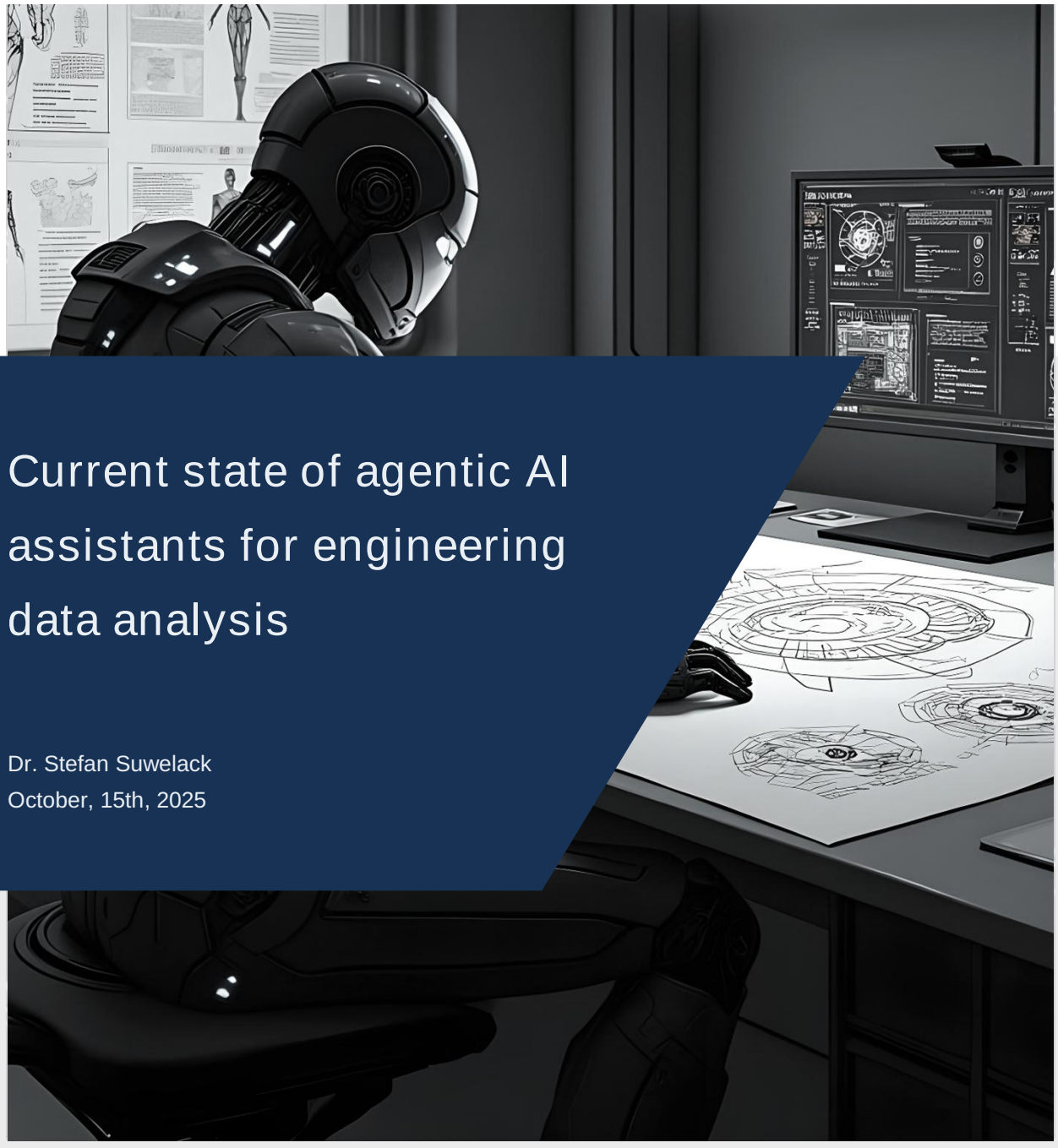
+49 173 805 3434
stefan.suwelack@renumics.com

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



Current state of agentic AI assistants for engineering data analysis

Dr. Stefan Suwelack
October, 15th, 2025

Table of Contents

Data analysis: A high ROI applications for agentic AI assistants in engineering

Agentic AI assistants have become an indispensable tool for software development. The same will happen for all engineering disciplines: Engineers will have agentic AI assistants by their side who can help to find necessary information and automate manual workflows. Engineering data analysis is a field of work that can especially benefit from the technology available today.

Introduction	03
Use Cases	05
Fundamentals of Agentic AI	11
Agentic patterns	12
Data infrastructure	13
Best practices	14
The Road Ahead	15

Introduction

In the beginning of 2025, I remarked that J.A.R.V.I.S-like AI assistants are no longer science fiction but are quickly becoming a reality. Just two weeks later, on Feb. 3rd 2025, Andrej Karpathy coined the term vibe coding. Today, most software engineers use agentic AI assistants in their day-to-day work. It is not yet clear, where these assistants can bring the most value (and where they can even be dangerous), but the fact is: Since the launch of Github Copilot just four years ago, AI assistants have become ubiquitous in software engineering.

This begs the questions: Why is this not the case in other engineering disciplines? And when do engineers who develop complex electromechanical products get their AI assistant?

There are several major reasons why Large Language Models (LLMs) work so well for programming: Not only is code naturally text-based, but it is also typically contained in a repository that is relatively easy to index. Thanks to the open-source movement, there is also a lot of data available for LLM providers to train their models on. Finally, software engineers do 90% of their work in a single integrated development environment (IDE)

In contrast, the situation for engineers working on electromechanical products is completely different: There is a lot of multi-modal data available: Geometries, time series data, videos or drawings. The data is also typically not contained in a single system, but engineers must juggle many different data sources and tools. The data is also almost exclusively proprietary, which means it is not easily accessible for LLMs during the training phase.

All these differences are major hurdles for AI adoption in engineering. However, the agentic AI paradigm can help to overcome these challenges. Before we explain why engineering data analysis is such a good use case for agentic AI, let's briefly define what agentic AI actually is:

Agentic AI is a catch-all term denotes several different design patterns that essentially describe how to use LLMs in a larger workflows. Of these design patterns, three are especially noteworthy:

1. Tool use: The LLM can execute code functions or use APIs to gather information, process data and take actions.
2. Reflection: The LLM is used to judge preliminary results to iterate on steps such as data gathering or summarization.
3. Flow control: The LLM is used to control the flow of the execution by making and executing plans to achieve a given goal.

In practice, this typically means running an agent loop that coordinates different sub-agents and tool call patterns. The core challenge in this setting is to optimally tailor the model context.

Let’s come back to engineering data analysis. Most engineers must deal with time series data that either originates from physical sensors or from simulations. Visualizing, understanding and reporting on this data is a cornerstone in product development: From understanding requirements through fleet data analysis to optimizing systems based on testing data and data-driven production.

In many scenarios, engineering data analysis is still a complicated manual process. The challenges that engineers face are representative for the whole product development process:

1. Data silos make it difficult to bring all information into a single view. This is true for both raw time series data (e.g. testing/simulation data) as well as additional metadata.
2. Highly-specialized experts control access to data sources. Most engineers do not have the skills to use current big data stacks (e.g. SQL) and have to ask dedicated data analysts for help even in simple cases.
3. Manual work is needed to derive insights and create reports. Although scripting is used extensively, it is often too expensive to maintain these scripts for many use cases due to edge cases and data evolutions.

Agentic AI can help to overcome these challenges: LLMs can not only help to semantically describe even *messy* data sources, but it can also use these descriptions to provide context to the data. This context then enables LLMs to transform natural language questions into structured queries. This means that we can give engineers access to data even though they do not have deep programming skills. And we can empower engineers to define flexible automated workflows in natural language. This enables experts to quickly infuse their domain knowledge into flexible workflow automations.

This white paper is my attempt to explain how agentic AI for data analysis works and why it is such a high ROI use case in engineering. This white paper 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.

Karlsruhe, 14.10.2025
Stefan Suwelack

Use Cases

Analyzing time series data is everywhere in engineering: From simulation, to testing to fleet and production data. However, engineering data analysis currently lives in two worlds:

One world is the big data world: Data often resides in a lakehouse or database. Data experts write data transformations in SQL and use BI tools to create dashboards that drive insights. Typically, these insights are based on aggregate data (e.g. histograms). Engineers can't typically change the dashboard by themselves or dive deeply into raw sensor data.

The other world is the small data world: Data is stored in files on a network share. Engineers use established desktop-based tools to visualize and analyze the data, often diving deep into the time signals. It is typically difficult to compare many measurements or data from different sources such as simulation and testing data.

Combining a modern data stack with agentic AI allows to bridge these two worlds: Engineers can build elaborate dashboards themselves by using natural language instead of SQL queries. It also becomes possible to connect different data sources into a single environment, because the world knowledge encoded in the LLM helps to automatically fuse different data models (e.g. mapping different channel names with the same meaning). If coupled with powerful web-based visualizations, this approach allows engineers to seamlessly switch between aggregate views on many data sources (big data world) to in-depth root cause analysis on a concrete sensor signal. This allows to break open data silos, democratize data access and to build flexible workflow automations. Here are eight use cases that exemplify these advantages:

Fleet data analysis

Engineers can ask data questions in natural language.

Endurance testing

Errors and events during testing can be summarized analyzed quickly.

Root cause analysis

Analyzing 100s of channels at once speeds up time-to-insights.

Lab measurement reports

Typical analysis and reporting workflows can be automated.

Computer Aided Engineering

Static reports can be replaced by context-dependent analysis.

Model based systems engineering

The complexity of MBSE can be simplified by providing context.

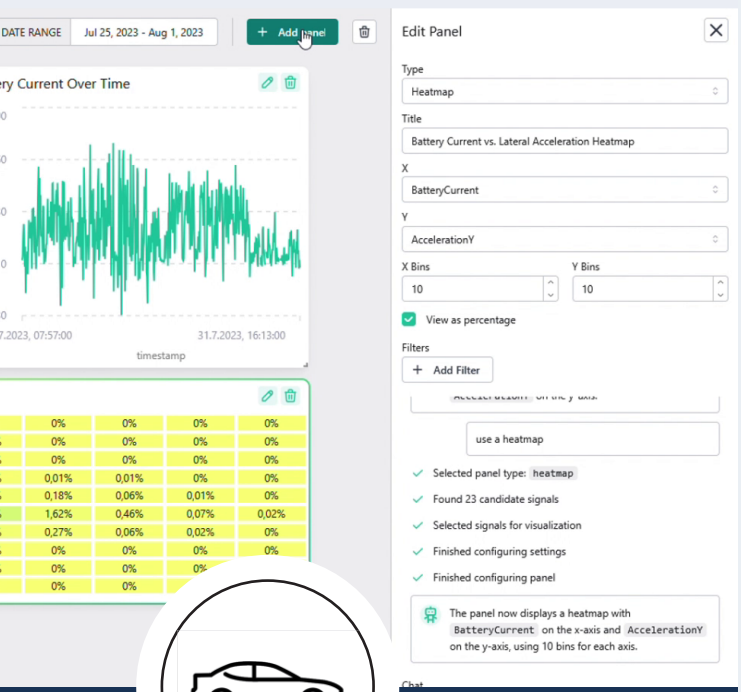
Production data analysis

Questions on complex production systems can be answered.

Motorsports

Insights for driving and set-up optimizations can be derived quickly-

Fleet data analysis



EXAMPLE WORKFLOW

Fleet data is currently saved in a data lakehouse. A description for the sensor data (data catalogue) has been automatically created from existing documentation (DBC-files, PDF documents) through an LLM.

An agentic AI system can connect to the lakehouse to query data. Engineers can create custom dashboards based on standard visualization components (e.g. histograms, heatmaps, linecharts) within minutes. The semantic descriptions enable the engineers to formulate the queries in natural language (e.g. *show me the distribution of the HV battery current*).

BACKGROUND

Over the last couple of years, most manufacturers of electromechanical systems have invested heavily to collect day-to-day telemetry data from their systems in the field. This process involves setting up appropriate instrumentation, communication channels and big data systems to process and store the data.

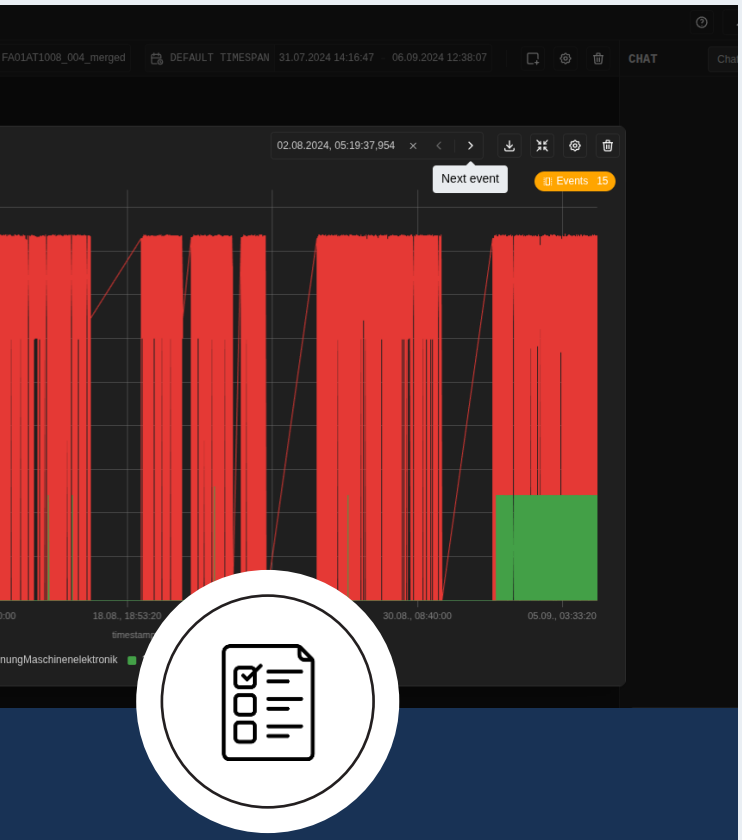
Now data teams have the possibility to query this data and build dashboards to answer recurring questions. However, engineers typically lack the SQL and programming skills to create visualizations for ad-hoc questions or in-depth analysis. Thus, they must ask their data experts to create a new dashboard for every one-off question. This process typically takes a few days.

Furthermore, the standard BI tooling used for this application typically does not offer the visualization and transformation capabilities that engineers to really dive deep into the data. For this reason, telemetry data is often additionally maintained in a native format (e.g. MDF in automotive) that can be opened with established desktop-based engineering tooling.

VALUE

Agentic AI assistants allow engineers to directly ask questions on their fleet data instead of asking data colleagues to build a custom dashboard for them. This can cut the time to answer a question from days to minutes and can save valuable time from data analysts.

Endurance Testing



BACKGROUND

Endurance testing is an important step within the product development cycle. Because the testing runs for a long time, engineers will only check interesting events during the test run (e.g. errors). To facilitate this process, many teams have already built scripting solutions that create automated reports for each endurance test run.

Engineers use this PDF- or HTML-based reports to identify relevant events. Then they use interactive visualization tooling such as NI Diadem to interactively investigate them.

Within this toolchain it is not directly possible to re-group events by their semantics. It is also not easily possible to aggregate event statistics or compare many testing runs side-by-side. This makes the process is time intensive. As a result “minor” events are often not really investigated.

Often, engineers could provide better report automations based on their specific knowledge. However, most engineers lack the necessary programming skills, and these automations would need to change often to accommodate for intra-experiment variations such as changing channel names.

EXAMPLE WORKFLOW

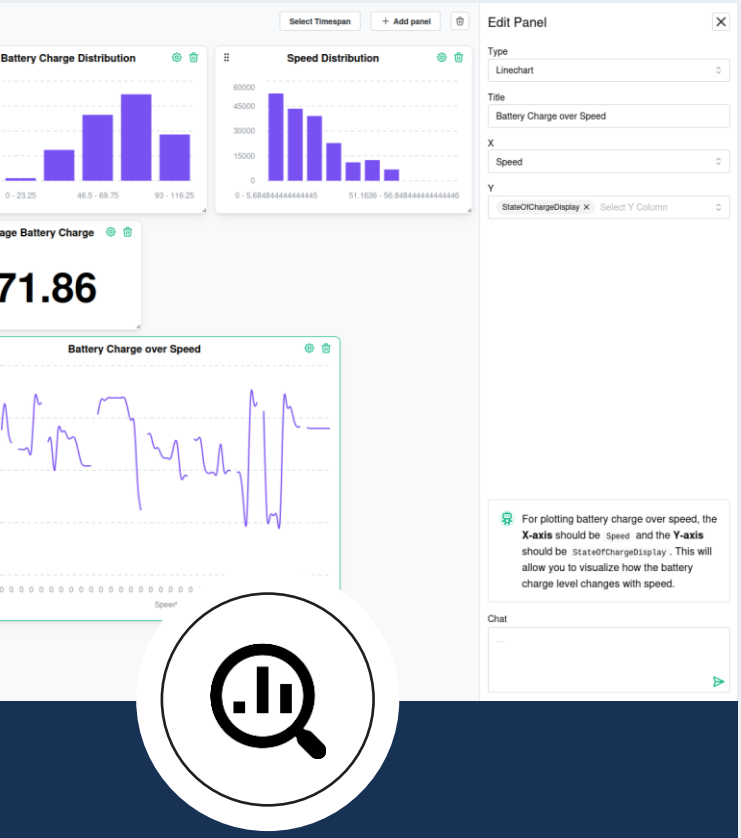
Test bench data is currently stored on a fileshare as a collection TDMS data format. By converting this data into a Parquet-based standard format, an SQL query endpoint can be established on the files.

An agentic AI system uses this query engine to analyze the endurance testing data and identifies critical events based on detailed descriptions given in natural language. When stepping through these events, engineers can pull in additional data from other runs and can trigger in-depth analysis workflows based on a simple command.

VALUE

Agentic AI systems can automatically prepare endurance testing reports based on workflow descriptions in natural language. Engineers can then start to investigate critical events systematically and directly from the report, saving time and increasing testing quality.

Root Cause Analysis



EXAMPLE WORKFLOW

A ticketing system provides engineers with problems from an automotive testing fleet. When a new ticket is created, an agentic AI system automatically downloads the attached MDF file and analyzes the problem. The Analysis is guided by workflow instructions that individual experts can provide in natural language.

Engineers can the interactively explore the preliminary report. By pulling in information about the connection between different components and automatically analyzing hundreds of channels for signal variations, engineers can quickly get to the root of the problem.

BACKGROUND

Finding the root cause of errors in testing and telemetry data is a core engineering task. Often, this process starts with a ticket created by an operator or testing personnel. The engineer starts the analysis process by opening the ticket and downloading corresponding data traces.

Upon reading the ticket and additional documentation, the engineer opens the data in an interactive data visualization tool. For many complex machine such as cars, the trace data can easily contain thousands of channels. Thus, engineers have to use their personal experience to find a first hint where to look for the error. This often means to ask around for help.

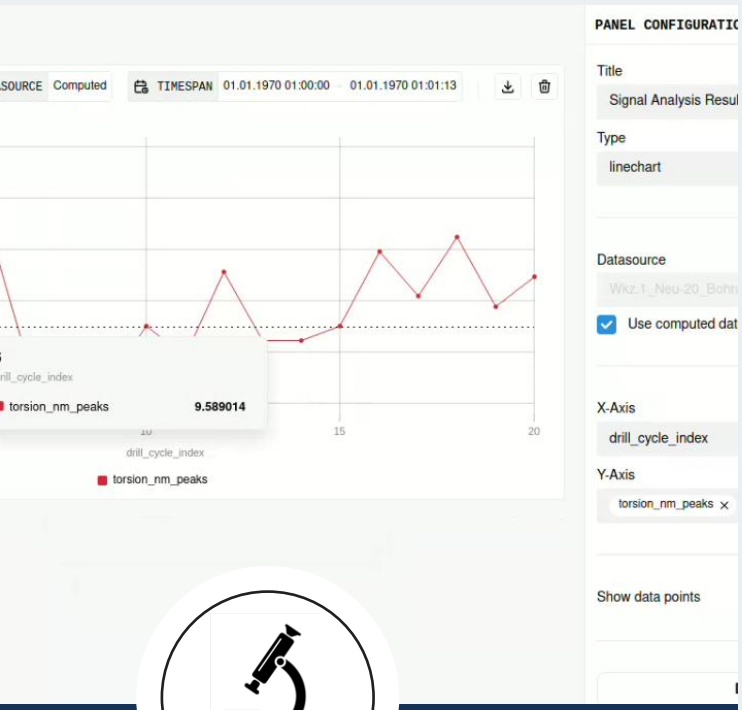
Some engineers have built scripts to identify important events in the data (e.g. signal toggling before error), but these scripts are expensive to maintain if the error traces vary in terms of channel names.

If engineers want to pull in other sources of information such as additional product information or data format specifications they have to do it manually.

VALUE

Agentic AI systems can automatically prepare a preliminary root cause analysis based on a ticket description. By connecting to different data sources, they can give engineers a complete 360° view on the problem within minutes. Engineers can also use the agents to interactively explore hundreds of channels at the same time with minimal efforts.

LAB MEASUREMENT REPORTS



BACKGROUND

R&D labs generate valuable measurement data. In the analysis and reporting phase, engineers must often fuse data from different measurement devices.

Because many devices come with their own proprietary software, data fusion is difficult. After post-processing the results manually in the proprietary, desktop-based software, the preliminary results are exported to Excel. Here, final results and charts are calculated, and a report is created.

As such a manual process can take weeks depending on the amount of measurement runs, teams have already started to create Python-based scripting workflows to automate the reporting process. However, as not all engineers are fluent in Python, many colleagues do not possess the skills to tailor these scripts to their needs. Consequently, they stick to time-consuming, Excel-based workflows.

EXAMPLE WORKFLOW

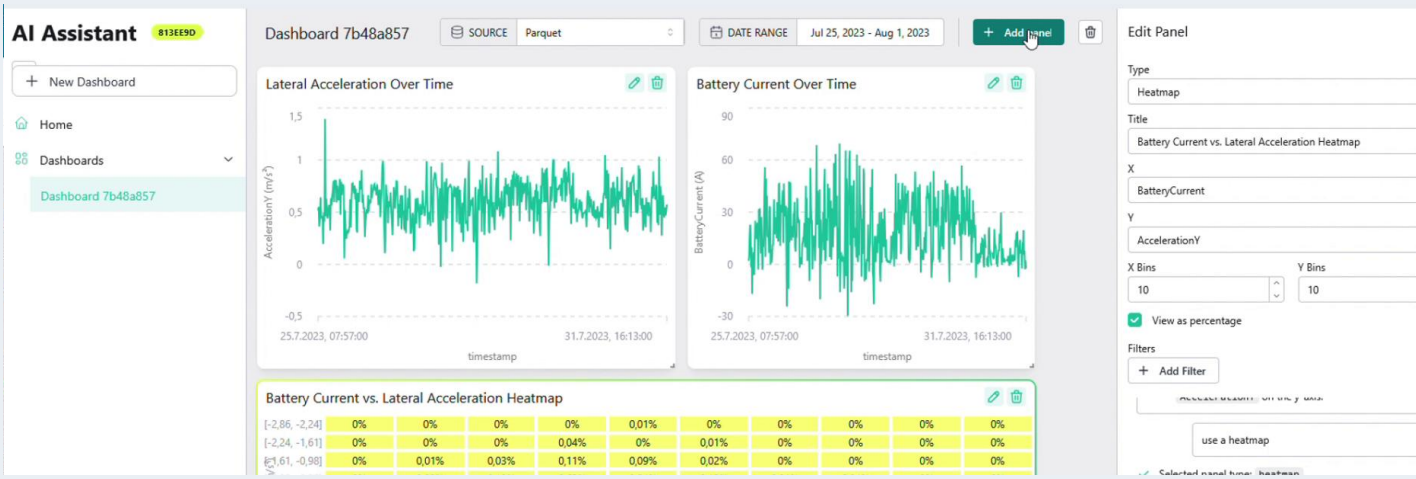
Lab measurement data is saved in a proprietary, text-based data format. By ingesting the data into the standard Parquet format, an agentic AI system can perform standardized signal analysis and visualizations workflows.

To facilitate specific report types, engineers write small workflow descriptions that includes critical measurement parameters to perform automated signal segmentation and KPI extraction. This allows the whole reporting workflow to run in minutes where the old manual process required days of work.

VALUE

Raw measurement files are ingested into an agentic AI system. Building upon Python scripts that are created by experienced power users, engineers can create their own processing workflows described in natural language. This gives engineers the possibility to infuse critical domain knowledge to simple signal analysis algorithms (e.g. threshold-based segmentation). Consequently, analyzing lab measurement data and report creation can be highly automated.

Other Use Cases



Computer Aided Design

Simulation reports are often automatically generated from post-processing scripts. While these PDF or HTML reports are enough for standard questions and compliance purposes, they are unsuitable for analyzing specific questions or variant comparisons. Agentic AI systems can provide context-dependent, interactive reports with a single command. They can also connect to established CAE tools to further investigate initial findings.

Model-based systems engineering

Model-based systems engineering provides a powerful method to design complex systems and is increasingly used in the development of complex electro-mechanical systems. However, due to the complexity of the models, MBSE has a high barrier of entry, hindering its adoption. Agentic AI systems can quickly extract core insights from a SysML-based simulations or can explain components in SysML specifications.

Production data analysis

Most production data that is generated today is stored in big data systems. Data teams have used these databases to create dashboards for different personas and applications. This has created a situation where many stakeholders can't find the specific information they need, because there are just too many dashboards to look through. Agentic AI systems can answer direct questions in minutes and help stakeholders to create their own personal dashboards.

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. Agentic AI Assistants can help by combining different data sources (incl. video) into an agentic platform that helps with reporting and data analysis.

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.

A core property of an agentic AI system is the ability to interact with existing tools, data sources and systems. This pattern is called *tool use* and enables AI systems to integrate into and control existing engineering workflows.

Hype vs. Reality

Reading through LinkedIn, it is sometimes difficult to separate hype from what really works. I highly recommend the Github repository “12 factor agents” as a source for best practices when building an agentic AI system. In the introduction the author Dexter Horthy already sets the tone on what to expect for production-grade agentic AI systems: *[…] most of the products out there billing themselves as “AI Agents” are not at all agentic. A lot of them are mostly deterministic code, with LLM steps sprinkled in at just the right points to make the experience truly magical.*

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.

Agentic Patterns

The most simple agentic system just runs a loop in which one agent has the option to call additional tools, produce output or terminate the loop. This can be extended by giving the option to hand off the current context to a another, more specialized agent. This might be an anomaly detection agent, for example that has specific prompting and tools to do this task well. Starting from this simple mechanisms, a lot of patterns are possible. When planning an agentic architecture, we always have to balance between flexibility and robustness. This is exemplified by two of the most important patterns: Code generation and tool calling.

Code generation

The most straight forward way to do data analysis with LLMs is by letting the model write Python code to perform the analysis. Although this approach is simple and flexible, it comes with huge disadvantages:

First of all, code generation and execution takes a long time. This diminishes the interactive experience for the user. More importantly, code generation is not deterministic and can quickly become unreliable. Finally, arbitrary code generation and execution can become a security nightmare.

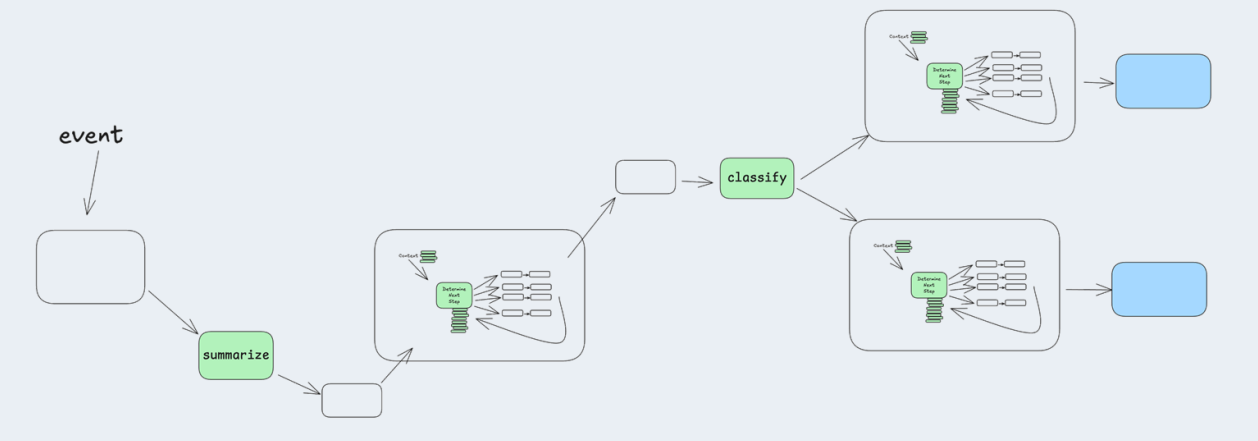
In practice it is often a good idea to avoid code generation as much as possible and only use it where absolutely necessary.

Tool calling

The idea behind tool calling is to provide the LLM agent with pre-defined functions (e.g. for data queries and transformations). The agent's task is then to determine the right order in which to call the tools and to parameterize these functions.

While this approach is not as flexible as code generation, it allows to re-use highly optimized functions and scripts. That is why tool calling generally result in much faster and more reliable workflows.

Bi-directional tool calling (e.g. via MCP) also allows to quickly integrate existing software into an agentic AI system.



Data Infrastructure

General-purpose LLMs can't directly work with time series data. Instead, agentic LLM systems use tool calling for data queries, transformations and signal analysis. This must be supported by an appropriate data infrastructure. Typical operations that should be supported is to be able to do conditional filtering over different channels, perform calculations or aggregate data over signals. What makes it especially difficult is that we typically want to do these operations at interactive speeds (< 1 minute). This is a big challenge for classical engineering stacks and established big data stacks alike. Fortunately, a modern data stack is emerging that is perfectly suited for handling engineering time series data.

Are you really working with big data?

Data from a test fleet can easily hit several dozens TB. So, we are definitely working with big data and have to bring out the big guns like Apache Spark. Right? Not so fast! First, this data might be 50TB, but there are also 20k channels. And we are typically only looking at a handful of channels at the same time. Also, we are most often interested only in very recent data. And maybe even only from a certain group of vehicles or with a certain software version. Once we have all these filters applied, we probably end up with an amount of data that easily fits onto a single machine.

By leveraging this specific structure of the sensor data in an hierarchical data format, we can efficiently select only theses subsets that are relevant. By using modern data infrastructure based on the Parquet container file format and single-node query engines like DuckDB, we can store our data cheaply in blob storage and query it interactively with low computational cost.

Parquet file format

Parquet has emerged as the de-facto standard format for tabular and time series data in the big data worlds. It is easy to use and has four major advantages over legacy engineering data formats:

1. Parquet is a first class citizen of the Apache Arrows ecosystem and supported by nearly every data tool available.
2. Parquet has very good integrated compression mechanisms.
3. Parquet stores indices over row groups that query engines can use via filter push-downs for blazing fast queries.
4. Parquet can provide versioning via several existing lakehouse solutions.

DuckDB query engine

Traditional query engines like AWS Athena or Apache Spark are built to execute long-running queries on large amounts of data. Their superpower are the methods and abstractions that help to make these queries parallelizable for running them on a large cluster. However, this comes at a cost: Large startup times, big overhead and complicated infrastructure.

DuckDB is specifically build to run on a single machine in the application process. That means we need no additional server-side system for the database. That is why DuckDB is as easy to use as a Pandas function, but gives us all the superpowers of a complicated big data system.

Best practices

#1 Avoid code generation

The most straight forward approach to perform AI-assisted data analysis is to write Python code that solves the question from the user. Although the approach is very flexible, it is very difficult to make this method reliable. Generating and executing code also takes very long. Using tool calling for data queries, transformations and visualizations is not as flexible, but very robust and can be optimized for performance and security.

#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 interactively visualize results is the most important part of an AI assistant. It is in particular important that the time series charts are engineering-grade.

#3 Optimize the data schema

The performance of the query engine is a very important factor for both the overall usability and the compute cost of the AI system. Generally speaking, interactive queries should run in under a minute for optimal user experience. Often, the data schema has to be optimized to achieve this performance. It can also make sense to re-evaluate the choice of the query engine / database for these interactive workloads.

#4 Use data enrichment

A core value proposition of agentic AI systems is the ability to connect different data sources. This does not only mean to be able to programmatically connect to the appropriate API, but also to store the semantics of the data source. Enriching the data (e.g. channel descriptions) with an LLM from existing documentation and specification is cheap and works surprisingly well.

#5 Context control

Tailoring the context of the LLM is the name of the game in every agentic AI application. In the context of data analysis this concretely means: Determining relevant data sources and signals, selecting appropriate agents and tools for the query and representing the overall state of the dashboard. The critical part is to find out which of these selections should be done automatically and when the user should take control.

The Road Ahead

Engineers will soon always have an agentic AI assistant by their side. This AI can see the full context of the engineer´s work: All software tools and documents that are relevant for the current task. Engineers can use the AI to quickly pull in new data into this context from arbitrary enterprise data sources. They can also define automation workflows in natural language based on their specific domain expertise and preferences. These automations can seamlessly interact with existing tools and systems through the model context protocol (MCP).

Agentic AI systems can connect to existing tools and data sources via the model context protocol (MCP) and act as MCP hosts. Existing systems can also integrate the assistant via an API.

MCP – the holy grail?

Connecting all data sources in an enterprise has been a long-standing dream. We all know that the data silos within an organization are a major bottleneck in product development.

The model context protocol (MCP) was invented by Anthropic and is an open standard to connect LLMs to other tools, systems and data sources. It is described by Anthropic as *USB-C for LLMs.* In other words: It promises to solve the enterprise data integration problem once and for all.

This fundamental promise relies on the fact that the flexibility of LLMs can deal with API changes and semantic meaning much better than any previous solution.

Avoiding Clippy 2.0

Many software vendors are currently rushing to integrate AI assistants into their products. This is often seen as another add-on to provide incremental value (e.g. better help functionality) and not an AI-first integration.

It is important to note that users do not want a Clippy-like assistant integrated into every software tool they use. Instead, they want to talk to a single assistant that knows their preferences and workflows.

To avoid building a Clippy 2.0 assistants, it is important to think about how a tool integrates into the future agentic AI ecosystem.

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 partners, 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. We have built several successful production-grade Agentic AI systems for leading companies in automotive, aerospace and machinery.

“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

Introductory White Paper

Building J.A.R.V.I.S - Current state of LLM-based assistants for engineering and manufacturing

More resources

Industrial AI Canvas

Blogs, Papers, White Papers