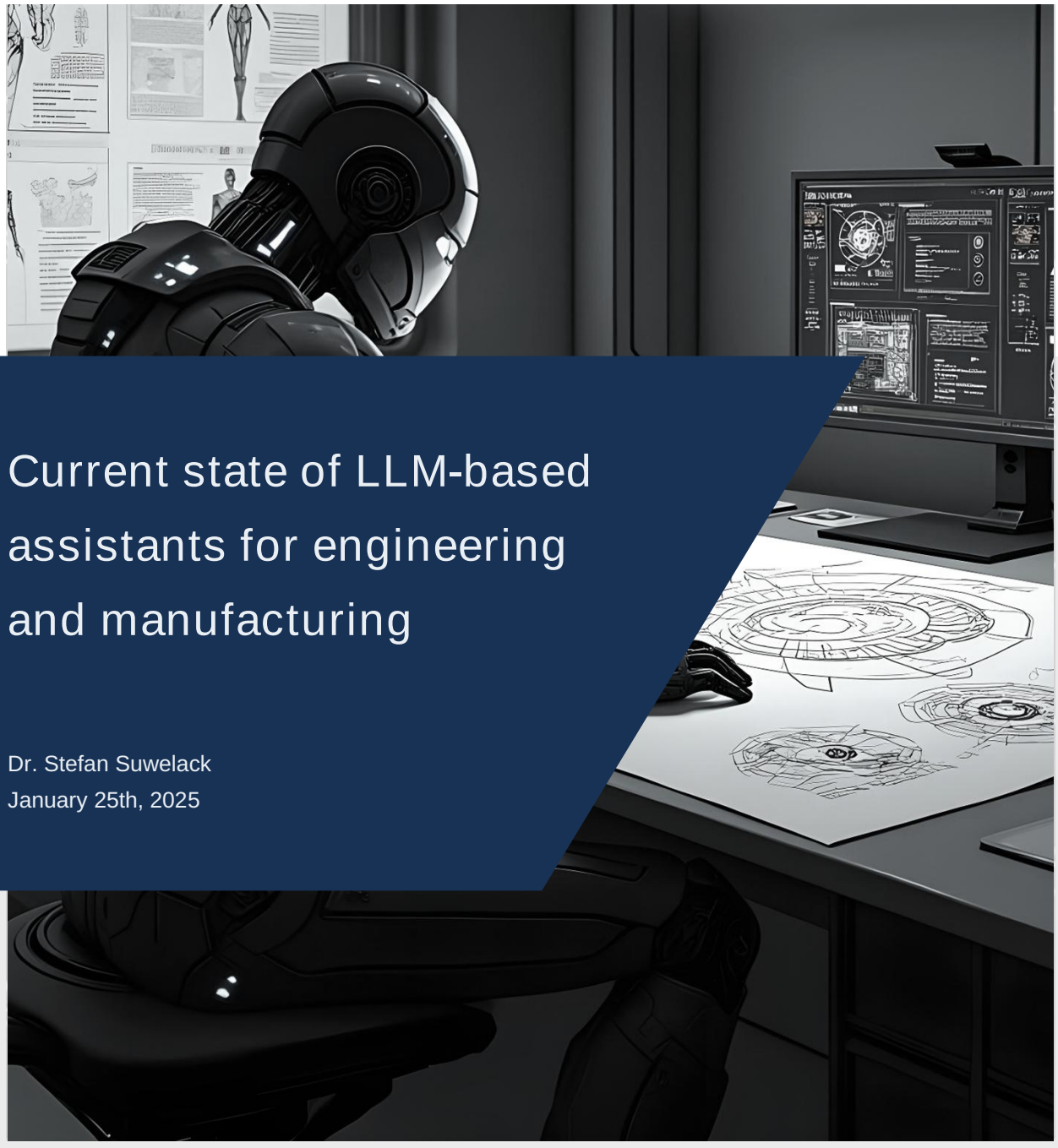
+49 173 805 3434
stefan.suwelack@renumics.com

Building J.A.R.V.I.S



Current state of LLM-based assistants for engineering and manufacturing

Dr. Stefan Suwelack
January 25th, 2025

Table of Contents

RAG and agentic AI beyond the hype

LLM-based assistants are here to stay. They will not only change how engineers work, but also how we design our IT infrastructure, how we use software tools and ultimately how we think about developing and manufacturing complex products. This guide serves as an honest introduction to the topic. It is intended for innovators, decision makers and practitioners that want to take the journey of building AI assistants for their teams and organizations.

Introduction

Having an artificial intelligence (AI) assistant that helps engineers to co-create and operate complex machinery has been a staple of science-fiction for a long time. Nothing embodies this idea more than J.A.R.V.I.S – the AI that helps Tony Stark to design and operate the Iron Man suit. Just 24 months ago this concept was pure fiction and I hated to see projects being labeled as “Jarvis” when in fact it was at best a simple chatbot.

A lot has changed in the last 24 months. It is clear now that the technology is there to make the J.A.R.V.I.S vision a reality: An AI that helps engineers during the whole product lifecycle by finding important information and taking over operational tasks. This will not only change how engineers work, but also how we design our IT infrastructure, how we use software tools and ultimately how we think about developing and manufacturing complex products.

However, the biggest driver for AI adoption in engineering is not that the technology is just awesome or that we all would love to be Iron Man. No, large language models (LLMs) and agentic AI systems promise to solve the biggest problem in product development and manufacturing:

Designing, manufacturing and operating modern mechatronic systems is incredibly complex. A first reaction to this increasing complexity was to split engineering work across specialized teams and organizations. At large automotive OEMs, engineering knowledge is now fragmented across hundreds of software systems and thousands of suppliers. As a result, critical contextual information has become siloed which constitutes a major hurdle for innovation in every step of the product lifecycle. This is especially true for innovations that require holistic thinking such as the software-defined vehicle. Ironically, we have these huge challenges in using engineering data effectively although we have orders of magnitude more data available than a decade ago.

The biggest value proposition of LLMs is to contextualize an input to the model. This is because the training corpuses for these foundational models essentially encompass at least all publicly available text data that exists today. Through technologies such as retrieval augmented generation (RAG) this ability can be enhanced by finding and using additional information from local documents and databases. In this way, highly specific domain knowledge becomes available to break up data silos.

Unfortunately, this superpower of LLMs can only be used if the data is accessible at all. Furthermore, LLMs aren’t a silver bullet; while they can contextualize data, their level of accuracy still hinges on the data quality,

especially in terms of semantic descriptions and completeness. This is why the first core challenge for building J.A.R.V.I.S-like assistants is to have a data infrastructure that makes all information across the product life cycle accessible.

While contextualizing data is the biggest strength of LLMs, their stochastic nature and lack of logical thinking is their biggest weakness. This can lead to so-called hallucinations which is a euphemism for “the model just makes up random facts that sound convincing but are completely wrong”. To build reliable functionality on top of LLMs, it is almost always necessary to keep a human in the loop. This means that building efficient user interfaces (UIs) for LLMs is the second core challenge when building AI assistants for engineers. And we should point out that (at least right now) modern AI assistants do not look like J.A.R.V.I.S; instead of a voice-based communication, efficient UIs enable engineers to interact seamlessly with documents and data through old fashioned mouse and keyboard interfaces.

Both challenges imply that we just can’t realize J.A.R.V.I.S by bolting a Clippy 2.0 on top of existing applications. Instead, building J.A.R.V.I.S. means building out the necessary data and application landscape. This is a journey every organization must take. And while there might be shortcuts in terms of copying best practices or using off-the-shelf components for AI assistants, nobody should expect to be handed the keys for their very own J.A.R.V.I.S just by paying up for a standardized enterprise software solution.

This guide is intended for innovators, decision makers and practitioners that want to take the journey of building AI assistants for their teams and organizations. We will start with a short introduction to the technical fundamentals (I recommend skipping this if you are already familiar with the topic) and continue to describe the current state of the art in terms of use cases, system architectures and best practices.

The white paper is my attempt to put the current state of the art for engineering AI assistants into a self-containing 20-page limit. It is based on our substantial project experience and interviews with dozens of engineering teams. However, due to the evolving nature of the topic, it definitely contains bias and personal judgement. If you disagree with some of the concepts or recommendations in this guide, I’d love to hear from you! Please write me an email (stefan.suwelack@renumics.com) or connect with me on LinkedIn.

I hope the guide helps you to accelerate your personal AI journey. Especially given the current economic circumstances, I truly believe that speed and agility are essential to be successful in this transformation. Or, as Iron Man himself would put it: "J.A.R.V.I.S, sometimes you gotta run before you can walk!"

Karlsruhe, 20.01.2025

Stefan Suwelack

Transformers & LLMs

In this chapter we briefly look at the core technology that powers AI assistants: The Transformer architecture and the Large Language Models (LLMs) that can be built upon this model type.

We present basic concepts such as tokens, temperature or structured output that are relevant on a user and systems level.

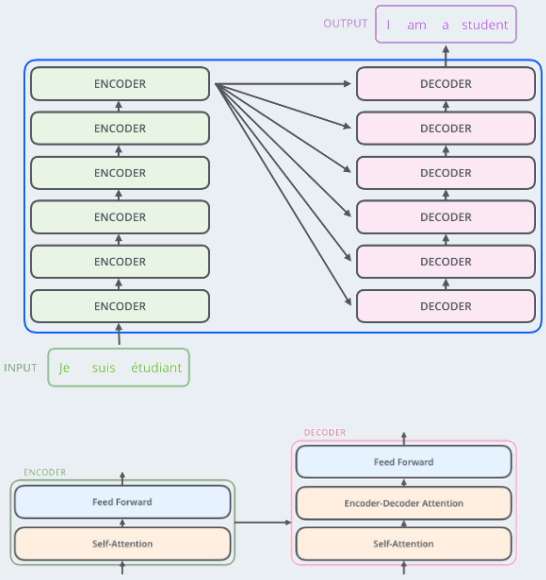


Fig 1: A sample transformer architecture to translate from French to English (top). Encoder and decoder layers each consist of an attention module and a feed forward network (bottom). Source: Jay Alammar

Transformers

Introduced in the seminal paper “Attention is all you need” in 2017, the Transformer model is a very efficient and scalable neural network architecture. It works by transforming a sequence of input tokens with arbitrary length into a series of output tokens. Transformers are autoregressive models, which means that they always predict the next token based on a given input/output sequence. The reason Transformers work so well is because the attention mechanism can handle long-range dependencies and fine-grained details in complex sequences.

The magic of the Transformer lies in its powerful ability to handle long-range dependencies in different data modalities such as text, image and time series data. One of the key drawbacks of the transformer technology is that the required inference compute depends quadratically on the size context size, i.e. the length of the token sequence that is currently processed. Before we discuss how Transformer architectures are trained into foundational large language models, let’s look at an example to understand how these token sequences actually work.

Imagine we want to train a Transformer to translate French sentences into English. (Fig. 1). Given the sentence “Je suis étudiant”, our model should output the sentence “I am a student”. Inside the model, each layer consists of

attention modules and a feed forward neural network and transforms the tensors flowing through. Understanding exactly what happens in every layer is not only out of scope for this guide, but also gives only marginal insides for practitioners. If you want to take a deeper dive I highly recommend the blog series "The illustrated transformer" by Jay Allamar and/or the amazing hands-on video "Let's build GPT: from scratch, in code, spelled out." by Andrej Karpathy.

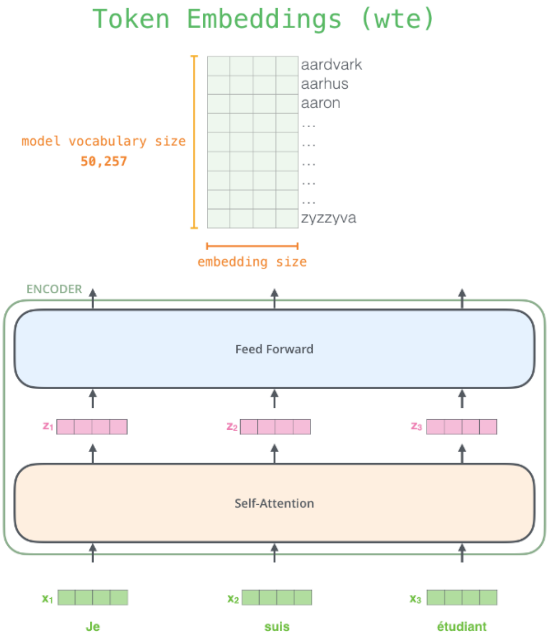


Fig 2: By assigning embedding vectors to input tokens (top), text data can be fed to a transformer (bottom). Source: Jay Alammar

What we need to understand in more detail is how we can feed our input data to the transformer and how we obtain the final output from the transformed tensor representation. To feed text data to a transformer, we first must transform the input data into vectors (Fig. 2). To do this, a vocabulary of so-called tokens is defined. Each entry can be a word or a character combination. The vocabulary is learned in such a way that common character sequences / words are efficiently parsed and represented. Token vocabulary sizes typically range from 50k (GPT-2) to 200k (GPT-4o). To serve as model input, the tokens must be converted to numbers. Instead of just passing an index, this is done by computing an embedding (Fig. 2 bottom) to capture the semantic meaning of each token. It is interesting to note that there are some edge cases where poor tokenization can lead to a total model failure. If you want to learn more about these glitch tokens, I can recommend this video.

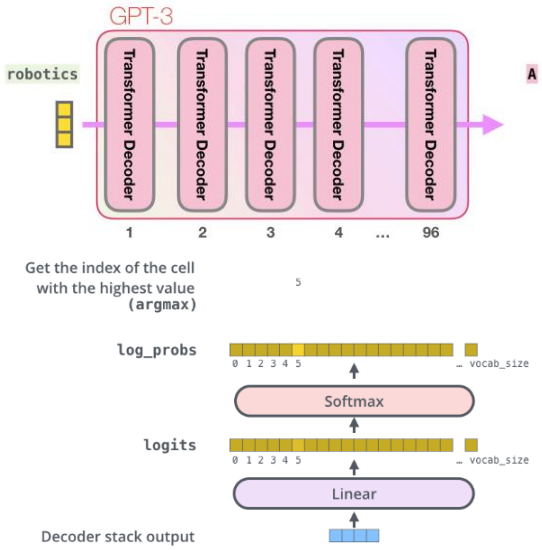


Fig 3: Input embeddings are transformed by passing through the layers of a Transformer model (left). The next token prediction is obtained by mapping the output of the model to a probability vector. Source: Jay Alammar

These input tensors are then passed through the transformer layers of the model (Fig.3 left) with the goal of predicting the next token. So, let's find out how we can transform the tensor output of the transformer model into a next token prediction. We first add a linear layer that maps the decoder stack output (Fig. 3 right) to a vector that has the size of the token vocabulary. We then apply the Softmax transformation to these logits: This transformation ensures that all output values are between 0 and 1 and the sum

of all entries in the resulting vector is 1. Therefore, the result gives us a probability for each token to be the candidate that should continue the sequence. We can now simply pick the token with the highest probability. If we want more randomness in the output, we can also sample from the given probability distribution. The hyperparameter that introduces more randomness into the sampling process is called Temperature. Sometimes we have prior knowledge of what the result should look like (e.g. should be valid Python code). In this case we can constrain the sampling such that only the most probable, but a valid next token is picked. This is typically called *structured output.*

LLM training and key shortcomings

LLMs make use of the efficient and scalable transformer architecture. In a first step, LLMs are trained by predicting the next token in a given sequence. More concretely, text-based transformers are trained by predicting the next word over a large training corpus. Current frontier models use all publicly available text data on the internet plus additional curated datasets that are specifically licensed for model training. There is also an increasing number of models that is not only trained on text data, but additional modalities like images or audio. After the pre-training phase there are typically additional training steps that adapt the model to a specific application. There are two different strategies used in this context: Classical fine-tuning uses the same self-supervised next-token prediction as the base training. However, fine-tuning is performed on a much smaller, highly-curated (and thus high-quality) dataset to adapt the model to a specific domain. To align the model with the intention of the user, a technique called Reinforcement Learning on Human Feedback (RLHF) is used. Here, the model generates different outputs and human labelers choose the one that they find most appropriate. This makes a model much more useful in a chat application. RLHF-based alignment is also used to safeguard the model against misuse.

These training strategies allow us to build powerful foundation models that have broad knowledge and can be used for a wide range of applications without additional training steps. However, LLMs have some important drawbacks that makes them difficult to use for many engineering and business applications:

1. LLMs can give false answers and there are no reliable methods to prevent these hallucinations.
2. LLMs perform poorly on tasks that require logical reasoning (see this Apple paper for a deep dive on the topic).
3. Although finetuning on custom data has become cheaper thanks to techniques like LoRA, it still requires significant resource for data curation and compute.
4. It is currently impossible to protect content that the LLM has been trained on (check out the Gandalf game for a playful introduction to LLM security).

All these challenges are directly related to the general model architecture and it solving them will require a major breakthrough in machine learning. It is very unlikely that this will happen over the next few years. However, as we will see in the next section, there are a lot of strategies to mitigate these challenges at the systems level.

Agentic AI

If I were to write a financial report as a human, I would not do so by typing the whole thing from the first to the last character in one go. Instead, I would reflect on my sources, come up with a structure, use Excel to calculate tables and draw figures, and discuss the whole thing with colleagues. These patterns can be transferred to LLM-based systems. Instead of zero-shot prompting the model in one go, we can engineer so called agentic LLM systems. This catch-all term denotes several different design patterns that essentially describe how to use LLMs in a larger workflows.

We should note that “AI Agents” is the tech buzzword of early 2025. While NVIDIA`s NVIDIA’s Jensen Huang says that IT will “become the HR of AI agents”, AWS ships a Github tutorial that promises an fully automated startup advisor. In this white paper, we adopt a very conservative and hands-on definition for agentic AI: LLM-infused workflows that use agentic patterns such as tool use, reflection, planning and flow control.

Slippery slope:
Agentic frameworks allow to cast nearly every problem as a use case for AI assistants. In some applications such as democratizing standard data transformations this can indeed make sense. For others (e.g. predictive maintenance), the assistant is only the icing on the cake. The challenging part of the system has still to be solved with classical ML or physics-based methods.

In practice, many different design patterns are fused together in complex agentic LLM workflows. When building such systems it becomes obvious that debugging and securing more complex agentic systems is a big challenge. It will take more time until best practices and tooling for this will mature. Especially because even simple agentic can provide a lot of user value, I highly recommend to stay away from overly complex pipe dreams.
Here are three basic agentic patterns that you can start with:

01 Tool Use

The LLM can execute code functions or use APIs to gather information, process data and take action.

02 Reflection

The LLM is used to judge preliminary results to iterate on steps such as data gathering or summarization.

03 Flow control

The LLM is used to control the flow of the execution by making and executing plans to achieve a given goal.

Retrieval Augmented Generation (RAG)

Among LLM-based workflow patterns, there is one that is especially noteworthy: Retrieval Augmented Generation (RAG) is the standard architecture for building AI assistants that can answer questions on document data. RAG is essentially a two-step process that is very straightforward (Fig. 4). Given a query and collection of documents, we first *retrieve* document sections that contain information to answer the query. Then, we *augment* the model context with these sections and ask the model to *generate* an answer based on the augmented context.

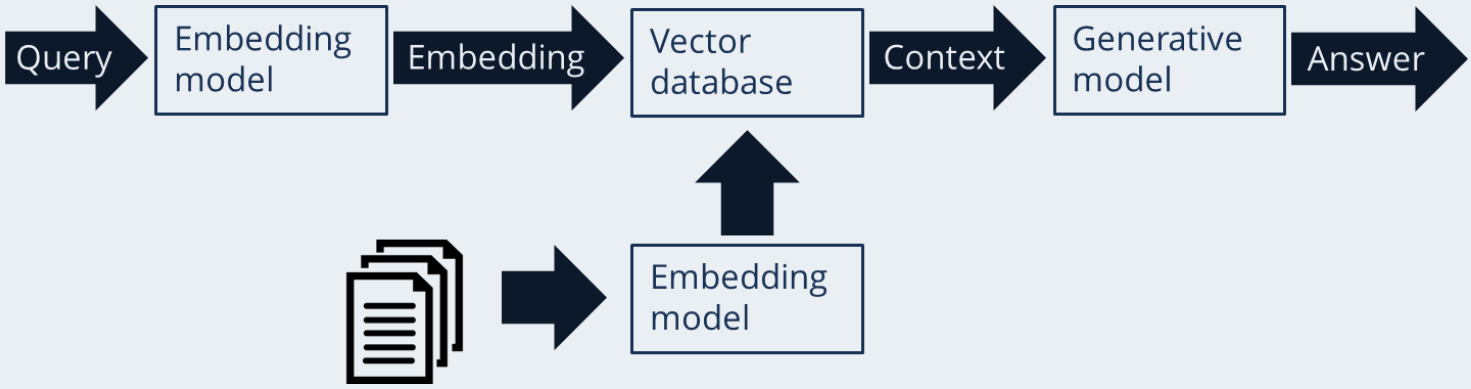


Fig 4: RAG is a two-step process: First, we retrieve relevant sections from a document collection, then an LLM generates an answer based on a context that has been augmented with these sections.

1. RETRIEVAL

The critical step for the accuracy of a RAG system is the retrieval process: Finding relevant sections (chunks) in the document collection. To do this, we first index our collection: We parse each document into sensible chunks, use an LLM to compute an embedding, and store the results in a vector database. As discussed in the introductory section, an embedding is a fixed-length vector representation of the input (e.g. text, images) that encodes the meaning of the input in a latent space. We can thus find relevant chunks by computing an embedding for the query and finding similar embeddings in the vector database. This can be done very efficiently, even for large multimodal data collections.

2. GENERATION

In a second step, the relevant chunks are fed to the model along with the instruction to answer the query based on the information given in the chunks. The size of each chunk and the number of chunks used are the primary parameters for this step. Current frontier models work well with larger context sizes (100k tokens), but there is a trade-off between the increased accuracy and speed / cost.

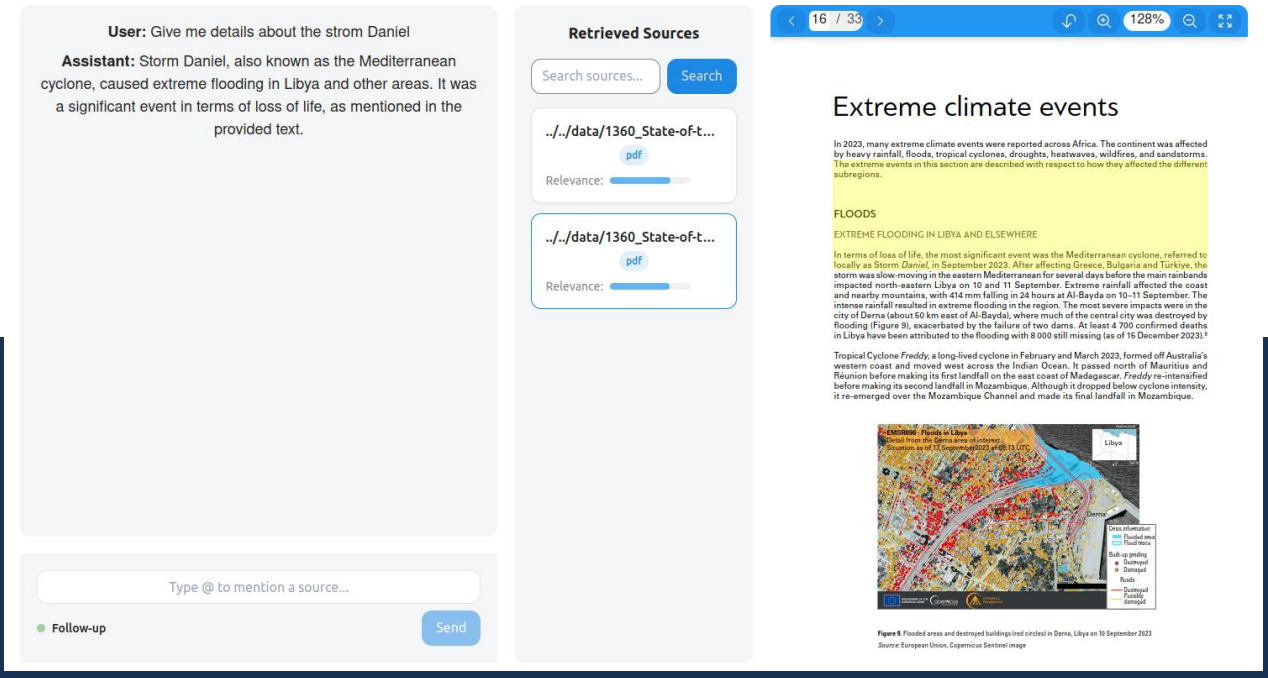


Fig 5: Simple RAG UI with source highlighting built on top the Renumics Lexio UI

OPTIMIZATIONS

In practice, nearly all optimization towards better accuracy and robustness is done within the retrieval step. There are many ways to enhance the presented *vanilla* RAG retrieval scheme: We can make the query more meaningful through *query re-writing*. We can also combine the embedding-based search with classical word-based search algorithms like BM25. Or we can re-rank the retrieved chunks by avoiding redundancy through the maximum marginal relevance (MMR) method or even train a specific LLM-model for the re-ranking task. For applications that involve documents with many tables or spreadsheets, enhanced pipelines for parsing the documents into chunks is typically the most effective optimization step.

ADVANTAGES

The RAG architecture solves three of the four big challenges of single-shot LLM use: First, RAG allows us to build assistants that can answer questions on internal data without any finetuning. Also, RAG systems can be secured in a straightforward way, because data access controls can be directly integrated into the retrieval process. Most importantly, hallucinations are greatly reduced. If the retrieved context does not contain enough information to answer the query, the system can just answer “I don’t know”. And the best part is that sources can be actively linked to and checked by the user. In this way, RAG system can be used for many critical workflows as long as a human stays in the loop.

Best Practices for RAG

#1 Metrics and evals

It is important to evaluate the performance of a RAG system through quantitative metrics in order to optimize it. In the beginning this is based on an initial set of questions (*golden dataset*). However, it is very important not to get too hung up on both question set and the accuracy metrics: Not only is it extremely difficult to judge if these questions are really really representative, but the accuracy is often only a weak proxy for user value.

#2 UX /UI is everything

The north star for every AI assistant should be to bring value for specific user workflows. The overall value of the system (e.g. time savings, quality improvements) is the sum of all these small steps. In practice good UX/UI that allows to manipulate the context and to explore sources is the most important ingredient for generating this value. Almost always does a good UI help to smooth deficiencies in RAG accuracy.

#3 Optimizing the data pipeline

Most failure modes for a RAG system are caused by data sources that are multimodal or ambigous (e.g. different document versions). More concretely, parsing images and tables correctly is always an issue. While the landscape for parsing tools is rapidly improving, the problem can often be side-stepped: Instead of using PDFs directly, try to find out if the documents are also available in a better format (e.g. HTML, markdown).

#4 Re-ranking and query re-writing

Query re-writing can be used to improve the relevancy of the embedding-based retrieval. In particular when specific vocabulary is used or nuanced difference (e.g. between products) are important, the method can significantly increase accuracy. Once you have a RAG system running with regular users, their feedback can be used to optimize the system by training custom re-ranking transformer model.

#5 GraphRAG is for LinkedIn

Vanilla RAG struggles with questions that require complex context and information from many sources. GraphRAG sounds like a promising solution, because it automatically extracts entities, relationships and communities from the sources. That is why graph-based approaches are very popular on social media. However, in practice these are not reliable enough and don't move the needle for real-world applications.

Industrial Use Cases

In a few years, every engineer will work with AI assistants based on agentic systems across a variety of tasks. This will have a tremendous impact on the data infrastructure and software landscape in engineering and manufacturing. More concretely, the available tooling will converge towards two poles: Tools that have a tightly integrated assistant with frequent interactions (e.g. software IDEs, CAD) and software components that have an API which is optimized to be called from an agentic AI system (e.g. data management, simulation solvers). The more pronounced this convergence will be, the more disruptive AI is going to be for engineering IT.

This means that software vendors, end users and IT teams must ask themselves the following questions: To which pole is my software component converging to? How will this change the requirements and value propositions for the software component? Which use cases should I tackle first to understand the new requirements and value propositions better?

As of now, it is very early days for the technology and nobody has the complete answer for these questions. All current use cases only scratch the surface of what will be possible with agentic AI. However, we already can already see that even these early applications can generate tremendous engagement and value

In this chapter, we will look at representative use cases. We explain their value, current maturity level and discuss if it is a build or buy solution. Here is an overview of the use cases along with a short description of the target audience:

Coding

Companies that have at least one developer should definitely buy this.

Technical Support

Companies with more than 5 support colleagues should start building this.

Data analysis

Companies with sensor / fleet data platforms should start integrating this.

PLM & Knowledge Management

All companies that design complex products should start building this.

Computer Aided Design

Companies that use CAD should consider buying this.

Requirement Analysis

All companies that design complex products should start looking into this.

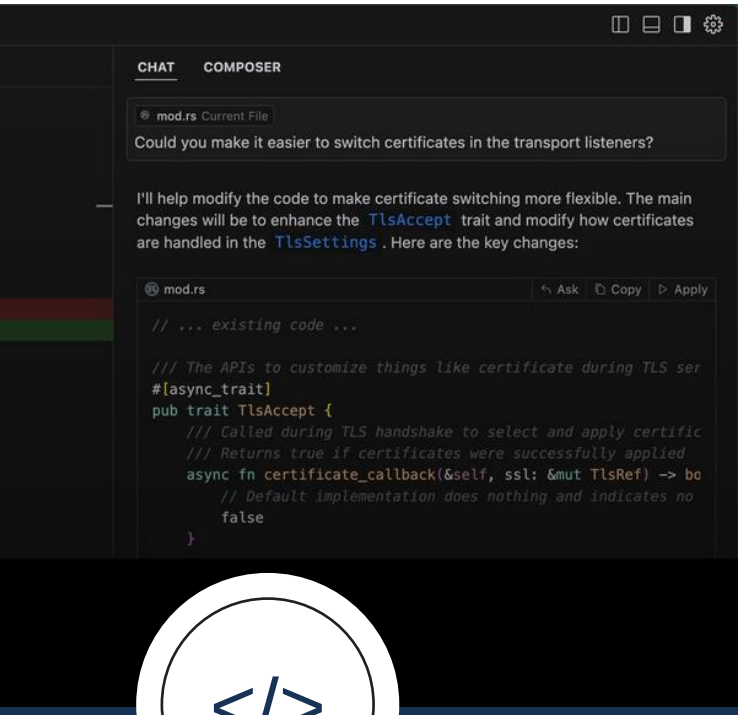
Machine operations

Operators and manufacturers of complex machines should look into this.

Motorsports

All motorsports teams with data scientists should start building this.

Coding Assistant



GENERAL LEARNINGS

It is interesting that early adopters of coding assistants started working with the generalized ChatGPT-Style user interface. Using this simple interface to prototype a solution is a great strategy for many applications. However, the use case also demonstrates that a deep integration into a dedicated UI such as Cursor AI is important to unlock real value and to convince a majority of users to use the tool.

Cursor AI is also a great example how users can interactively select the context for the assistant to create more reliable and trustworthy results.

INTRODUCTION

Because LLMs are exceptional at translating between languages and summarizing large chunks of information, it should be no surprise that coding assistants emerged as a first killer use case for the technology. When ChatGPT was released, people immediately started using the chat interface to generate code and then simply copied the code to their development environment. The next step was marked by the launch of Github Copilot, which integrated a specialized LLM into the Visual Studio Code and Jetbrains IDEs. Since then, different tools have been launched that provide even better assistance by integration LLM assistance deep into a complete development environment: Anthropic launched Artifacts to improve the coding capabilities of their Claude chatbot and just a few months later, OpenAI introduced a similar concept called canvas. Meanwhile, the team at Cursor built an AI-first IDE that reportedly grew to $50 Million in ARR within half a year and that is, more importantly, already loved by many developers (including colleagues here at Renumics). Within the industrial application of PLC programming several companies such as Siemens and Beckhoff already launched offerings that are tailored towards the needs of automation engineers.

WHO SHOULD USE IT?

Although certain development tasks such as frontend work benefit more from current tools, there is a good ROI for every software developer.

BUILD OR BUY?

Clearly something you buy.

Technical Support

GENERAL LEARNINGS

RAG-based chatbots are typically optimized for both accuracy and usability. The accuracy is primarily influenced by the available data (not the LLM!). Data quality can be improved by writing better source documents, implementing better parsing (especially for tables) and including additional information such as glossaries or data catalogues.

In practice, the user interface is more important for user adoption than accuracy. Here, two functionalities are key: Quickly displaying relevant sections in the source files and interactively selecting the relevant context. We are building the open source UI framework Lexio that provides these functionalities.

INTRODUCTION

Customer support chatbots have been around for a long time. However, RAG-based systems are much more powerful and easier to set up than traditional, heavily scripted systems. By leveraging the background knowledge of the LLM, modern technical support assistants can answer complex questions based on available manuals, guidelines or support documentation.

Most automotive, engineering and machinery companies have already started to introduce AI assistant for customer support. One example is the Ask ARBURG system (left). For teams with larger support and pre-sales teams, it makes sense to first start with an internal system. There are two advantages for this approach: First, internal LLM-systems are easier to implement due to lower security requirements. Additionally, it is easier to work with internal users when iteratively improving the system.

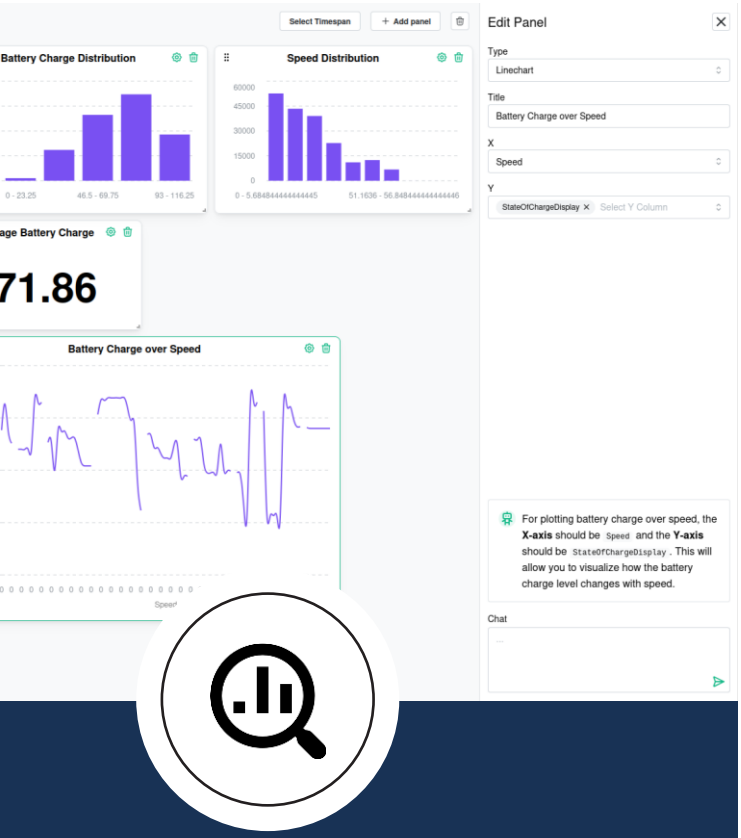
WHO SHOULD USE IT?

AI-based technical support assistants are valuable for organizations with complex products and a sizable team for support and pre-sales (10+). The use case is also interesting as a starting point towards other applications such as knowledge management.

BUILD OR BUY?

Our customers often do a first prototype using OpenAIs Custom GPT or Microsoft Copilot Studio. From there, it makes sense to adapt available open source software to the specific needs of the use case. This typically means processing different data sources, building prompts & glossaries as well as tuning the user interface.

Data Analysis



GENERAL LEARNINGS

AI assistants for data analysis can't replace experienced data analysts. But: They can provide domain experts like test engineers or machine operators with completely new capabilities. This democratization has tremendous value, because it allows to answer important questions within minutes instead of days. Looking for value in use cases where LLM-based assistants can disrupt long processes through the democratization of data access and tool use is often a good strategy.

INTRODUCTION

For more than a decade, manufacturers of complex mechatronic products have been working hard to integrate test, simulation, and operational data holistically (*digital twin*). However, these datasets are only used in a limited and fragmented way for product development and operations. This is primarily due to the often-missing semantic description of the data, insufficient data quality, and the fact that engineers frequently lack the programming skills needed for data analysis and interactive visualization.

AI assistants for data analysis can overcome these hurdles: They can contextualize sensor data from documentation, translate natural language queries into machine-readable database queries, or create data visualizations. Most general-purpose BI applications like Microsoft PowerBI have already started to integrate AI assistants. We have also seen in customer projects that integrating specialized assistants for engineers (left) can bring great value.

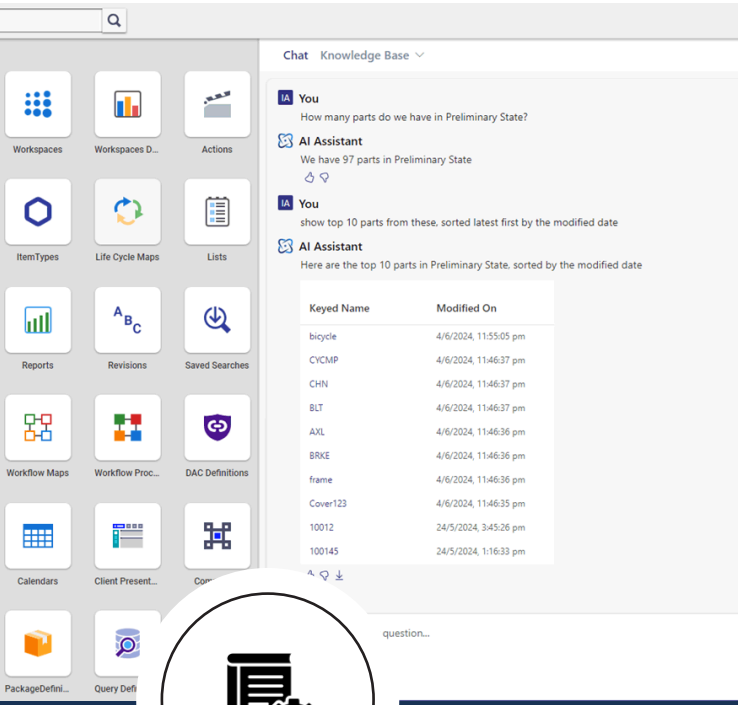
WHO SHOULD USE IT?

Organizations that have been building data infrastructure for product-related sensor data such as testing data, fleet data or machine data should start integrating AI assistants. This is true for large cloud-based systems as well as for smaller infrastructure that still relies on file-based systems.

BUILD OR BUY?

Generic BI tools are often unsuited for engineering applications: They are difficult to integrate with existing infrastructure and engineers often have different requirements for data analysis (e.g. raw time series visualization). That is why we have developed standard components for engineering data analysis that can be integrated into existing toolchains. Feel free to get in touch if you are thinking about integrating an AI assistant in your data analysis platform or toolchain.

PLM & Knowledge Management



GENERAL LEARNINGS

Engineering knowledge management is one of the areas, where the user experience will change dramatically thanks to LLM-based assistants. Currently, the best option is to build new web-based systems that provide a UI for the PLM backend. It will be interesting to see if this trend continues. Satya Nadella from Microsoft is convinced that agentic AI systems will generally kill SaaS apps. A vendor-agnostic IT strategy is important to prepare for this scenario.

INTRODUCTION

Engineers spend a lot of time looking for information: Requirements from a previous project, best design practices, documentation for a component etc. What makes this task especially difficult is that documents and information artifacts are typically spread across many systems like Sharepoint, Wikis, ERP/PLM/MES Systems or other databases. Also, traditional keyword-based search tools often fail to produce relevant results as they are not able to capture the meaning and context of most queries.

LLM-based systems can help engineers to quickly find what they are looking for across different systems. Most PLM providers and implementation partners have developed demo applications for this use case (e.g. ARAS AI assistant (left) or Teamcenter AI chat). Because most engineering data is already available under APIs, it is also straight forward to build web-based, stand-alone systems for this purpose. The main advantage of this approach is that different data sources can be easily integrated.

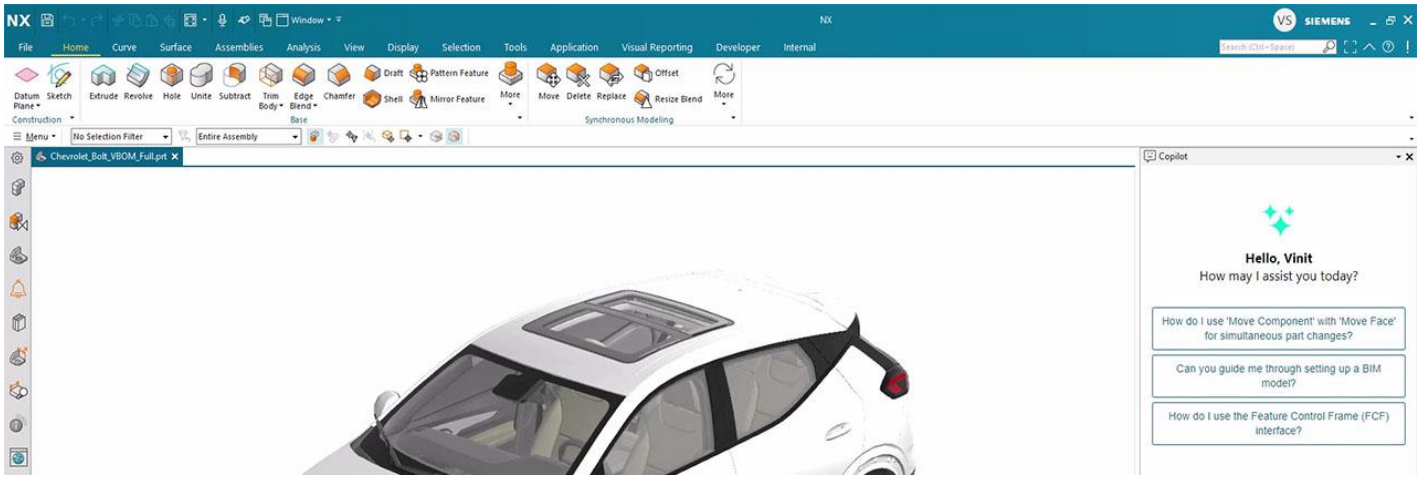
WHO SHOULD USE IT?

Companies that design complex products and have processes in place and systems in place to manage this complexity.

BUILD OR BUY?

We have seen that Microsoft Copilot Studio can be a good starting point for Sharepoint-based data. However, the UI (no display of sources) and the accuracy of this approach is limited. A stand-alone version based on open source software is future-proof and currently the best option in terms of performance.

Other Use Cases



Computer Aided Design

Using Computer Aided Design software (CAD) requires not only a lot of experience, but also repetitive manual work. The Siemens NX X assistant not only guides the user through the software, but can also automate small design tasks. We are also seeing new solutions like ZOO that directly provide text-to-CAD functionality. This paves the way for designing mechanical parts without any prior CAD knowledge.

Requirements analysis

Complex products are driven by requirements that are developed collaboratively by different teams across various companies. Creating, understanding and revising these requirements is a lot of manual work. LLM-based assistants like the IAV devpanion can compare requirements, find missing details or check for consistency. This tools help to speed up offer processes, reduce manuals work and to reduce errors.

Machine operations

Operating complex machines requires a lot of experience. Even then, optimizing machine settings or solving machine issues is difficult. AI assistants such as the Siemens Industrial AI Copilot or the Tulip Frontline Copilot can help with these tasks. This works by providing access to relevant documentation, best practices and sensor data through a natural language interface.

Motorsports

Motorsports has always been at the forefront of using data analysis and machine learning in engineering. However, sifting through the large amounts of available data to optimize car dynamics or for driver coaching takes a lot of expertise and manual effort. AI assistants like MOVEcenter can help by combining different data sources (incl. video) into an agentic platform that helps with reporting and data analysis.

The Road Ahead

In this white paper, we have learned how LLMs and AI assistants fundamentally work. We have seen that they excel at using background knowledge then answering queries or performing simple tasks. And we have learned how existing industrial use cases map to the core challenges we are facing in the development and operation of complex electro-mechanical systems.

If you are still not convinced that AI assistants will be universally used in engineering and manufacturing, I suggest to revisit the „*Coding Assistant*“ use case: We have gone from „nice to have“ to „most developers using it“ within 18 months. It is very obvious that these systems will become indispensable within the next year and might even replace certain developer roles. And I‘m saying this not just because Mark Zuckerberg says so, but because I am seeing this in my own workflows and in our team.

If you have understood the importance of the technology for your team and your organization, you are probably thinking about how to get started. I hope this white paper helped you to get a basic understanding of the fundamentals and some ideas what to do next.

BUILD or BUY

You won‘t be able to buy the perfectly tailored AI assistant who covers all your use cases and integrates perfectly with existing tools and workflows: Not now and not in the future. However, you also don‘t want re-invent the wheel by developing standard components for this rapidly evolving technology. Managing the build vs. buy approach is walking a tightrope and must be decided on a case-by-case basis. Here are some fundamentals that I considered important: Getting started early and developing a good understanding of the technology is paramount. Leveraging open source makes sense, because the field moves so fast (even big tech is doing this). Standardization is important and open standards help to achieve this without vendor lock-in.

HORIZONTAL vs VERTICAL

Implementing AI assistants horizontally in your organizations means to roll-out a general-purpose solution like ChatGPT or Microsoft CoPilot to many users. While this can make sense to understand the technology and for prototyping, we have found that focusing on specialized workflows is what drives real value and change (vertical implementation). This is because currently only tailored solutions offer the performance (UI and accuracy) that deliver significant value-add. I suggest to look through established use cases such as these presented in this white paper and to examine their value for your organization.

THE GLOBAL CHALLENGE

Companies that develop and manufacture complex electro-mechanical products are currently facing several challenges: Shorter development cycles, a pressure to localize the supply chain, the increasing significance of software & services just to name few. With their core promise to overcome data silos and reducing manual efforts, AI assistants will be a key technology to increasing the competitiveness in the coming years.

THE DEATH of SaaS

LLM-based assistance have already started to change how we interact with software. Cursor AI is demonstrating this in the coding IDE space. It remains to be seen of all software will integrate an assistant or will be an API. But there is growing evidence that this process has already started: Payment provider Klarna replaced the Salesforce CRM with an in-house AI system and Satya Nadella proclaimed that agents will be the death of SaaS.

Getting Started

The hype around AI is a double-edged sword: While it brings management attention to the topic, it also breeds unrealistic expectations. Together with our customers and partnes, we deeply believe that AI will change everything. However, we are not addicted to fancy tech or AI magic. We focus on building the best tool for the user and prefer the pragmatic solution that works over shiny demos. Please have a talk with us if you want to get a grounded perspective on your use case.

“The secret of getting ahead is getting started.
— Mark Twain

Coffee Talk with us

Book a coffee talk on Calendly

Write me an e-mail: stefan.suwelack@renumics.com

Our Github

Lexio UI for RAG

RAG-Demo

More resources

Industrial AI Canvas

Blogs, Papers, White Papers