

WEB APPLICATION FOR GENERATING TECHNICAL DOCUMENTS USING LARGE LANGUAGE MODELS

Martin Šulc¹, Ivo Pisařovic¹

¹Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

DOI: https://doi.org/10.11118/ecos2026_037

ABSTRACT

This paper investigates the viability of leveraging large language models (LLMs) to generate formal corporate documents, moving beyond rigid, traditional template-based approaches. The study aims to verify whether combining reference samples, natural language instructions, and parallel multi-model processing can yield a practical enterprise tool. The proposed microservices architecture utilizes a Node.js/LangChain backend to abstract calls across four LLM providers (OpenAI, Anthropic, Gemini, and Ollama). This is supported by Python microservices for input parsing and multi-format document generation (Word, PDF, Excel), a Vue.js frontend wizard, and PostgreSQL/Redis persistence. System validation involved end-to-end scenarios across five document types, benchmarking seven models, and user testing with ten participants. Results demonstrate the solution's efficacy. Furthermore, comparative analysis reveals that local open-source models achieve generation quality comparable to mid-range cloud services for Czech-language tasks, albeit with trade-offs in increased latency and initial hardware investments.

Keywords: large language models, document generation, web application, microservices

JEL Code: C88, L86

1 INTRODUCTION

Companies work with a large number of formal documents every day – technical documentation, protocols, emergency plans or notes. These documents need to be created, updated and maintained regularly, typically by a worker taking an older document of a similar type, manually editing it, adding current data from various sources and compiling the result into the required form. The whole process is slow and prone to errors. At the same time, there is empirical data that the deployment of generative language models in writing processes significantly increases worker productivity and output quality (Noy and Zhang, 2023), while extensive field experiments in companies have



This article is licensed under CC BY-SA 4.0 Attribution-ShareAlike 4.0 International license. <https://creativecommons.org/licenses/by-sa/4.0/deed.en>

recorded double-digit productivity increases even for routine text tasks (Brynjolfsson *et al.*, 2025). These observations have framed a broader debate about the opportunities and risks of generative AI in professional practice (Dwivedi *et al.*, 2023).

Available tools for creating corporate documents only partially solve this problem. Template generators such as Docupilot (Docupilot, 2026) operate on the principle of predefined fields into which the system fills in data – an approach useful for repetitive documents with a fixed structure, but failing when it is necessary to adapt the content to a new context or generate text that was not in the template. Document extraction and analysis tools such as Azure AI Document Intelligence (Azure AI, 2025) only solve the opposite direction of the flow, i.e. converting existing documents into structured data. Platforms with integrated artificial intelligence (Writer, 2026) handle content generation, but they mainly focus on marketing and business output, not technical documentation.

Large language models bring new quality, as they can analyze the structure of input documents and generate new content based on requirements formulated in natural language (Ke *et al.*, 2025; Min *et al.*, 2023). However, the LLM market is experiencing rapid development and the number of commercially and locally available models is in the hundreds (Chang *et al.*, 2024), which for practical deployment means the need to introduce an abstraction layer above a specific provider so that model replacement does not require rewriting of application logic. Prompt-based learning has emerged as the central paradigm, where the behavior of the model is controlled by carefully designed system instructions and templates for user inputs without the need for fine-tuning (Liu *et al.*, 2023). At the same time, empirical evaluations across dozens of NLP tasks show that the quality of LLM outputs varies significantly by task and language (Kocoń *et al.*, 2023), which justifies conducting our own comparison on a specific scenario of generating formal documents in Czech. It also remains an open question to what extent current local open-source models available through runtime environments such as Ollama (Ollama, 2026) are able to compete with cloud services in this task.

The aim of the work (Šulc, 2026) on which this contribution is based was to design and implement a web application that fills the gap identified in the market overview. The user uploads older sample documents, formulates requirements in natural language, and the application generates a new formal document in the selected format (Word, PDF or Excel) using LLM. Key assumptions were support for several providers including local deployment, the possibility of parallel comparison of multiple models on the same assignment, a primary focus on the Czech language, and a user interface usable even for participants without detailed knowledge of artificial intelligence. The intention was not to create a production-ready product, but to explore currently available technologies, demonstrate their practical use for a given purpose, and verify the functionality of the resulting solution on real documents of a specific company.

This paper summarizes the design and selected results of the comparison of language models and user testing. The Methodology and Data chapter describes the chosen development approach, the microservices-based architecture, the technologies used, and the testing methodology. The Results chapter presents a comparison of seven models from three cloud providers and four on-premises variants, and selected conclusions from user testing. The Discussion and Conclusions then summarize the achievement of the objectives, identified weaknesses, and recommendations for further development, including considerations of the economic and environmental dimensions of LLM deployment.

2 METHODOLOGY AND DATA

The application was developed using an incremental approach (Hoda *et al.*, 2017), where functionality is delivered gradually in smaller units. The choice is based on the nature of the project, as many design decisions (for example, the optimal structure of the system instruction or the scope of the data model) depend on the knowledge gained only during experiments with real documents. A fully sequential waterfall approach was not suitable here, as some requirements were refined continuously based on the behavior of specific language models.

The system architecture is implemented as a set of microservices run via Docker Compose. This arrangement follows the principles described in Newman (2015) and allows for leveraging the strengths of different technologies where it makes the most sense, ensuring that the failure of one service does not affect the rest of the system, and facilitating independent scaling and deployment of individual components (Jamshidi *et al.*, 2018).

The core of the application is a back-end on the Node.js platform with the Express application framework and the LangChain.js library. This provides a unified abstraction over language model providers (OpenAI, Anthropic, Google Gemini and locally operated Ollama), so that adding another provider only requires implementing the corresponding adapter without interfering with the rest of the application logic (Topsakal and Akinci, 2023). The behavior of the model is controlled by a central system instruction passed to each call along with the user input, which corresponds to the prompt-based learning paradigm (Liu *et al.*, 2023): a single carefully designed instruction maintains output consistency (language, formatting, rules for subsequent modifications) across all models and eliminates duplication in the code. Communication with the client is carried out via REST API and WebSocket connection through the Socket.IO library, which allows sending continuous notifications about the processing status without burdening the server with periodic polling.

Specialized document operations are separated into two Python microservices. Markdown Parser uses the open-source Markdown library from Microsoft (Markdown, 2024) to convert input documents (PDF, DOCX, XLSX, PPTX, RTF) into a unified Markdown format, which is the optimal input for language models, as it preserves the structure (headings, lists, tables) in readable text form without noise. Document Generator then creates output files from structured data in Word (python-docx library), Excel (openpyxl) and PDF (reportlab) formats. The choice of Python for these operations is based on significantly more advanced libraries for working with document formats, which do not have a full-fledged equivalent in the Node.js ecosystem.

The front-end is implemented in Vue.js with the Quasar application framework, which provides an extensive set of components ready for creating a multi-step wizard without the need to integrate third-party libraries (Kaluža and Vukelić, 2018). Persistence combines a PostgreSQL relational database for all persistent data (sessions, files, instructions, results, conversations) with an in-memory Redis storage, which serves as a fast cache, a priority queue for asynchronous instruction processing, and a communication channel for WebSocket notifications. All services are containerized and executable with a single command (Merkel, 2014), eliminating problems with inconsistent environments between development and production installations. User interaction takes the form of a four-step wizard: uploading files and assigning roles, entering instructions, configuring the model (with the possibility of parallel selection of multiple models for comparison), and reviewing the results with export. The architecture of the entire system is depicted in Figure 1.

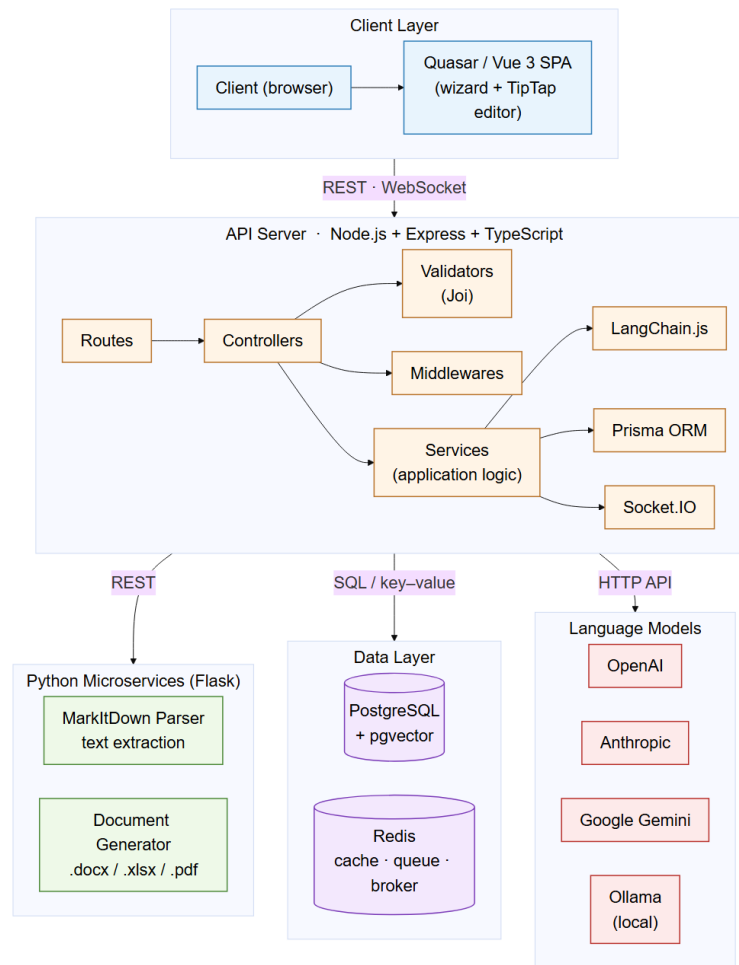


Fig. 1: System architecture with indication of main services (back-end, Python microservices, data layer, client) and their communication links (Šulc, 2026)

Testing was carried out in three layers. Unit and integration tests in the Jest application framework cover validators, provider configuration, REST API endpoints and input file parsing. End-to-end verification of generation quality was performed on five types of documents (technical report, annual report, multi-file project documentation, completed form and additional variants) using prepared scenarios with defined tasks and checklists. Comparison of language models included seven representatives – three cloud-based (OpenAI GPT-5.4, Anthropic Claude Sonnet 4.6, Google Gemini 3 Flash) and four local ones available via Ollama (Qwen 3:32b, Gemma 3:27b, Llama 3.1:8b and Mistral 7b). The design of evaluation criteria (quality of Czech, adherence to the required structure, processing speed, operational cost) is based on the approach of multidimensional LLM assessment across tasks described by Kocoń *et al.* (2023). The measurement was performed on a workstation with an NVIDIA RTX 5090 graphics card with 32 GB VRAM. User testing was based on Nielsen's (2024) recommendation for a combination of task scenarios and a think-aloud protocol and was conducted on a sample of ten participants (seven with an IT background, two economists, one translator), who were tasked with creating a template for future use from the completed form.

3 RESULTS

The result of the work is a functional application that meets the main objectives of the assignment. The application allows uploading documents in key formats (PDF, DOCX, XLSX, CSV, RTF, Markdown), converting them to Markdown via a Python microservice, compiling instructions with the ability to target a specific file or section, parallel calling of several language models from different providers and subsequent generation of structured outputs in Word, Excel and PDF formats. The user interface guides the user through the entire process linearly, and the Socket.IO library sends continuous notifications about the processing status. The specific form of the finished application is illustrated in Figure 2.

3.1 Comparison of language models

Key findings from the comparative testing of the models are summarized in Table 1. Several conclusions emerge from the measured values. Cloud models show significantly lower response times (typically 5–25 seconds) than local models (1–3 minutes), which corresponds to the computing capacity and optimization of the infrastructure of individual providers. The quality of the generated Czech and the ability to adhere to the specified structure are at a high level among the cloud models and reach the highest values in the Claude Sonnet 4.6 model. The observed variability in performance across models corresponds to the general finding from larger comparative studies, according to which no current model represents a clear winner in all tasks and metrics (Kocoń *et al.*, 2023; Min *et al.*, 2023).

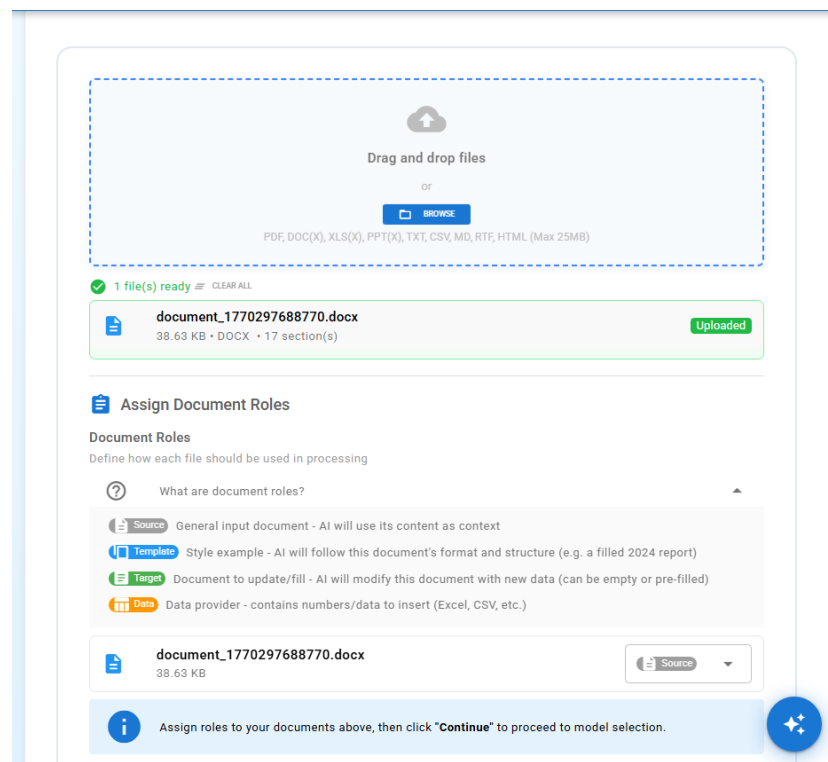


Fig. 2: Sample wizard user interface – first step with drag-and-drop file upload and role assignment (Šulc, 2026)

Tab. 1 Comparison of language models according to key features for generating formal documents in Czech

Model	Type	Context	Price / Specifications	Czech	Structure	Speed
GPT-5.4	cloud	1M	\$2.50 / \$10 per 1 M stream	8/10	8/10	10–20 seconds
Claude Sonnet 4.6	cloud	1M	3 / 15 USD for 1 M flow	9/10	8/10	15–25 seconds
Gemini 3 Flash	cloud	1M	\$0.50 / \$4 per 1 M flow	8/10	7/10	5–15 seconds
Qwen 3:32b	local	32,768	32 B / 24 GB VRAM	8/10	7/10	2–3 minutes
Gemma 3:27b	local	128K	27 B / 24 GB VRAM	7/10	6/10	2–3 minutes
Llama 3.1:8b	local	128K	8 B / 8 GB VRAM	6/10	4/10	1–1.5 minutes
Mistral 7b	local	8,192	7 B / 8 GB VRAM	3/10	2/10	≈ 2 minutes

Among the local models, Qwen 3:32b and Gemma 3:27b show significantly the highest quality of Czech and structured output, but they require a graphics card with at least 24 GB VRAM (typically NVIDIA RTX 3090 or 4090). Smaller models (Llama 3.1:8b and Mistral 7b) running on a regular consumer graphics card with 8 GB VRAM are on the verge of usability for generating longer formal texts in Czech – Mistral 7b in particular proved to be insufficient in tests. These findings have a direct impact on the decision to deploy in an organization with strict data protection requirements: a high-quality local alternative to cloud models is available, but at the cost of higher initial investments in hardware and several times longer response times.

3.2 User testing

Of the ten participants in the user test, all completed the task without external assistance. The application was generally rated as clear and no participant was visibly lost in the guide. Most participants used the parallel processing feature and compared the outputs of one local model (Qwen 3:32b or Gemma 3:27b) with the cloud-based GPT-5.4 simultaneously. This functionality provided users with immediate feedback on the differences between the models and made it easier to choose the appropriate variant for a specific type of task.

At the same time, two specific areas with a worse user experience were identified. The first is the file role selection in the initial step, where the labels template, source and target were not intuitively understandable for the task of creating a template from an existing document – the term template does not make intuitive sense here, since a template is only supposed to be an output, not an input. The second is the screen for selecting the output type before starting processing, where the labels of the individual options and their mutual relationship were not easily distinguishable, which led to uncertainty among some participants as to whether the settings were correct. This observation was consistent across professions and there was no significant difference between participants with and without an IT background.

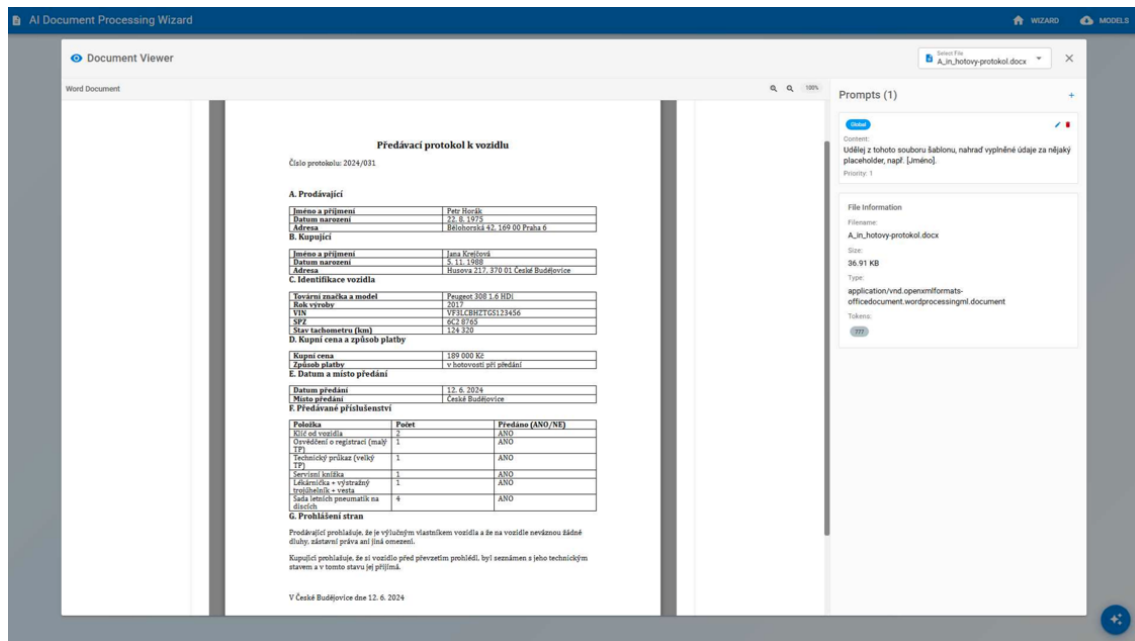


Fig. 3: Document Viewer – preview of an uploaded document along with user's prompt (Šulc, 2026)

A useful side benefit of testing was the discovery of a bug that was not replicated more often during actual development: some generated documents contained fragments of language model meta-comments (for example, sentences like “if you want, I can add...”), even though the system instruction explicitly prohibits such introductory and accompanying phrases. The cause is the undefined behavior of specific models and the need to handle this additionally at the level of subsequent response processing.

The Document Viewer component (see Figure 3), which allows viewing the input document alongside the instructions, was explicitly identified by participants as one of the most beneficial features. The difference in response speed between the cloud and on-premises models was perceived by participants, but in most cases was not considered a barrier. There were minor stylistic differences between the outputs of individual models, but the structure was maintained consistently across all models in most cases, which can be considered an achievement considering the comparison of a top-of-the-line cloud model with on-premises variants running on commonly available hardware. The ability to complete the task depended more on the nature of the specific input document than on the user's experience with information technology, suggesting that the wizard successfully lowers the barrier to entry for users without detailed knowledge of artificial intelligence.

4 DISCUSSION AND CONCLUSIONS

The verification demonstrated that the combination of an abstraction layer over language model providers, a microservice architecture, and a wizard interface represents a functional approach to automating the generation of formal documents that goes beyond the traditional template approach. The application is a fully containerized multi-component solution ready to run with a single command, significantly reducing the need for manual configuration. It fills the gap identified in the

market analysis: a complete open-source solution that would combine input document analysis, new content generation via LLM, support for multiple providers including local deployment, and a focus on technical documentation in Czech was not available at the time of development.

The main value of the resulting application, as revealed by user testing, is the combination of a clear guide, support for multiple AI providers including local variants, and the function of parallel comparison of models on the same assignment. The Document Viewer for simultaneous display of the input document and entered instructions was appreciated by the users as one of the most beneficial functions. The choice of the Vue.js application framework with Quasar proved to be suitable for the rapid creation of a comprehensive guide, which corresponds to the conclusions of comparative studies of front-end technologies (Bielak *et al.*, 2022).

The identified weaknesses also indicate the direction of further development. The highest priority should be to rework problematic labels in the initial and final steps of the wizard, or to add contextual help directly in the interface, and to integrate subsequent output processing, which would automatically filter out meta-comments of the language model. The second significant topic is the stability of subsequent modifications within a single session, as the current version works reliably only for the initial instruction, while subsequent modifications of the same session are sensitive to the specific behavior of the model and do not achieve the same stability. The proposed solution includes dividing extensive modifications into smaller steps and more thorough work with the context of previous versions, or the use of structured output and function calls.

From the point of view of economic competitiveness, the work provides empirical evidence for deciding on the strategy for deploying LLM in a company. Selected local open-source models (especially Qwen 3:32b and Gemma 3:27b) achieve a quality comparable to mid-range cloud services for tasks of generating formal documents in Czech, albeit at the cost of several times longer response times and the necessary initial investment in hardware. This trade-off needs to be interpreted in the broader context of computing costs. Schwartz *et al.* (2020) introduced a distinction between Red AI (oriented to maximum performance at the cost of increasing computing demands) and Green AI (taking into account energy and operational efficiency) for deciding on the effectiveness of AI systems. A more detailed analysis of the energy and carbon footprint of operating large models (Patterson *et al.*, 2022) shows that choosing a smaller local model can be an economically and environmentally rational alternative for organizations whose tasks do not require top-quality output or that work with sensitive data for which sending content to third-party servers is unacceptable. Conversely, for tasks with high document volumes and tolerance for cloud processing, cloud providers remain the choice with the best balance of speed, quality and operating costs. At the same time, growing empirical evidence on the productivity benefits of generative AI in the corporate environment (Noy and Zhang, 2023; Brynjolfsson *et al.*, 2025) suggests that investing in document creation automation, whether via cloud or local deployment, represents a rational decision with a demonstrable return.

In conclusion, it can be stated that the main objectives of the work – design, implementation and verification of a web application for generating formal documents with support for large language models – have been fulfilled. The application was tested on five document types, end-to-end scenarios, seven language models and user testing with ten participants covering both technical and non-technical professions. The identified areas of further development (editing the interface labels, post-processing of the output, stabilization of subsequent edits and deeper integration of reasoning models) represent a direct continuation of the design described here.

Acknowledgements

This paper was supported by the project CZ.02.1.01/0.0/0.0/16_017/0002334 Research Infrastructure for Young Scientists, this is co-financed from Operational Programme Research, Development and Education.

Anthropic Claude was used to summarize the text of the final thesis (Šulc, 2026) and transform it into the form of this compact paper. The paper was then reviewed and fine-tuned by the authors. The authors are responsible for the correctness of the final text.

REFERENCES

- AZURE AI. 2025. Azure AI Document Intelligence documentation. *Microsoft*. [Accessed: 2026-02-14] <https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/>
- BIELAK, D., BOREK, K., PLECHAWSKA-WÓJCIK, M. 2022. Web application performance analysis using Angular, React and Vue.js frameworks. *Journal of the Computer Sciences Institute*. 23, 100–106. <https://doi.org/10.35784/jcsi.2827>
- BRYNJOLFSSON, E., LI, D., RAYMOND, L. R. 2025. Generative AI at Work. *Quarterly Journal of Economics*. 140(2), 889–942. <https://doi.org/10.1093/qje/qjae044>
- CHANG, Y., WANG, X., WANG, J., WU, Y., ZHU, K., CHEN, H., YANG, L. *et al.* 2024. A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology*. 15(3), 1–45. <https://doi.org/10.1145/3641289>
- DOCUPILOT. 2026. *Docupilot*. [Accessed: 2026-02-19]. <https://www.docupilot.com>
- DWIVEDI, Y. K., KSHETRI, N., HUGHES, L., SLADE, E. L., JEYARAJ, A., KAR, A. K. *et al.* 2023. “So what if ChatGPT wrote it?” Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International Journal of Information Management*. 71, art.102642. <https://doi.org/10.1016/j.ijinfomgt.2023.102642>
- HODA, R., SALLEH, N., GRUNDY, J., TEE, H. M. 2017. Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology*. 85, 60–70. <https://doi.org/10.1016/j.infsof.2017.01.007>
- JAMSHIDI, P., PAHL, C., MENDONÇA, N. C., LEWIS, J. W., TILKOV, S. 2018. Microservices: The Journey So Far and Challenges Ahead. *IEEE Software*. 35(3), 24–35. <https://doi.org/10.1109/MS.2018.2141039>
- KALUŽA, M., VUKELIĆ, B. 2018. Comparison of front-end frameworks for web applications development. *Zbornik Veleučilišta u Rijeci*. 6(1), 261–282. <https://doi.org/10.31784/zvr.6.1.19>
- KE, W., ZHENG, Y., LI, Y., XU, H., NIE, D., WANG, P., HE, Y. 2025. Large Language Models in Document Intelligence: A Comprehensive Survey, Recent Advances, Challenges, and Future Trends. *ACM Transactions on Information Systems*. 44, 1–64. <https://doi.org/10.1145/3768156>
- KOCOŃ, J., CICHECKI, I., KASZYCA, O., KOCHANIEK, M., SZYDŁO, D., BARAN, J. *et al.* 2023. ChatGPT: Jack of all trades, master of none. *Information Fusion*. 99, art.101861. <https://doi.org/10.1016/j.inffus.2023.101861>
- LIU, P., YUAN, W., FU, J., JIANG, Z., HAYASHI, H., NEUBIG, G. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys*. 55(9), 1–35. <https://doi.org/10.1145/3560815>
- MARKITDOWN. 2024. MarkItDown – Python tool for converting files and office documents to Markdown. *Microsoft*. [Accessed: 2026-03-14] <https://github.com/microsoft/markitdown>
- MERKEL, D. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*. 2014(239).

- MIN, B., ROSS, H., SULEM, E., VEYSEH, A. P. B., NGUYEN, T. H., SAINZ, O., AGIRRE, E., HEINTZ, I., ROTH, D. 2023. Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey. *ACM Computing Surveys*. 56(2), 1–40. <https://doi.org/10.1145/3605943>
- NEWMAN, S. 2015. *Building Microservices: Designing Fine-Grained Systems*. Sebastopol: O'Reilly Media. ISBN 978-1-491-95035-7
- NIELSEN, J. 2024. Usability Testing 101. *Nielsen Norman Group*. [Accessed: 2026-04-15] <https://www.nngroup.com/articles/usability-testing-101/>
- NOY, S., ZHANG, W. 2023. Experimental evidence on the productivity effects of generative artificial intelligence. *Science*. 381(6654), 187–192. <https://doi.org/10.1126/science.adh2586>
- OLLAMA. 2026. Ollama – Get up and running with large language models. *Ollama*. [Accessed: 2026-03-08]. <https://ollama.com/>
- PATTERSON, D., GONZALEZ, J., HÖLZLE, U., LE, Q., LIANG, C., MUNGUIA, L.-M., ROTHCHILD, D., SO, D. R., TEXIER, M., DEAN, J. 2022. The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink. *Computer*. 55(7), 18–28. <https://doi.org/10.1109/MC.2022.3148714>
- SCHWARTZ, R., DODGE, J., SMITH, N. A., ETZIONI, O. 2020. Green AI. *Communications of the ACM*. 63(12), 54–63. <https://doi.org/10.1145/3381831>
- ŠULC, M. 2026. *Webová aplikace pro generování formálních dokumentů za podpory AI*. Diploma thesis. Brno: Mendel University in Brno.
- TOPSAKAL, O., AKINCI, T. C. 2023. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. In: *Proceedings of the International Conference on Applied Engineering and Natural Sciences (ICAENS)*, pp. 1050–1056.
- WRITER. 2026. *Writer – The enterprise AI platform*. [Accessed: 2026-02-21]. <https://writer.com>

Contact information

Martin Šulc: e-mail: xsulc2@mendelu.cz

Ivo Pisařovic: e-mail: research@pisarovic.cz