investigate the application of NLP, ML, and graph-based models, such as GraphSAGE, for predicting and generating optimal internal links.
- Develop a framework: create a comprehensive framework that combines data processing, vector embeddings, and graph-based algorithms to automate the internal linking process.
- Evaluate effectiveness: apply the proposed framework to a real website.
- Provide strategic insights: offer practical recommendations for SEO professionals on implementing advanced internal linking strategies to enhance search engine visibility and improve user experience.

Sponsors: WordLift.io and Kalicube.com

2.2 Target audience

This research is aimed at tech-savvy marketers and marketing engineers who possess a strong understanding of advanced data analytics and are familiar with the tools and techniques involved in data-driven marketing strategies. Our target audience consists of professionals who either have direct experience with Python and other programming languages or have access to development support within their teams. These individuals are not only proficient in interpreting and utilizing data but are also comfortable working with technical tools to optimize marketing efforts. Whether they are directly coding or collaborating with developers, these experts are well-versed in leveraging data and technology to enhance customer engagement, streamline marketing operations, and drive business growth.

3 Problem Statement

Large websites, especially those in enterprise settings, often contain millions of pages that need to be logically organized and interconnected from both a graph and semantic perspective. The main challenge is to implement intelligent internal linking strategies to achieve this. This is complicated by the fact that webmasters and SEO engineers often lack direct access to the front or backend systems, making large-scale changes difficult.

Effective scalability requires the ability to quickly adapt to new trends on a biweekly or monthly basis, with site architecture automatically adjusted based on SEO priorities set by professionals. We aim to develop a system that provides a robust foundation for addressing this challenge. By utilizing the website's graph structure and semantic data from each webpage, we seek to create a scalable, adaptive solution that improves the organization and connectivity of large-scale websites.

4 Literature Survey

The importance of internal linking in SEO has been extensively discussed in recent literature, highlighting the role of graph theory and dynamic linking strategies in enhancing website structure and user navigation. A study by Omio Engineering[1] outlines how graph theory can be applied to optimize internal linking by treating a website as a network of interconnected nodes, where pages are connected based on relevance and authority. This approach not only improves search engine crawlability but also enhances user experience by guiding visitors through a logical pathway of content that aligns with their search intent.

Another perspective emphasizes that the optimal internal linking structure varies depending on the business model. Kevin Indig[2] discusses how e-commerce sites, content-driven platforms, and service-based websites each require different linking strategies to maximize SEO benefits. For instance, e-commerce sites may benefit from

a hierarchical structure that connects product pages to category pages, while content-driven platforms might adopt a more mesh-like structure to promote related articles. This adaptability ensures that the linking strategy aligns with each business model's specific objectives and user behavior.

WordLift[3] introduces the concept of dynamic internal linking, which automatically generates and updates internal links based on real-time data and evolving SEO priorities. This method uses natural language processing (NLP) and machine learning (ML) to continuously assess the relevance and performance of content, dynamically adjusting links to reflect changes in user behavior and search trends. This approach is particularly effective in large-scale websites where manual linking is impractical and scalability is crucial.

Together, these studies provide a comprehensive view of current approaches to internal linking in SEO, showcasing the need for adaptable and scalable strategies to meet the diverse requirements of different website types and industry contexts.

4.1 Overview of website hierarchies

Website hierarchies [4] are essential for effective information architecture and user navigation on digital platforms. Different architectures—flat, hierarchical, matrix, database, and mixed offer unique ways to organize content, each suited to specific website types and sizes. Flat architectures maximize PageRank flow but struggle with larger sites, while hierarchical structures use parent-child relationships and taxonomies to create more organized, content-rich experiences. Matrix architectures focus on interconnectedness, offering dynamic exploration, and database-driven sites adapt content dynamically based on user input, providing flexibility and scalability.

A key insight here is that a website is a link network, represented as a directed cyclic graph (DCG), where each page represents a node, and internal links between pages form the directed edges:

- websites represent *directed graphs* because links have a specific direction from the source page to the destination page.
- websites represent *cyclic graphs* since the graph has loops or cycles. For example, the second-level nodes (links) from the picture below link to each other (also the leaves back to the homepage).
- website graphs are *weighted graphs* since links have different importance (weights or costs).

Intelligent website structures enable efficient navigation and topical clustering[5], crucial for search engine optimization (SEO). Let's explore some website hierarchies:

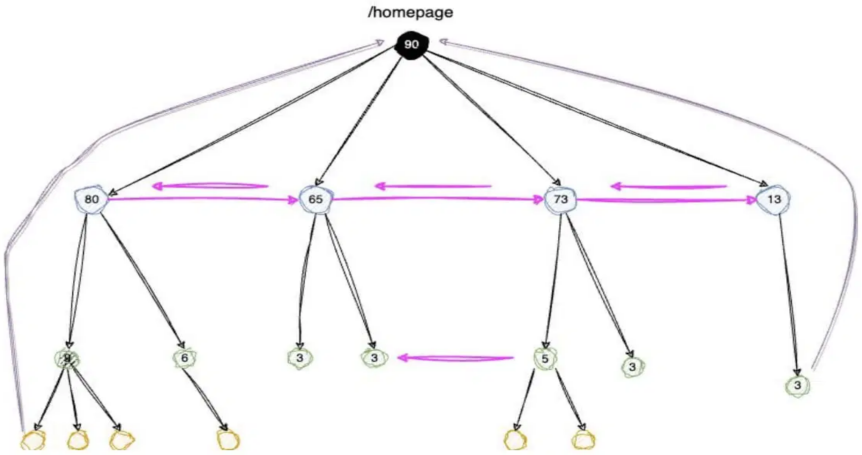

Figure 1: Website as a Directed Cyclic Graph (DCG)

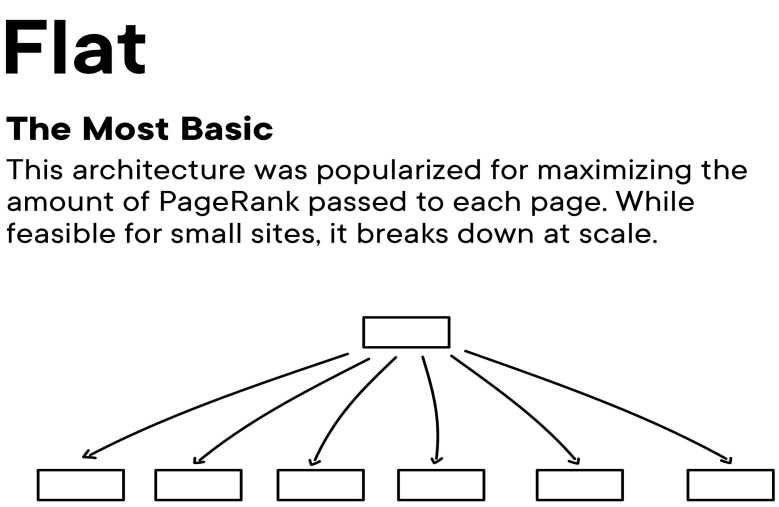

Figure 2: Flat hierarchy

Considering the diversity of website hierarchies influenced by factors such as industry, business model, and technical configuration, our focus is on identifying effective linking candidates in general, rather than prescribing specific connections. However, our research will also provide insights and suggestions on potential ways these links can be established.

Sponsors: WordLift.io and Kalicube.com

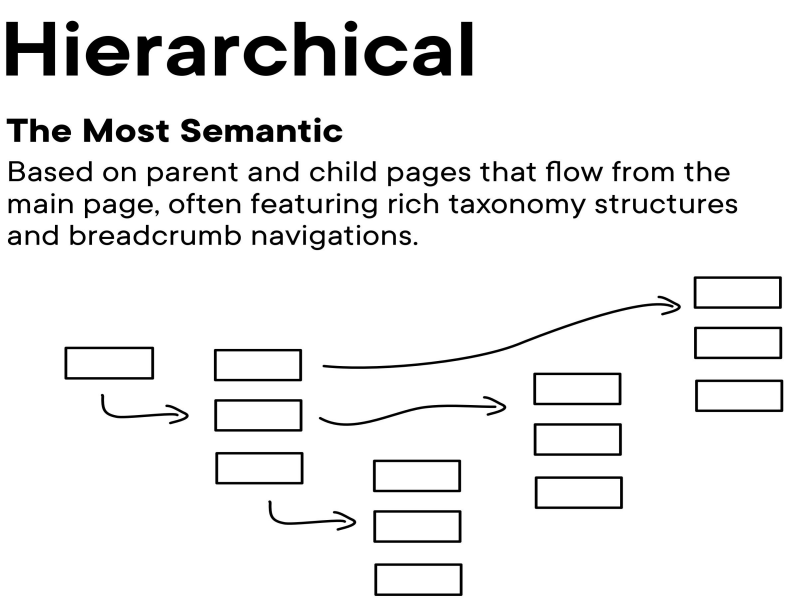

Figure 3: Hierarchical hierarchy

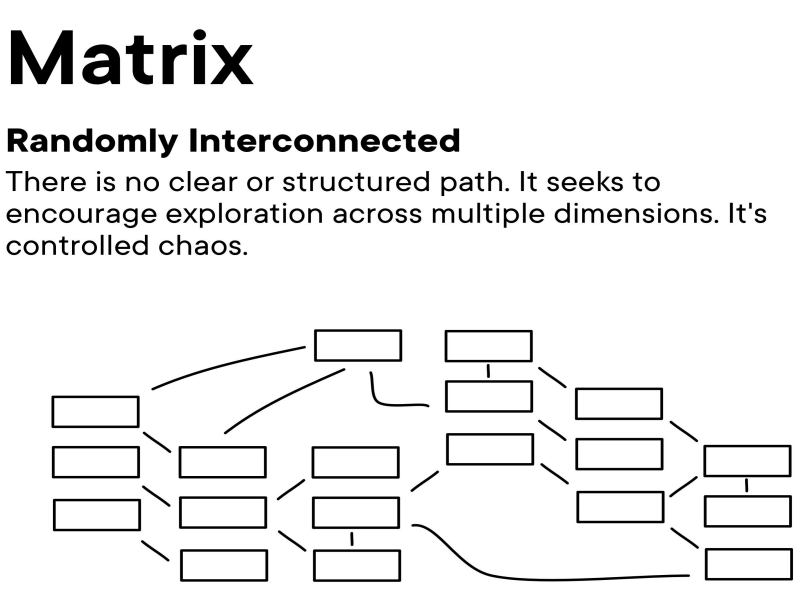

Figure 4: Matrix hierarchy

Database

Completely Dynamic

Think of sites like Pinterest or Google, and many types of marketplaces. Aside from the main navigation, content is generated based on input/output.

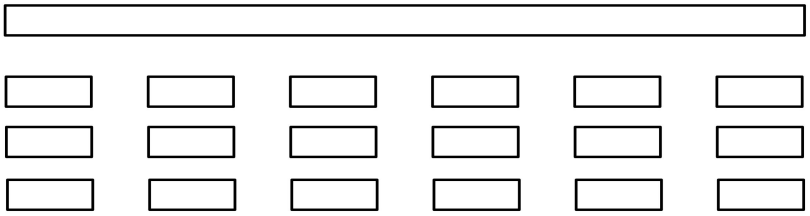

Figure 5: Database hierarchy

Flat + Matrix

Ex.) A blog or news website

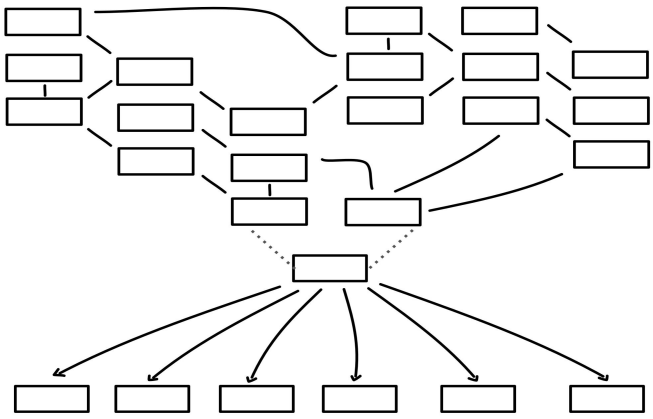

Figure 6: Flat + matrix hierarchy

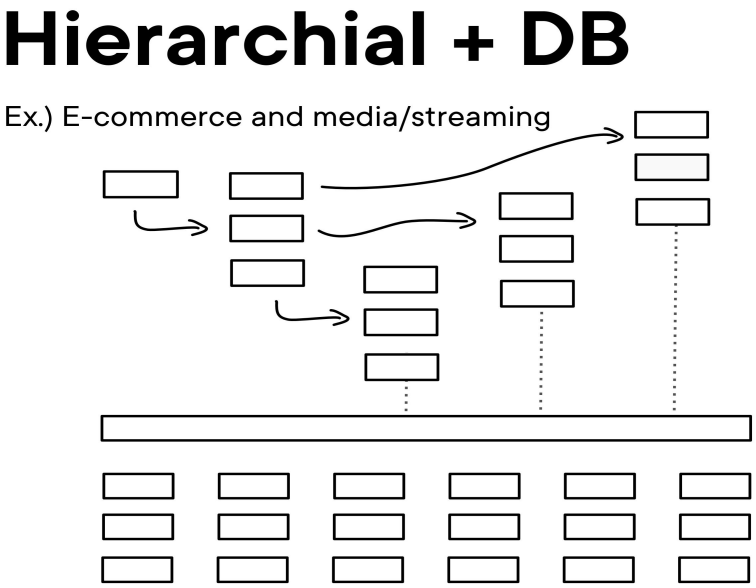

Figure 7: Hierarchical + DB hierarchy

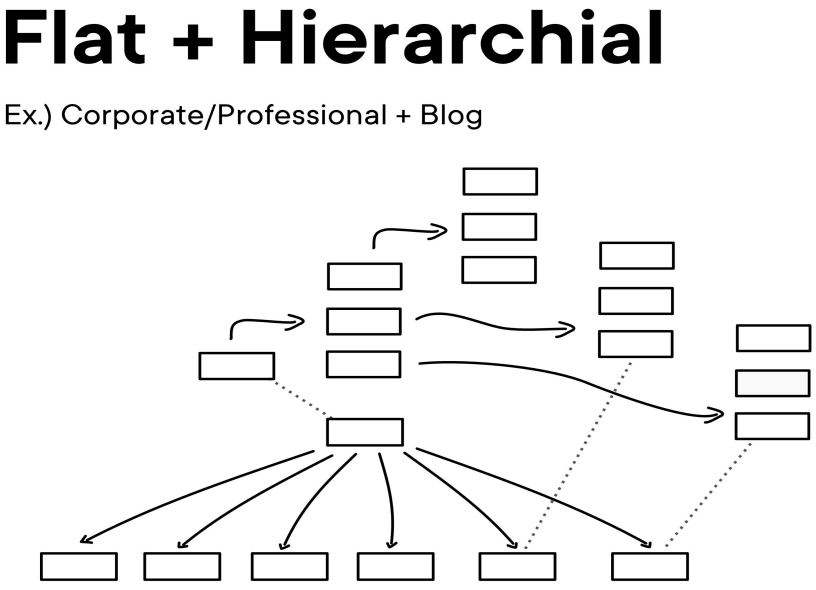

Figure 8: Flat + Hierarchical hierarchy

Sponsors: WordLift.io and Kalicube.com

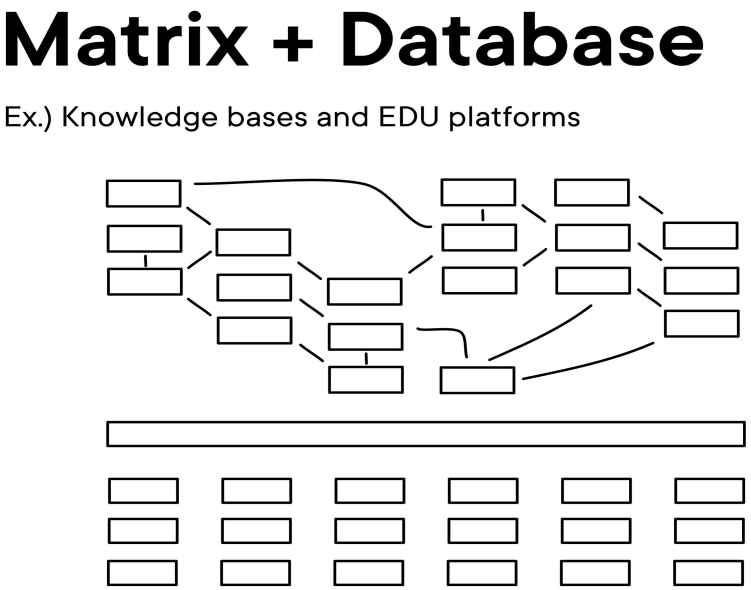

Figure 9: Matrix + DB hierarchy

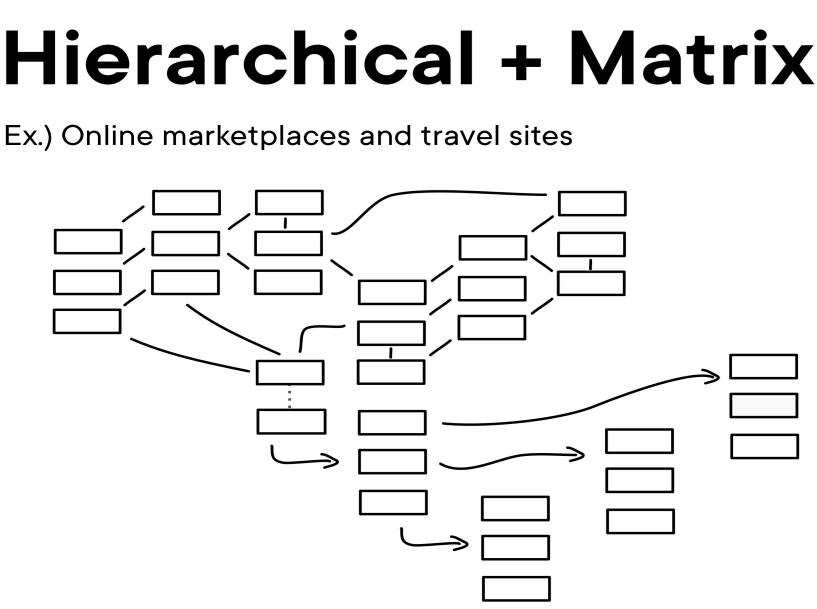

Figure 10: Hierarchical + Matrix hierarchy

Sponsors: WordLift.io and Kalicube.com

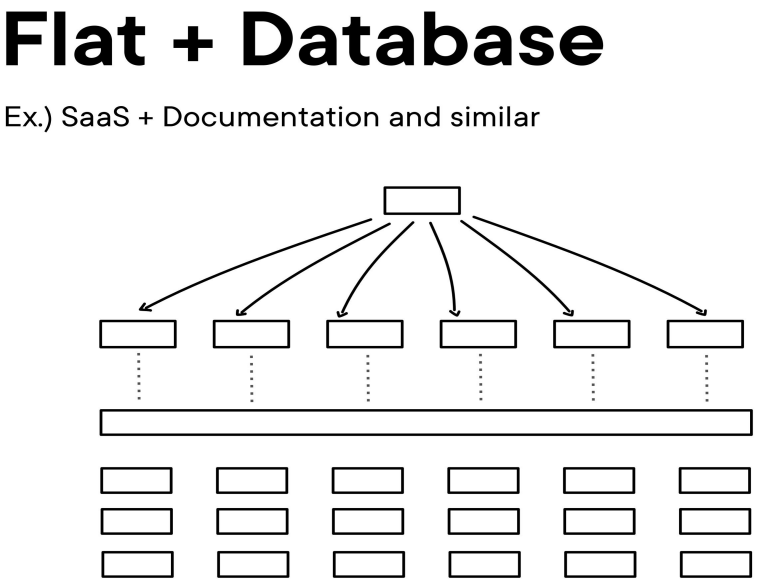

Figure 11: Flat + DB hierarchy

5 The Dataset

The dataset comprises information gathered through web crawling of English texts from the learning space directory of the Kalikube website. The primary elements of this dataset include URLs, raw HTML content, HTTP response codes, crawl depth, and inlinks associated with each URL, used to create the final website graph.

5.1 Uniform Resource Locators (URLs)

URLs are the fundamental elements of the dataset, serving as the unique identifiers for each web page crawled. Each URL represents a specific web resource, which could be a webpage, an image, a video, or other online content. Therefore, we made sure to exclude all links that do not represent webpages, to enable efficient processing of article-based content. The URLs in this dataset are stored as strings, which typically include components such as the protocol (e.g., HTTP or HTTPS), domain name, path, query parameters, and fragment identifiers. These components allow for the precise identification and retrieval of content from the web.

- Data type: string.
- Definition: the string uniquely identifies a resource on the web.

5.2 Raw HTML

The raw HTML data is the full content of the HTML source code retrieved from each URL. This data encapsulates the structure and content of a webpage, including the tags, metadata, text content, links, and embedded media. Raw HTML is critical for further analysis, such as parsing for specific elements, extracting metadata, and understanding the semantic structure of the webpage.

- Data type: text (string).
- Definition: the complete HTML rendered source code of a webpage as retrieved by the web crawler.

5.3 HTTP response code

The HTTP response code is a numerical status code sent by a web server in response to a request made by the web crawler. This code provides insight into the outcome of the HTTP request, indicating whether it was successful, resulted in a redirection, or encountered an error. Common response codes in the dataset include 200 (OK), 301 (Moved Permanently), 404 (Not Found), and 500 (Internal Server Error). These codes are integral to understanding the accessibility and status of each URL.

- Data type: integer.
- Definition: a three-digit code that indicates the outcome of the HTTP request for a specific URL.

5.4 Crawl depth

Crawl depth refers to the level of distance from the root URL or starting point of the crawl to the specific URL in question. It represents how many "clicks" away a particular page is from the homepage or the initially targeted URL. The crawl depth is a critical measure in understanding the structure of a website and how deeply the crawler has navigated within it. A depth of 0 indicates the root URL, a depth of 1 refers to a direct link from the root, and so on. Crawl depth is recorded as an integer in the dataset.

- Data type: integer.
- Definition: the number of links or levels between the root URL and the specific URL.

5.5 Inlinks

Known as inbound links, they refer to hyperlinks from other URLs that point to a particular webpage within the dataset. The count and source of inlinks are significant for assessing the importance and authority of a webpage, as pages with a higher number of inlinks are typically considered more valuable for SEO purposes. In the dataset, inlinks are often represented as a list of URLs that link to the specific page, providing context for the page's position within the web structure.

- Data type: list of strings.
- Definition: URLs of web pages that contain links pointing to the specific URL within the dataset.

The dataset in this research comprises a rich collection of web data, each element carefully structured and defined to facilitate comprehensive analysis. By detailing the characteristics, data types, and definitions of URLs, raw HTML, HTTP response codes, crawl depth, and inlinks, this section lays the groundwork for the subsequent analytical processes. Understanding the nature of this dataset is crucial for interpreting the results of the study and ensuring that the insights drawn are based on accurate and well-defined data components.

5.6 On the use of first-party data sources in SEO

When optimizing internal linking architecture for SEO, leveraging first-party data sources is crucial to understanding how users and search engines interact with a website. First-party data includes information collected directly from the website, such as crawl data, Google Search Console (GSC) metrics, Google Analytics (GA) reports, and log file data. However, the utility and applicability of these data sources vary significantly depending on the context and the technical expertise required to manage them.

5.6.1 Crawl data

We collaborated with Kalicube and the Faculty of Computer Science and Engineering to develop our logic using real-world data, which we plan to publish as-is. Using evergreen data, like links, was key since they remain stable despite content changes, unlike GSC and GA, which provide sampled data and may not reflect updates promptly.

Thus, crawl data is one of the most valuable sources for SEO analysis because it provides a comprehensive view of a website's structure and how search engines navigate it. This data includes all the pages that the crawler has visited, the links between them, and metadata such as response status codes. By analyzing crawl data, we can identify issues like broken links, orphaned pages, and inefficient internal linking structures that might hinder SEO performance. Crawl data should be prioritized because we have:

- Direct control: SEO professionals have direct control over crawl data, making it a reliable and actionable data source.
- Comprehensive view: crawl data offers a full picture of the website's internal structure, which is essential for optimizing internal links.
- No sampling issues: unlike some first-party and third-party tools, crawl data is typically not sampled, providing a complete dataset for analysis.

Given these advantages, crawl data is prioritized in this project as it is accessible, actionable, and relevant for all users.

5.6.2 Google Search Console (GSC)

Google Search Console is a valuable tool that provides insights into how Google's search engine interacts with a website. It offers metrics such as click-through rate (CTR), impressions, clicks, and queries that bring users to the site. However, it's important to note that GSC data is often sampled, meaning that the data shown might not represent the entire dataset. This sampling can introduce inaccuracies in metrics like CTR and impressions, making them less reliable for detailed SEO analysis. Excluding GSC metrics is good because we:

- Avoid sampling issues: GSC data is sampled, so it may not provide a complete picture, especially for websites with large volumes of traffic or numerous pages.
- Have data integrity: for a project focused on precise internal linking optimization, relying on unsampled, complete data is essential. Sampled data could lead to misleading conclusions.
- Enable applicability: to keep the project applicable to a semi-technical audience, we avoid delving into complexities and potential pitfalls of sampled data, focusing instead on more reliable sources.

While GSC can offer valuable insights, we exclude metrics like CTR, impressions, and clicks from the core analysis to maintain data integrity and ensure the project's findings are robust, complete, actionable, and testable.

5.6.3 Google Analytics (GA)

Google Analytics offers valuable insights into user behavior, but like GSC, its data can be sampled, especially for high-traffic sites, affecting metrics like session duration, bounce rate, and pageviews. Excluding GSC metrics is advisable for the same reasons mentioned earlier.

5.6.4 Log file data

Log file data provides a granular view of every request made to a server, including those by search engine bots. This data can offer insights into how search engines crawl a site, which pages are prioritized, and where potential issues like crawl errors might occur. However, accessing and analyzing log files can be challenging for several reasons:

- Technical complexity: managing and analyzing large log files requires technical expertise, particularly in handling big data. Most SEO professionals may not have the necessary skills to effectively utilize this data.
- Security and protocols: log files contain sensitive information, and accessing them often requires navigating strict security protocols. This can be a barrier for many users who lack knowledge of data security practices.
- Accessibility: due to the technical expertise required to handle log files, this data source is less accessible to the typical SEO professional.
- Applicability: by focusing on data sources that are more easily accessible and understandable to a broader audience, the project remains practical and applicable, even for semi-technical users.

5.6.5 Prioritization and practicality

The decision to prioritize certain data sources—such as crawl data—over others, like sampled GSC or GA metrics and complex log file data, is driven by the goal of making the project more accessible and hands-on to a broader audience. For semi-technical users, understanding and implementing complex data-driven strategies can be challenging. By focusing on data that is reliable, unsampled, and easier to manage, we ensure that the project remains practical for SEO professionals at all levels of technical expertise. This approach not only simplifies the process of internal linking optimization but also ensures that the insights derived are based on accurate and actionable data. By doing so, we enable users to implement effective SEO strategies without requiring deep technical knowledge or access to sophisticated data processing tools.

6 System Design and Requirements Engineering

A crucial aspect of implementing advanced SEO strategies is the ability to process and analyze large volumes of data efficiently. Developing a scalable, cost-effective crawler is essential for handling the complexities of large-scale internal linking optimization. A well-designed crawler must be capable of processing thousands of pages, to extract content and link data necessary for building a comprehensive internal linking strategy.

To make this feasible on a single machine or a cloud-based service like Google Colab Pro, several factors need to be considered:

- Efficiency: the crawler must be optimized to minimize resource usage while maximizing data extraction speed.
- Data processing: post-crawl data processing should be streamlined to quickly parse and analyze content, generating the necessary embeddings for further analysis. This step often involves cleaning and normalizing data to ensure consistency across different pages and websites.
- Memory management: given the scale of data, efficient memory management is critical. Techniques such as incremental processing and data streaming can help handle large datasets without overwhelming the system's resources.
- Cost: a key consideration for many businesses and SEO professionals is the cost of implementing such a system. Google Colab Pro offers a cost-effective solution by providing access to powerful GPUs and extended computing resources. However, careful management of these resources is necessary to avoid exceeding usage limits, especially when processing large datasets.
- JavaScript handling: many modern websites use JavaScript for dynamic content, which traditional crawlers may miss since they only parse static HTML. To address this, crawlers need headless browsing tools like Puppeteer or Selenium to execute JavaScript and fully render pages as users see them. This approach ensures all content, including dynamic elements, is extracted. Effective JavaScript handling also involves using smart wait strategies to manage page load times and avoid detection by anti-bot systems, enhancing the accuracy of internal linking strategies.

6.1 Design and implementation of the crawler

To meet these requirements, the crawler should be designed with scalability and efficiency at its core. Here's a high-level overview of the design requirements:

- Modular architecture: the crawler should be modular, with separate components for URL fetching, content extraction, and data processing. This allows for easier maintenance and scaling.
- Javascript handling: the crawler should be able to handle Javascript-heavy websites and adjust accordingly.
- OOP and SOLID principles: the code should be organized into classes and logical sections where possible but without overdoing it since the class and instances' creation affects speed performance.
- Data storage: we should store data in a database during crawling and resume the next iteration from where the previous crawl ended.

- Maximum crawling: 6 hours of data processing on a daily level is recommended.
- Polite crawling: increasing and decreasing speed based on server responses and different codes (e.g. 529) should be considered to adjust processing speed.
- Redirects: response codes like 301 and 302 need to be considered and the algorithm needs to process only the final URL instead of the first URL in the redirect loop.
- User string definition, throttling, and proxies: we should utilize these approaches to mimic human behavior and distribute efforts across different agents and IP addresses.
- Error handling and logging: these should be provided for better debugging.
- Caching the results: we should keep a list of visited URLs to dedupe URLs and avoid duplication.
- Processing subfolders or subdomains: starting small simplifies similarity calculations, as content within a folder is likely more similar than across folders. This approach also promotes polite crawling and reduces the risk of a DDoS attack by processing fewer links.
- Crawler configuration: several parameters control the crawler's behavior, such as request delays, the maximum number of pages per session, save intervals, retries, and redirects should be in place.
- Crawling strategy: the crawler uses BFS to explore all links on a page before moving deeper and DFS to follow a path as far as possible before backtracking. These strategies are implemented through different classes that inherit from a common abstract class.
- URL validation: to ensure relevance, the crawler validates URLs based on specific criteria, such as protocol type, file type exclusion, and URL patterns.
- Page data extraction: the crawler extracts content from each page, stores it in an SQLite database, and manages HTTP responses, including handling redirects and failed requests. It employs a range of user agents to mimic different browsers and reduce the chance of being blocked.
- Delay adjustment: the request delay is dynamically adjusted based on server response times to avoid overloading the server and potential blocking.
- Crawling process: the crawler iteratively extracts data, follows links, and tracks progress using a progress bar. It saves data periodically to prevent data loss during interruptions.

Sponsors: WordLift.io and Kalicube.com

- Database onteraction: the crawler uses an SQLite database for persistent storage, creating tables if necessary, and writing data in batches for efficiency.
- Scalability and performance: designed for scalability, the crawler can handle large websites efficiently by adjusting operational parameters.

7 Technical Stack and Setup

The project leverages a comprehensive and specialized technical stack to efficiently manage web crawling, data storage, and advanced machine learning tasks. This section outlines the tools and technologies employed, detailing their roles and how they are integrated into the overall workflow.

7.1 ScreamingFrog for web crawling

Screaming Frog SEO Spider is utilized for the initial crawling of specific directories within the target website. This tool is chosen for its robust ability to extract comprehensive data from web pages, including inlinks, which are critical for understanding the link structure of the site. The spider is configured to focus on certain directories, ensuring that the crawl is both targeted and efficient. The extracted data includes key attributes such as response codes and crawl depth, which are essential for further analysis.

7.2 Python on Google Colab Pro

To enhance the crawling experience and streamline data processing, Python scripts are executed on Google Colab Pro. Colab Pro is selected for its increased computational power, faster GPUs, and extended runtime, which are crucial for handling large-scale data processing tasks. Python is used to automate the crawling process, interact with APIs, and manage data extraction from Screaming Frog, ensuring that the process is both scalable and reproducible.

7.3 SQLite for data storage

SQLite serves as the primary database for storing the crawled data. The lightweight nature of SQLite, combined with its ability to handle complex queries, makes it an ideal choice for this project. The database stores various attributes of the crawled pages, including HTML content, response codes, and crawl depth, in a structured format. This setup facilitates efficient querying and retrieval of data for subsequent processing steps.

7.4 DuckDB for vector embeddings

DuckDB is employed for handling vector embeddings of the crawled pages. Given the need for high-performance analytics and the ability to work with large datasets in memory, DuckDB is a suitable choice. It supports the fast execution of complex analytical queries, which is crucial when dealing with vector embeddings and their manipulation. The embeddings are stored in DuckDB for efficient retrieval and analysis.

7.5 Embeddings creation

The embeddings capture the pages' semantic content and structural properties, which are essential for tasks like link prediction and clustering. Some embedding options we explored include:

- /nomic-ai/nomic-embed-text-v1.5
- jina-embeddings-v2-base-en
- xlm-roberta-longformer-base-4096
- Longformer
- LLM embeddings

Our dataset consists of English webpages with an average of over 4,200 words per page, exceeding the 4,096 token limits of roberta-longformer and standard longformer models. Using LLM embeddings for long articles is also costly and doesn't ensure quality. Therefore, we chose nomic and jinaai embeddings, which are open-source, fast, efficient, and tested in real-world setups.

7.6 CSV files for intermediate data storage

Intermediate results from various processing steps are stored in CSV files. This approach ensures that data can be easily shared, inspected, and processed in subsequent stages. CSV files provide a simple and portable format for storing and exchanging data, making them ideal for intermediate storage before the data is ingested into databases or used in machine learning models.

7.7 GraphSAGE for link prediction

For the link prediction task, the project utilizes GraphSAGE, a graph neural network framework designed for inductive learning on large graphs. GraphSAGE is particularly effective in generating node embeddings by sampling and aggregating features from a node's local neighborhood, which is crucial for predicting links in the web graph. The embeddings generated by GraphSAGE are used to predict missing links or to suggest potential new links, thereby enhancing the overall structure and connectivity of the site.

Sponsors: WordLift.io and Kalicube.com

7.8 Computer machine details (CPU, GPU, memory)

Google Drive with at least 50 MB of storage is recommended for storing 1,000 long-form articles, each around 8,000 words of cleaned content. Allow several hours for data crawling and at least 3.5 hours for content vectorization.

7.9 Subnet verification and IP whitelisting

Effective crawling operations require close collaboration between the entity that owns a website and the parties engaged in crawling it. Two critical components in establishing this collaboration are subnet verification and IP whitelisting, both of which can significantly enhance the efficiency and security of crawling processes. For those looking to conduct advanced SEO analysis and implement intelligent internal linking, it's essential to adhere to subnet verification and IP whitelisting protocols.

7.9.1 Subnet verification

Subnet verification is a crucial step that involves confirming the legitimacy of the IP ranges from which crawlers operate. Website owners can use this process to ensure that the traffic directed at their sites originates from authorized crawlers, reducing the risk of bot attacks and unauthorized data scraping. For crawlers, verifying their subnet with the website owner builds trust and ensures that their activities are recognized and respected, thereby avoiding unnecessary blocks or throttling.

From a technical perspective, subnet verification involves the crawler providing the website owner with a list of IP ranges they operate within. The website owner then verifies these IPs to confirm they fall within expected ranges and are associated with the organization performing the crawling. This step can also involve mutual confirmation through digital certificates or other authentication mechanisms. Once verified, these subnets can be treated differently in the web server's access control policies, allowing for smoother, uninterrupted crawling.

7.9.2 IP whitelisting

IP whitelisting complements subnet verification by explicitly allowing traffic from specific IP addresses or ranges to access a website. This process is beneficial for both website owners and crawlers. For the website owner, IP whitelisting offers granular control over who can access their site, protecting against malicious activities and conserving server resources. It ensures that only recognized, legitimate crawlers are allowed to conduct operations on the site, thereby safeguarding data integrity and user experience. For crawlers, having their IPs whitelisted is essential for efficient crawling. It prevents their access from being blocked by security measures such as firewalls or intrusion prevention systems, which may otherwise flag their activity as suspicious. Whitelisting allows crawlers to perform their tasks without frequent interruptions, leading to more comprehensive and timely data collection.

7.9.3 Collaboration for efficient crawling

The effectiveness of subnet verification and IP whitelisting relies on strong communication between the website owner and the crawler. This begins with transparency about the crawler's goals, scope, and infrastructure, while website owners must share their access policies, server capacity, and any crawling limits. Establishing dedicated communication channels, such as real-time alerts for IP changes and regular reviews of whitelisted IPs, builds trust and ensures stability. This collaboration allows website owners to safeguard resources while enabling uninterrupted crawling, leading to more efficient and sustainable operations.

8 Integration and Workflow

The workflow begins with Screaming Frog crawling the specified directories and extracting relevant data. This data is then processed using Python scripts on Google Colab Pro, where it is stored in SQLite for efficient management. Embeddings for each page are generated using jinaai/jina-embeddings-v2-base-en with DuckDB handling the storage and retrieval of these embeddings. Intermediate results are stored in CSV files, ensuring flexibility in the workflow. Finally, GraphSAGE is applied for the link prediction task, leveraging the embeddings stored in DuckDB to improve the link structure of the site.

8.1 Data collection and processing workflow tips and tricks

When undertaking a data-intensive project, such as building and deploying a web crawler for large-scale data collection, it's crucial to approach the process with a set of best practices in mind. These tips and tricks aim to guide you through the entire workflow, ensuring efficiency, reliability, and data integrity while avoiding common pitfalls.

8.2 On overloading the target website

When developing a web crawler, it's crucial to avoid overloading the target website. Too many requests in a short time can strain the server, increase costs, cause IP bans, or crash the site, disrupting data collection and harming the website. Testing the crawler on different sites to distribute server load is recommended before launching.

8.3 Patience during data collection and processing

Data collection, especially when involving web scraping, can be time-consuming. The process of retrieving data, cleaning it, and processing it into a usable format can take a significant amount of time, depending on the volume and complexity of the data. Tips:

- Plan Your Time: allocate ample time for data collection and processing. Rushing through these steps can lead to errors, incomplete datasets, or flawed analyses.
- Monitor Progress: regularly monitor your data collection process to ensure it is running smoothly. Tools like logging and alert systems can help you keep track crawler's performance.
- Use Batch Processing: instead of trying to process all data at once, break the task into smaller batches. This approach can make the process more manageable and reduce the likelihood of errors. Patience in this phase ensures that your data is both comprehensive and accurate, forming a solid foundation for subsequent analysis.

8.4 On saving full HTML files for additional Processing

During the data collection process, it's advisable to save the full HTML files of the web pages you are crawling. This practice can be extremely beneficial for several reasons:

- Data integrity: having the original HTML files allows you to revisit and reprocess the data if necessary. This is particularly useful if you later discover errors in your data extraction methods.
- Flexibility: storing the raw HTML gives you the flexibility to extract additional information that you may not have anticipated needing initially.
- Comparison: a way to compare raw data with the cleaned data to ensure that the cleaning process has not inadvertently altered or removed important information is advised.
- Organizing storage: consider creating a systematic storage structure (e.g., organized by date or category) to store your HTML files. This will make it easier to locate specific files later.
- Using version control: consider using version control systems like Git to track changes in your HTML files, especially if you are iteratively improving your scraping scripts.
- Maintaining raw and cleaned datasets: ensure consistent backtracking and validating of your data processing methods.

8.5 Ensuring adequate cloud storage

The data collected from web scraping can be quite large, especially when saving full HTML files alongside cleaned data. Therefore, it is essential to ensure sufficient cloud storage, such as Google Drive, to store your datasets and databases. Some tips include:

- Estimating storage needs: before starting the project, calculate the required storage based on the expected data volume. This helps prevent running out of space during the project.
- Utilizing Google Drive: Google Drive offers reliable cloud storage with easy access and sharing capabilities. Consider upgrading to a larger storage plan if needed.
- Automating backups: setting up automatic backups of your data to Google Drive ensures that the data is safe and recoverable in case of local storage failures.
- Proper storage management: safeguarding data ensures easy access for analysis and further processing.

8.5.1 Runtime Verification in Google Colab

When running data collection and processing scripts, especially in environments like Google Colab, it's essential to be available to handle any runtime issues that may arise. While Google Colab is a powerful tool for running scripts, it may require your attention for runtime verification or to resolve unexpected issues:

- Stay close to the computer: during long-running scripts, it is advised to be near your computer to respond to any prompts or issues. This will help prevent disruptions in the data collection process.
- Notifications usage: setting up notifications (e.g., via email or messaging apps) that alert the dev/marketer when the script completes or encounters an error.
- Checkpointing: checkpoints in the code are needed to periodically save progress, avoiding data loss.

Preparation for handling runtime verifications and other issues ensures that the data collection process is as smooth and uninterrupted as possible. Adhering to these guidelines will assist in effectively navigating the complexities of web data collection and processing. By carefully considering the impact on target websites, maintaining a patient and methodical approach, ensuring diligent data storage, and being prepared to address runtime challenges, we can guarantee workflow optimization for greater efficiency and accuracy.

9 Vector Embeddings Analysis

The use of the JinaAI embeddings model (v2-base-en) from Hugging Face provides an efficient solution for handling large English text datasets. This model is particularly suited for processing extensive documents, as it generates high-quality sentence embeddings that capture semantic meaning effectively. Its ability to process large volumes of text while maintaining accuracy makes it an optimal choice for applications requiring

robust text representations, such as semantic search, document clustering, and text classification.

One notable feature of JinaAI embeddings is their scalability when working with long texts. The model can efficiently process substantial text input without compromising performance, making it well-suited for large-scale text analysis. Furthermore, its architecture ensures that the embeddings retain the contextual nuances of the language, which is crucial for tasks like semantic similarity and document comparison.

To improve performance further, especially for tasks such as fast clustering and large-scale semantic search, embedding quantization can be applied. Quantizing embeddings, as demonstrated by the sentence-transformers framework, reduces the size of embeddings from 32-bit floats to lower-bit representations, such as 8-bit or binary embeddings. This compression significantly speeds up search and retrieval processes without a substantial loss of accuracy. Quantization also reduces the memory footprint, allowing more embeddings to be stored in memory and enabling faster operations during real-time querying.

By combining JinaAI embeddings with quantization techniques, it is possible to optimize both the quality of the embeddings and the efficiency of operations like clustering and search. This approach is particularly beneficial when processing large datasets, as it allows for rapid, memory-efficient execution without sacrificing the depth of semantic understanding.

10 GraphSAGE for Link Prediction

Link prediction in a graph involves predicting whether a link (edge) between two nodes should exist. Using GraphSAGE for link prediction is explained in the following subsections.

10.1 Node embedding generation

In this section, we outline a comprehensive approach for implementing link prediction using GraphSAGE (Graph Sample and Aggregated) solution for link prediction. GraphSAGE helps identify new linking opportunities within a graph network by leveraging both the structural and semantic information of nodes. Each node represents a web page or entity and is characterized by several features, such as a vector of its content, crawl depth, and response code. Here's an overview of the implementation:

- Feature aggregation: GraphSAGE learns node embeddings by aggregating the features of each node's neighbors. In this case, each node has features like content vectors (representing the textual or contextual information of the web page), crawl depth (indicating how far the node is from the starting point of the crawl), and response codes (showing the status of the web page). GraphSAGE incorporates these features and those of neighboring nodes to generate an embedding for each node that captures both local structure and node-specific features.

- Node embeddings for similarity: the embeddings generated by GraphSAGE represent nodes in a high-dimensional space where similar nodes (both in terms of graph structure and semantic content) are positioned closer to one another. This allows the model to identify nodes (web pages) that share similar characteristics, making them suitable candidates for linking. For example, a node with similar content and a crawl depth close to another could be a potential linking opportunity.

- Link prediction: once nodes are embedded, a link prediction algorithm uses these embeddings to determine the likelihood of a link (hyperlink or connection) between two nodes. The model assesses both the graph-based proximity (how close the nodes are in the network structure) and the semantic similarity (content similarity based on the vector representation) to predict new links. This enables the discovery of linking opportunities that make sense from a structural perspective (e.g., nodes that should be connected in a website hierarchy) and from a semantic viewpoint (e.g., content-relevant linking).

- Finding relevant candidates: by evaluating the graph's structure and the content of each node, GraphSAGE suggests linking opportunities that align with the website's or the graph's intent. For instance, a web page about "machine learning" may find linking opportunities with other pages that also cover similar content but are currently not connected. Additionally, the model considers other features like crawl depth and response codes to ensure the new links are practical and won't point to irrelevant or inaccessible pages.

11 The Final User Interface of the Solution

The final user interface (UI) is designed to simplify the discovery of new linking candidates based on a website's semantic and structural analysis. Users input a sample link, which the system analyzes for content relevance and position within the site's graph to suggest optimal linking opportunities.

Key UI elements include input fields for the sample link, adjustable parameters, and visual indicators for the analysis status. The results section will present proposed links intuitively for easy review and selection.

The design prioritizes a clean, user-friendly experience, helping users quickly establish meaningful internal links to improve navigation and SEO. This enables webmasters and SEO professionals to choose their preferred hierarchy while using our solution to easily find good linking candidates, eliminating manual effort.

```
Enter a URL to get link recommendations: https://kalicube.com/learning-spaces/knowledge-nuggets/brand-s

Top 10 recommended links:
1. Destination URL: https://kalicube.com/learning-spaces/faq/knowledge-panels/, Probability: 0.9718
2. Destination URL: https://kalicube.com/learning-spaces/faq/seo-glossary/the-12-concrete-results-of-im
3. Destination URL: https://kalicube.com/learning-spaces/knowledge-graph-update/, Probability: 0.9663
4. Destination URL: https://kalicube.com/learning-spaces/faq/brand-serps/, Probability: 0.9653
5. Destination URL: https://kalicube.com/learning-spaces/faq/, Probability: 0.9645
6. Destination URL: https://kalicube.com/learning-spaces/knowledge-graph-update/timeline/, Probability:
7. Destination URL: https://kalicube.com/learning-spaces/faq/brand-serps/what-are-the-kpis-for-the-kali
8. Destination URL: https://kalicube.com/learning-spaces/faq/seo-glossary/, Probability: 0.9567
9. Destination URL: https://kalicube.com/learning-spaces/faq/brand-serps/how-does-the-kalicube-process-
10. Destination URL: https://kalicube.com/learning-spaces/knowledge-nuggets/, Probability: 0.9561
```

Figure 12: The link recommendation solution

12 Testing and evaluating results

At this stage, we are conducting rigorous testing together with WordLift to ensure the robustness of the solution. The testing phase involves implementing the solution in a real-world environment, but in a controlled manner, allowing us to carefully monitor performance, identify potential issues, and refine the approach. This process is crucial to validating the effectiveness of the solution in diverse scenarios and ensuring its scalability and adaptability to various use cases.

By systematically experimenting with different configurations and settings, we aim to optimize both the technical and functional aspects of the solution. Feedback from real-world use cases will help us fine-tune the algorithms, improve user experience, and ensure that the solution meets the practical needs of its intended audience.

As this project is dynamic and evolving, we will share comprehensive results once it's fully complete. We anticipate that the insights gained during this testing phase will lead to further refinements, and we will publish detailed evaluations and findings shortly as part of our ongoing efforts to enhance the solution's impact.

In the meantime, we invite the technical marketing community to try it and share feedback. This is crucial to our product-driven, agile approach, as collaborative knowledge sharing is key to our team's success.

13 Final Words

Thank you for being with us so far, dear reader.
It's your turn to act now. This research has been quite a journey, and I sincerely hope it was worthwhile, providing you with new insights and valuable perspectives. Conducting this study was no easy task, but with the support of an innovative, research-driven team, *anything is possible*. Your engagement with this work means the world to us, and I hope it has sparked ideas, questions, or perhaps even a new path forward for your own endeavors.

As we close this chapter, I encourage you to take action—whether that's applying

the findings in your own projects, pushing the boundaries of the research presented, or simply reflecting on what you've learned. Progress, after all, comes from curiosity and the courage to explore uncharted territory.

This research is part of a larger, ongoing conversation, and I would love for you to stay connected as we continue to innovate and grow. Your thoughts, feedback, and collaboration are always welcome, as they help shape the future of this dynamic field. Let's continue this journey together—because, in the end, it's the exchange of ideas and shared passion that makes real impact. Thank you once again for your time, your curiosity, and your commitment to learning.

Sponsors: WordLift.io and Kalicube.com

References

[1] P. Suzart, "Nailing internal linking with graph theory — by paulo suzart — omio engineering — medium." https://medium.com/omio-engineering/nailing-seo-internal-linking-with-graph-theory-2c45544a024d, August 2020. (Accessed on 09/04/2024).

[2] K. Indig, "The best internal linking structure depends on your business model." https://www.growth-memo.com/p/the-best-internal-linking-structure-depends-on-your-business-model, February 2019. (Accessed on 09/04/2024).

[3] E. Gjorgjevska and V. Izzo, "Dynamic internal links in seo: your superhero in the generative ai era." https://wordlift.io/blog/en/dynamic-internal-links-in-seo/, July 2024. (Accessed on 09/04/2024).

[4] M. Ciffone, "Linkedin post." https://www.linkedin.com/posts/mike-ciffone_common-website-architectures-and-powerful-ugcPost-7235710529627963392-FCSn/, September 2024. (Accessed on 09/05/2024).

[5] E. Gjorgjevska, "Mastering topic clusters in seo - wordlift blog." https://wordlift.io/blog/en/master-topical-seo/, January 2024. (Accessed on 09/05/2024).

Sponsors: WordLift.io and Kalicube.com