Performance and Scalability of Frontends in Processing Vectorized Text Data: A Comparison Between Traditional Monolithic and Modular Frontends

Luiz Felipe Cirqueira dos Santos[a], Alberto Luciano de Souza Bastos[b], Shexmo Richarlison Ribeiro dos Santos[c], Mariano Florencio Mendonça[d], Marcus Vinicius Santana Silva[e], Marcos Cesar Barbosa dos Santos[f], Marcos Venicius Santos[g], Marckson Fábio da Silva Santos[h], Fabio Gomes Rocha[i] and Michel S. Soares[j]

Postgraduate Program in Computer Science, PROCC/UFS, Federal University of Sergipe, Av. Mal. Cândido Rondon, Rosa Elze, 1861, São Cristóvão, Brazil

Keywords: Monolithic Frontend, Modular Frontend, Vectorized Data, Vectorized Textual Data, Vectorization.

Abstract: Text vectorization is essential for information search and retrieval systems, requiring efficient front-end architectures. This study compares monolithic and modular approaches for vectorized data processing, evaluating performance and scalability. Using Dynamic Capacity Theory as a basis, we developed two functionally equivalent implementations and evaluated them quantitatively using the Lighthouse tool (Chrome DevTools) in Timespan and Snapshot modes. Results show clear tradeoffs: while the monolithic architecture presents better initial performance, the modular solution shows superior scalability for large data volumes. Our conclusions provide practical guidelines for architectural selection based on specific vector processing requirements, contributing to the optimization of high-demand web systems.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Humans easily understand natural language and facilitate clear communication, yet it presents inherent challenges for computational search and information retrieval systems. One promising solution to this challenge is data vectorization, which enables text transformation into mathematical representations that support semantic analysis and retrieval of information based on contextual similarity rather than keyword matching (Supriyono et al., 2024).

In vector space models, all words are represented in terms of contextual relationships, adhering to the distributional hypothesis that words used in similar contexts tend to have similar meanings (Aka Uymaz and Kumova Metin, 2022).

Vectorization improves the precision and relevance of retrieval processes by allowing systems to return results aligned with the user's intent, not just literal matches (Mikolov et al., 2013; Gao et al., 2024).

In addition to enhancing search capabilities, vectorized data models and frontend modularity can be understood through the lens of Dynamic Capabilities Theory (Xiao et al., 2020). This theory posits that systems must be able to reconfigure themselves to respond effectively to change—particularly relevant in environments involving collaborative development and continuous deployment. From this perspective, modular frontend architectures provide a foundation for agility and evolution through independent, maintainable components.

[a] https://orcid.org/0000-0003-4538-5410
[b] https://orcid.org/0009-0002-3911-9757
[c] https://orcid.org/0000-0003-0287-8055
[d] https://orcid.org/0000-0003-0732-3980
[e] https://orcid.org/0000-0003-4538-5410
[f] https://orcid.org/0000-0002-7929-3904
[g] https://orcid.org/0009-0006-1645-6127
[h] https://orcid.org/0009-0001-6479-1900
[i] https://orcid.org/0000-0002-0512-5406
[j] https://orcid.org/0000-0002-7193-5087

Cirqueira dos Santos, L. F., Bastos, A. L. S., Ribeiro dos Santos, S. R., Mendonça, M. F., Silva, M. V. S., Barbosa dos Santos, M. C., Santos, M. V., Santos, M. F. S., Rocha, F. G. and Soares, M. S.
Performance and Scalability of Frontends in Processing Vectorized Text Data: A Comparison Between Traditional Monolithic and Modular Frontends.
DOI: 10.5220/0013655200003985
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 21st International Conference on Web Information Systems and Technologies (WEBIST 2025)*, pages 105-112
ISBN: 978-989-758-772-6; ISSN: 2184-3252
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

Therefore, software systems that handle vectorized data must prioritize flexibility, adaptability, and transparency-especially on the client side-to support seamless integration of new features and ensure efficient operation in distributed teams (Simões et al., 2024). Maintaining cohesion and code quality becomes increasingly difficult as web applications grow more complex and rich in functionality, particularly in large development teams (Taibi and Mezzalira, 2022).

Although modern frontend frameworks offer various implementation options, including Single Page Applications (SPA), Server-Side Rendering (SSR), and static HTML, many still follow a monolithic structure. While familiar and initially efficient, this architecture results in tight coupling and hinders scalability, especially in collaborative scenarios that demand quick feature deployment and integration of advanced techniques such as vectorization (Peltonen et al., 2021).

To overcome these limitations, modular architectures—often supported by micro frontend strategies—offer better component separation, easier maintenance, and improved scalability (Perlin et al., 2023; Simões et al., 2024). These architectures support technological heterogeneity and allow system parts to evolve independently. This is critical for applications adapting to fast-changing demands and large-scale data integration (Wang et al., 2020).

This paper compares two frontend architectures-monolithic and modular-in vectorized text data processing. Both implementations are functionally equivalent and integrated with the same dataset. Performance, accessibility, scalability, and maintainability are quantitatively evaluated using Chrome Lighthouse metrics (Snapshot and Timespan modes). The study investigates how architectural choices affect resource usage, frontend responsiveness, and long-term development flexibility.

The findings highlight significant trade-offs between the two approaches: while monolithic architectures offer better initial performance, modular designs exhibit greater alignment with modern development practices, enhanced maintainability, and better scalability in long-term applications.

The remainder of this article is organized as follows. Section 2 provides the context of this work. Section 3 discusses related work. Section 4 covers the methodology employed in this study. Section 5 describes and presents the results, and Section 6 discusses the results. Section 7 contains the final considerations.

2 THEORETICAL FRAMEWORK

This section presents the fundamental concepts of this study, including the theory of dynamic capabilities, data vectorization techniques, and frontend and micro-frontend architectures.

According to the theory of dynamic capabilities, a company's competitive advantage stems from its ability to apply these capabilities in volatile environments effectively (Xiao et al., 2020). Dynamic capabilities are defined as the organization's capacity to integrate, build, and reconfigure internal and external competencies to enhance operational performance and adapt to rapidly changing conditions (Xiao et al., 2020). Scholars view dynamic capabilities as a bridge between Business Data Analytics Capabilities (BDAC) and enterprise-level outcomes, offering essential insights into how analytics influence innovation and agility (Wamba et al., 2017).

As an extension of the resource-based view, the dynamic capabilities perspective emphasizes the deliberate transformation of tangible and intangible resources and operational routines (Mikalef et al., 2019; Schilke et al., 2018). Consequently, strategies that foster the continuous development of dynamic capabilities tend to outperform others in the software industry (Helfat and Winter, 2011; Lo and Leidner, 2018). Identifying and nurturing these capabilities allows organizations to respond swiftly to technological shifts and evolving market demands, thus maintaining relevance and competitiveness (Chen, 2023).

As highlighted by Occhipinti et al. (Occhipinti et al., 2022), the evolution of text data vectorization illustrates the need for adaptability and innovation in increasingly complex data environments. Vectorization techniques have progressed from traditional statistical methods such as Bag-of-Words (BoW), n-grams, and Term Frequency-Inverse Document Frequency (TF-IDF), which, despite their robustness, ignore word order and context, and often suffer from sparsity issues (Occhipinti et al., 2022).

Several techniques are available to convert textual data into numerical form. BoW, as described by Mariyam et al. (Mariyam et al., 2021), represents documents as word frequency vectors, disregarding the sequence of terms. TF-IDF, in turn, weighs terms based on their frequency in a document and their rarity in the corpus, helping to identify more informative words for classification tasks (Occhipinti et al., 2022; Lewis et al.,). Word2Vec offers a more advanced alternative by generating dense embeddings that capture semantic relationships based on context.

Each word is represented by a vector that reflects its meaning based on neighboring words in the corpus.

More recently, models such as BERT (Bidirectional Encoder Representations from Transformers) have been introduced to generate contextual embeddings, in which the vector representation of a word varies according to its usage in the sentence (Devlin et al., 2018; Mikolov et al., 2013). These models overcome previous limitations by incorporating bidirectional context, significantly improving the accuracy of tasks such as information retrieval, classification, and semantic search. Such techniques also have practical implications for frontend architectures. By enabling similarity comparisons directly on the client side, they reduce the dependency on backend requests, which improves responsiveness and efficiency, particularly when handling large volumes of semantically rich data (Park et al., 2022).

Architecturally, a traditional monolithic frontend consists of a unified application in which all services and components are bundled together within a single codebase. This design is commonly structured in horizontal layers and is characterized by strong interdependence between modules (Krishnamurthy, 2019; Peltonen et al., 2021). Although initially straightforward to implement, monolithic architectures hinder scalability and flexibility, especially in collaborative environments. In contrast, modular architectures—especially those using micro frontend principles—embrace component reuse, decoupling, and separation of concerns. As Ian (Ian, 2019) points out, this approach allows software to be built from smaller, manageable units that encapsulate complexity behind well-defined interfaces. Vescovi et al. (Vescovi et al., 2023) reinforce that modularity is key to enabling rapid development, scalability, reasoning about system behavior, and integration of new features without compromising the stability of the whole system.

3 RELATED WORK

Recent studies have investigated the interplay between frontend architectures, vectorized data processing, and modern information retrieval techniques, especially in systems that leverage semantic search and large language models (LLMs). This section organizes the related contributions into three thematic areas: modular frontend architectures, text vectorization and RAG systems, and practical transformations from monolithic to modular architectures.

Micro frontend architectures have gained attention as a strategy for improving modularity, maintainability, and scalability in complex web systems. Peltonen et al. (Peltonen et al., 2021) and Taibi and Mezzalira (Taibi and Mezzalira, 2022) highlight how micro frontends support information presentation through isolated, independently deployable modules—an essential capability in large teams or evolving applications. Simões et al. (Simões et al., 2024) further argue that this architectural style facilitates integration between distinct information retrieval and presentation components without compromising cohesion. Bian et al. (Bian et al., 2022) reinforce this by observing that traditional monolithic frontend development becomes increasingly unfeasible in large-scale applications maintained by multiple teams. They advocate for a component-based model inspired by microservices to circumvent architectural rigidity and enable more flexible workflows. Männistö et al. (Männistö et al., 2023) present a case study from the software company Visma, documenting its transition from a monolithic to a micro frontend architecture. Their findings suggest that, even in small organizations, modularization improves client-specific configurability and reduces implementation costs. Notably, they emphasize that native web standards—rather than heavyweight JavaScript frameworks—can suffice to achieve modularity effectively.

Another line of research focuses on vectorized data processing and semantic retrieval, particularly in Retrieval-Augmented Generation (RAG). Occhipinti et al. (Occhipinti et al., 2022) and Aka Uymaz and Kumova Metin (Aka Uymaz and Kumova Metin, 2022) demonstrate how classic and modern text vectorization techniques (e.g., TF-IDF, Word2Vec, and BERT) improve classification and sentiment analysis by leveraging the semantic relationships between terms. Wang et al. (Wang et al., 2020) and Gao et al. (Gao et al., 2024) explore how RAG architectures can enhance LLM outputs by injecting context from external vectorized sources. Lewis et al. (Lewis et al., 2020) describe the full RAG pipeline: receiving a user query, retrieving semantically related documents from a knowledge base, and feeding that context into an LLM to generate more accurate responses. Wang et al. (Wang et al., 2024) further extend this by detailing practical strategies for configuring RAG systems, including document chunking, embedding model selection, and ranking procedures. These optimizations significantly improve semantic search

workflows' relevance and latency, demonstrating the benefits of vector databases and dense retrieval for knowledge-intensive applications.

Few studies explicitly address the role of frontend architecture in the performance and evolution of RAG-enabled systems. Integrating modular interfaces with vectorized search layers is typically assumed, but not evaluated directly. While Simões et al. (Simões et al., 2024) suggest that modular frontends support vector-based retrieval, empirical validation of this claim is limited in the literature. This study seeks to fill this gap by directly comparing monolithic and modular frontend architectures in a controlled environment, integrated with a vectorized legislative dataset. Unlike previous work focusing primarily on backend architectures or model-level optimization, our approach emphasizes client-side performance, scalability, and maintainability, using objective metrics such as latency, memory usage, and Lighthouse scores.

4 METHODOLOGY

The methodology of this study was applied to a sample of legislation from the state of Sergipe, with data obtained from the legislation website of the Executive Branch at https://legislacao.se.gov.br/. These data were grouped into a JSON file containing the following keys: `file_name`, `file_extension`, `norm_type`, `publication_year`, and `text_string`, where the `text_string` key stores the full text of the norm. The `text_string` key was used in the text vectorization process.

To generate the embeddings, the Sentence Transformers module for Python was used, with the SentenceTransformerEmbeddingFunction function and the `all-MiniLM-L6-v2` model. This process generated arrays of values representing the semantic similarity between texts. The original data in JSON were augmented with the generated embeddings for each norm and then stored locally in a Chroma DB database.

With the data prepared, a comparative performance and scalability analysis was conducted using two frontend architectures: monolithic and modular. Both frontends consume the vectorized database. To evaluate the applications' performance and quality, tests were performed using Lighthouse in the development environment, collecting metrics such as initial load time, query latency, memory usage, throughput, and quality indicators like accessibility and best practices.

Two application versions are considered for comparison with the architectures: a traditional monolithic version and a modular monolithic version. Both were integrated with JSON vectorized data, which dynamically displayed the information on the screen. Implemented a search functionality to query the data filtering. The goal was to evaluate the differences in code organization, maintainability, and performance between the two structures, ensuring that both maintained the same basic functionality for a fair comparison of the metrics.

The Lighthouse metrics were analyzed during query execution to ensure that the vectorized data represented semantically similar information. Additionally, a statistical analysis of the vectors was performed, calculating the mean and standard deviation to verify data distribution.

The vectorization and data validation process was carefully documented, including detailed steps, parameters used, and specific tools applied. This documentation aims to ensure the transparency of the process and the potential for reproducibility by other researchers. The methodological approach adopted seeks to provide a solid foundation for comparisons and data validation, establishing a clear and reliable structure for the study.

5 RESULTS

This section presents the results from the tests conducted using the Lighthouse tool in Snapshot and Timespan modes. The objective was to evaluate the metrics of Performance, Accessibility, Best Practices, and SEO (Search Engine Optimization) in Snapshot mode and Performance and Best Practices in Timespan mode. The following sections detail the results, comparing the two approaches and discussing the most relevant points.

5.1 Lighthouse Snapshot Test

Lighthouse Snapshot tests were used to analyze various aspects of the quality and performance of both frontend architectures. This mode captures the application's state at a specific moment, providing valuable data for optimization and accessibility in four main categories: Performance, Accessibility, Best Practices, and SEO.

The performance analysis results showed significant differences between the two architectures. As indicated in Table 1, the monolithic frontend stood out in terms of optimization, with faster initial loading times and improved metrics such as First Contentful Paint and Speed Index. This

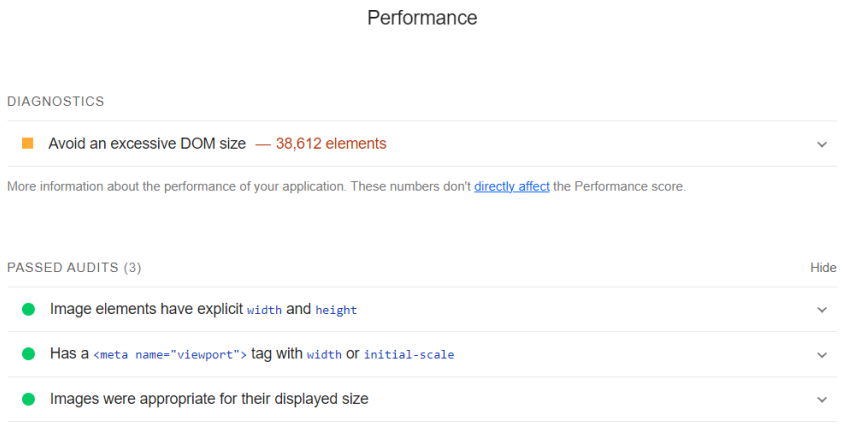

Figure 1: Snapshot Modular Performance.

Table 1: Snapshot Monolithic Performance.

Criterion	Status
Avoids excessive DOM	Passed
Images with explicit width and height	Passed
Has viewport meta with width and initial-scale	Passed
Properly sized images	Passed

indicates that the monolithic frontend allowed more efficient resource delivery, providing a smoother user experience.

As shown in Figure 1, the modular frontend presented areas for improvement, especially regarding DOM (Document Object Model) size and viewport settings, which showed inefficiencies and negatively impacted loading times. These inefficiencies may have contributed to lower performance compared to the monolithic version.

In the Accessibility category, both frontends implemented good practices, but with noticeable differences. The monolithic frontend showed fewer errors, especially in aspects such as color contrast, document title presence, and 'lang' attributes in the HTML element, ensuring better accessibility for users with disabilities.

The modular frontend encountered DevTools protocol timeout issues, compromising the complete data collection related to accessibility. Many tests related to ARIA attributes failed due to protocol timeout problems, indicating that adjustments are needed to improve the response of interactive components and ensure a comprehensive accessibility analysis.

In evaluating best practices, the modular frontend consistently adhered more to modern security and development recommendations. Fewer issues related to security, such as outdated libraries and lack of HTTPS protocols, were identified.

In contrast, the monolithic frontend displayed more warnings related to vulnerabilities, suggesting that adjustments are necessary to ensure security and compliance with recommended frontend development best practices. This difference may impact the robustness and user trust in the application, especially in production environments.

Regarding SEO optimization, both architectures scored high and followed the best practices recommended by Lighthouse. Both the modular and monolithic frontends demonstrated good mobile compatibility and used structured data to facilitate tracking and indexing by search engines.

These results indicate that both approaches are well-prepared in terms of SEO, providing good visibility and ease of content indexing.

5.2 Lighthouse Timespan Test in the Performance Category

This section presents the performance evaluation of the monolithic and modular applications using Lighthouse in Timespan mode, focusing on loading times and responsiveness of key visual elements.

The monolithic application scored 13 out of 23. Critical performance issues were identified while the Cumulative Layout Shift (CLS) was 0, indicating excellent visual stability. The Total Blocking Time (TBT) reached 150.020 ms, and Interaction to Next Paint (INP) was 35.620 ms (Table 2). According to the diagnostics (Table 3), the main thread was heavily burdened, consuming 162.2 seconds, and 53.9 seconds were spent executing JavaScript alone. These findings suggest excessive JavaScript usage, inefficient rendering, and latency in backend communication (8.450 ms). Lighthouse recommends optimizing script size and complexity,

improving backend latency, and using asynchronous loading to improve responsiveness.

Table 2: Performance Results.

Metrics	Value
Total Blocking Time	150.020 ms
Cumulative Layout Shift	0
Interaction to Next Paint	35.620 ms

Similarly, the modular version also scored 13/23 but showed higher TBT (246.390 ms) and INP (70.030 ms), as detailed in Table 4. JavaScript execution consumed 58.7 seconds, and although the backend response was faster (4.140 ms), script-heavy processing remained a bottleneck. Recommendations include reducing JavaScript complexity, splitting scripts into smaller bundles, leveraging asynchronous loading, and optimizing asset delivery to enhance performance and interactivity.

Both architectures exhibit performance limitations primarily due to heavy JavaScript usage and main-thread blocking. Addressing these issues through script optimization and backend improvements is essential to ensure better responsiveness and user experience.

5.3 Lighthouse Timespan Test in the Best Practices Category

This section presents the results of the Lighthouse tests in Timespan mode under the Best Practices category, which evaluates aspects related to security, performance, and compatibility. Eight key criteria were analyzed for both the monolithic and modular frontend implementations. The monolithic frontend scored 6 out of 8, indicating that six best practices were correctly implemented, while two require adjustments. Among the positive aspects were using HTTPS for secure communication and the proper configuration of images with correct aspect ratios and responsiveness, which ensures a consistent visual experience across devices.

However, two main issues were identified. First, the use of obsolete APIs, specifically the continued use of ReactDOM.render in React 18, which is deprecated and should be replaced with createRoot. Second, there is an absence of source maps, which are crucial for debugging minified JavaScript code in production. Additionally, the report warned about the upcoming deprecation of Chrome's third-party cookies and noted some browser console errors, indicating unresolved issues in the application code.

Overall, the monolithic implementation follows essential security and usability practices, but should update its API usage and provide source maps to improve maintainability and meet current development standards.

The modular frontend scored slightly higher, with 7 out of 8 criteria met. Like the monolithic version, it uses HTTPS and serves images correctly. In addition, it avoided obsolete APIs, did not rely on third-party cookies (a future requirement for compliance with modern browser standards), and included valid source maps, facilitating easier maintenance and debugging. No significant issues were reported in Chrome DevTools, indicating a well-structured and functional implementation.

The only shortcoming was the continued use of ReactDOM.render, which, as in the monolithic version, should be replaced by createRoot to align with React 18 standards. Aside from this, the modular frontend demonstrated strong adherence to best practices, showing a more future-ready and maintainable structure.

In summary, both implementations followed key development best practices. However, the modular architecture performed better by avoiding obsolete features and implementing support tools that enhance debugging and long-term maintainability.

6 DISCUSSION

The tests conducted with Lighthouse revealed significant differences between monolithic and modular architectures in terms of performance, accessibility, best practices, and SEO, particularly in the context of vectorized data ingestion. Due to its integration, the monolithic architecture excelled in Snapshot mode in metrics such as First Contentful Paint (FCP) and Speed Index, allowing faster loading of main visual elements. However, this approach can become problematic as the application grows in complexity, making code maintenance and expansion more difficult.

In contrast, the modular architecture showed limitations in loading time related to the size of the DOM and viewport settings, negatively impacting performance in applications processing vectorized data. Nevertheless, modularization provides a more flexible and scalable structure, enabling updates and optimizations to specific parts of the code without affecting the base. Backend latency and the processing of large data volumes also influence frontend performance. Thus, while the monolithic architecture offers initial advantages, modularity is more beneficial in terms of long-term scalability and maintainability.

Table 3: Lighthouse Diagnostics Summary.

Diagnostics	Details
Minimize main-thread work	162.2 s
Reduce JavaScript execution time	53.9 s
Enable text compression	Error
Minimize work during key interaction	35,620 ms on 'mousedown'
Minify JavaScript	759 KiB savings
Avoid serving legacy JavaScript	0 KiB savings
Reduce unused JavaScript	721 KiB savings
Avoid enormous network payloads	47,506 KiB total
Avoid long main-thread tasks	20 long tasks found

Table 4: Modular Performance Metrics Summary.

Metrics	Value
Total Blocking Time	246.390 ms
Cumulative Layout Shift	0.022
Interaction to Next Paint	70.030 ms

In big data scenarios, modular architecture, when combined with lazy loading and bundle optimization techniques, can outperform monolithic architecture in efficiency after optimizations. Modularity is more efficient, especially in projects that require continuous evolution.

In summary, the modular front broadly followed the recommended best practices, with one necessary adjustment related to console errors. Correcting this issue will bring the modular frontend fully compliant with development best practices, achieving a perfect score in future audits.

7 CONCLUSION

This study compared monolithic and modular frontend architectures, evaluating their performance across various quality and efficiency metrics using the Lighthouse tool. The results demonstrated that the monolithic architecture showed superior initial performance, especially in loading time and interactivity, due to resource integration and optimized delivery. The modular architecture proved to be more aligned with modern development best practices, such as maintainability and security, making it a more flexible and scalable choice for long-term applications that handle large volumes of vectorized data.

The analysis highlighted that although the monolithic frontend is initially faster, modularity offers a more organized and robust structure, essential for growth and adaptation to new demands, such as systems using Retrieval-Augmented Generation (RAG). However, a necessary limitation of this study was conducting the tests in a local environment, compromising the results' external validity. In a natural production environment, factors such as network latency and load balancing could impact the performance of both architectures differently. For future work, it is recommended to conduct tests in production environments simulating high-load scenarios, explore hybrid architectures that combine the benefits of both approaches, and assess the applicability of microfrontends in simplified RAG contexts.

REFERENCES

Aka Uymaz, H. and Kumova Metin, S. (2022). Vector based sentiment and emotion analysis from text: A survey. *Engineering Applications of Artificial Intelligence*, 113:104922.

Bian, Y., Ma, D., Zou, Q., and Yue, W. (2022). A multi-way access portal website construction scheme. In *2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 589–592. IEEE.

Chen, A. P. S. (2023). Using conversational ai to service organizational agility and flexibility: The dynamic capability and option view. In *Proceedings of the 2023 5th World Symposium on Software Engineering*, pages 67–70.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. (2024). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Helfat, C. E. and Winter, S. G. (2011). Untangling dynamic and operational capabilities: Strategy for the (n) ever-changing world. *Strategic management journal*, 32(11):1243–1250.

Ian, S. (2019). *Engenharia de Software*. Addison-Wesley/Pearson.

Krishnamurthy, S. (2019). What are micro frontends. *IEEE India Info*, 14(4):105–109.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Lo, J. and Leidner, D. (2018). Are dynamic capabilities the missing link between the is strategy and performance relationship? a model and exploratory test at three levels of environmental dynamism. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 49(2):35–53.

Männistö, J., Tuovinen, A.-P., and Raatikainen, M. (2023). Experiences on a frameworkless micro-frontend architecture in a small organization. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 61–67. IEEE.

Mariyam, A., Basha, S. A. H., and Raju, S. V. (2021). A literature survey on recurrent attention learning for text classification. *IOP Conference Series: Materials Science and Engineering*, 1042(1):012030.

Mikalef, P., Boura, M., Lekakos, G., and Krogstie, J. (2019). Big data analytics capabilities and innovation: The mediating role of dynamic capabilities and moderating effect of the environment. *British journal of management*, 30(2):272–298.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Occhipinti, A., Rogers, L., and Angione, C. (2022). A pipeline and comparative study of 12 machine learning models for text classification. *Expert Systems with Applications*, 201:117193.

Park, N., Liu, F., Mehta, P., Cristofor, D., Faloutsos, C., and Dong, Y. (2022). Evokg: Jointly modeling event time and network structure for reasoning over temporal knowledge graphs. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 794–803.

Peltonen, S., Mezzalira, L., and Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136:106571.

Perlin, R., Ebling, D., Maran, V., Descovi, G., and Machado, A. (2023). An approach to follow microservices principles in frontend. In *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–6. IEEE.

Schilke, O., Hu, S., and Helfat, C. E. (2018). Quo vadis, dynamic capabilities? a content-analytic review of the current state of knowledge and recommendations for future research. *Academy of management annals*, 12(1):390–439.

Simões, B., del Puy Carretero, M., Martínez, J., Muñoz, S., and Alcain, N. (2024). Implementing digital twins via micro-frontends, micro-services, and web 3d. *Computers & Graphics*, page 103946.

Supriyono, Wibawa, A. P., Suyono, and Kurniawan, F. (2024). Advancements in natural language processing: Implications, challenges, and future directions. *Telematics and Informatics Reports*, 16:100173.

Taibi, D. and Mezzalira, L. (2022). Micro-frontends: Principles, implementations, and pitfalls. *ACM SIGSOFT Software Engineering Notes*, 47(4):25–29.

Vescovi, R., Ginsburg, T., Hippe, K., Ozgulbas, D., Stone, C., Stroka, A., Butler, R., Blaiszik, B., Brettin, T., Chard, K., Hereld, M., Ramanathan, A., Stevens, R., Vriza, A., Xu, J., Zhang, Q., and Foster, I. (2023). Towards a modular architecture for science factories: Electronic supplementary information (esi) available. see doi: https://doi.org/10.1039/d3dd00142c. *Digital Discovery*, 2(6):1980–1998.

Wamba, S. F., Gunasekaran, A., Akter, S., Ren, S. J.-f., Dubey, R., and Childe, S. J. (2017). Big data analytics and firm performance: Effects of dynamic capabilities. *Journal of business research*, 70:356–365.

Wang, D., Yang, D., Zhou, H., Wang, Y., Hong, D., Dong, Q., and Song, S. (2020). A novel application of educational management information system based on micro frontends. *Procedia Computer Science*, 176:1567–1576.

Wang, X., Wang, Z., Gao, X., Zhang, F., Wu, Y., Xu, Z., Shi, T., Wang, Z., Li, S., Qian, Q., et al. (2024). Searching for best practices in retrieval-augmented generation. *arXiv preprint arXiv:2407.01219*.

Xiao, X., Tian, Q., and Mao, H. (2020). How the interaction of big data analytics capabilities and digital platform capabilities affects service innovation: A dynamic capabilities view. *IEEE Access*, 8:18778–18796.