PROVEN TECHNIQUE(S)

Using ThingWorx with LLMs

Best Practice V2.0

Abstract

This document outlines best practices for integrating large language models (LLMs), specifically ChatGPT-4.1, with ThingWorx, PTC's Industrial IoT platform. It emphasizes practical application through a detailed walkthrough of an example ThingWorx project. This project showcases data insights as well as analytics insights powered by ThingWorx Analytics.

David Kessler
dkessler@ptc.com

*This edition builds on Version 1.1, co-authored with Brad Cline bcline@ptc.com

Contents

Best Practice Intended Use

Introduction

ThingWorx is a leading Industrial IoT platform that connects factory equipment, sensors, and enterprise systems to drive insights and operational improvements. It can integrate with a wide range of data sources – from industrial historians and OPC UA servers on the shop floor to SQL databases and RESTful web services – aggregating real-time and historical data from across the enterprise.

With the emergence of large language models (LLMs) like GPT-4.1, organizations are exploring how to leverage generative AI on top of their IoT data to extract deeper insights, answer natural-language queries, and recommend optimizations. Across industries – from Industrial Manufacturing and Automotive to Aerospace & Defense, High-Tech Electronics, and Life Sciences – the fusion of IoT and LLMs promises to optimize processes, improve efficiency, and drive innovation. The following outline presents best-practice guidelines for integrating ThingWorx with LLMs, focusing on use cases, data needs, system design, security, and validation.

Complementing ThingWorx's connectivity capabilities, ThingWorx Analytics brings embedded advanced analytics and machine-learning tools directly into the platform. This enables users to build predictive models, detect anomalies in streaming data, and generate prescriptive recommendations through an intuitive interface. It does all this without requiring deep expertise in statistics or data science. Customers in many verticals leverage ThingWorx Analytics to implement predictive maintenance strategies. These strategies reduce unplanned downtime, optimize asset utilization, and drive significant cost savings by modeling equipment behavior and diagnosing root causes of failures in production environments.

Use Case

This document focuses on a demo application that integrates ThingWorx with ChatGPT-4.1. We explain the components of the application and how they collectively facilitate communication between ThingWorx and the LLM through an API. Ultimately natural language questions from the user are answered in a similar fashion.

Although our example uses one LLM service, you can easily swap different LLMs or data stores. The "Data Insights" demo imports data into a ThingWorx Data Table, then extracts, formats, and prompts the LLM to generate natural-language answers. Meanwhile, the "Analytics Insights" demo lets you load other demo data into a ThingWorx repository, prepare Signals, Profiles, and predictive jobs, and craft prompts so you can request deeper analytical insights in plain English.

The methods outlined here are flexible. Even if your data comes from ThingWorx Value Streams, historians, or other databases, the overall approach remains the same. You simply extract the relevant data, add context, and feed it into the LLM. Likewise, if you choose a different LLM service, the main steps—API communication, data transformation, and response checking—will still apply with only minor changes.

A specific context for using ThingWorx and LLMs is for Statistical Process Control (SPC). In many factories, SPC is used to track when processes go "out of control" over time. With this setup, you can ask questions like, "What were the out-of-control events for Machine 5 last month?" The system can then pull together historical and real-time SPC data to show when and where rules were violated. This helps teams quickly spot problems and make improvements.

A further example is using ThingWorx with LLMs for Time Loss Analytics (TLA). TLA gathers data about downtime at various work centers. With this integration, you can ask questions such as, "Which work center lost the most time last quarter?" or "When did we experience the highest time loss today?" The LLM can then analyze the time loss data and provide a clear summary. This insight helps managers identify bottlenecks, reduce downtime, and boost overall productivity.

Intended Persona/Process

This section is for both technical teams and business users. For technical implementers, the document provides simple, step-by-step instructions to build an application using ThingWorx integrated with an LLM. This guide helps ensure the system is set up correctly, the data flows smoothly, and the connection with the LLM is secure. Business users, on the other hand, will find that the paper shows how this application can deliver real value. They should list their business needs clearly and work with the technical team to verify that the application meets those needs.

For instance, a manufacturing operations team might want to answer questions such as, "What were the out-of-control events for Machine 3 last week?" or "Which work center lost the most time last month?" These insights help reduce downtime, improve quality, and boost efficiency. While the technical team sets up the data connections and the LLM interface, business users need to check that the answers are accurate and useful. This teamwork ensures that the solution works well and meets business goals.

It is important to set up clear guardrails. One that this document focuses on is using the LLM to provide insights and not make changes to your system. Business users should treat its outputs as recommendations that must be reviewed before any action is taken. Another guardrail is to only send data to an LLM that has strong data protection. For example, Microsoft Azure OpenAI offers robust safeguards that keep your sensitive data safe.

Finally, best practices require close collaboration between technical teams and business users. Regular review sessions, pilot tests, and feedback loops help ensure that the application continues to meet business needs. With clear guardrails and open communication, organizations can successfully use a ThingWorx and LLM integrated solution to make better, data-driven decisions.

Intended Business Value

With more data becoming available, traditional data analysis tools and processes are struggling to keep up with demand for insights. Business users want the ability to get answers from the data without having to wait for the building of a custom report or exporting data to a different tool to dive into detailed analysis.

This document highlights research to enable users to use natural language to ask questions from their data even without having to build a new report, extract data to another system or build a custom application The AI Chatbot application can translate user prompts into the queries required to deliver the answer back to the user.

Additionally, the user can get answers directly from the data that could not be obtained through the existing application user interface. Traditionally, these answers could only be obtained through customization or by exporting the data to build a custom report.

Prerequisites and Caveats

Disclaimer:

Users are solely responsible for the use of and reliance on the demo application and the accompanying document which integrates ThingWorx with ChatGPT-4.1, a generative Artificial Intelligence (AI) tool. The project file for the demo application is provided for demonstration purposes only and should not be used in production without thorough testing.

Users acknowledge that PTC Inc. owns the intellectual property rights to this demo application. A limited, free, non-exclusive license is granted to users for demonstration and educational purposes only.

Due to the inherent unpredictability of generative AI tools, users further acknowledge that generated output may be inaccurate or harmful, and that users must apply human judgment and consider associated risks when evaluating generated output.

PTC disclaims liability for any losses or damages arising from the use of the demo application and this document. Users are responsible for outcomes from using the demo and document, and assume responsibility for complying with all relevant laws, including but not limited to data privacy, security, and the EU AI Act.

Your license to commercial versions of PTC products may not include all APIs needed to connect to AI tools and may require additional licensing.

***

Before you begin using this document, you should have a solid background with the ThingWorx platform. You need to be comfortable with creating and managing Things, Data Tables, Data Shapes, Mashups, and other entities within ThingWorx. A basic understanding of ThingWorx Analytics is helpful. In addition, some familiarity with large language models (LLMs) is important so that you understand how they work and how to integrate them with ThingWorx via APIs.

You must also have the proper access and permissions in the ThingWorx environment where you plan to build this application. This means your ThingWorx account must have permissions to create and modify entities as described in this document. Likewise, you need the appropriate access to the database, historian, or any other data source that holds the data used by the application. Without the necessary permissions and access, following the steps in this document may not be possible.

It is recommended to use an environment that has ThingWorx Analytics installed as well as ThingWorx Platform, to take advantage of the “Analytics Insights” content referred to in this document.

This document is intended for users working with ThingWorx and ThingWorx Analytics version 9.5.0 and above. If you are not on this version or later, some steps and features described here might not be available.

For more information on the prerequisites and necessary configurations, please review the following resources:

- Getting Started with ThingWorx
- Setting Up User Accounts in ThingWorx
- Assigning User Permissions in ThingWorx
- Connecting ThingWorx to Your Data Source
- ThingWorx Analytics Training

By ensuring you meet these prerequisites, you can follow the best practices outlined in this document more effectively and avoid common pitfalls during implementation.

Overview of Best Practice

Data Preparation

As a prerequisite to building an application like this demo, an appropriate data pipeline with required data engineering should be established. This data pipeline should ensure that the data to be queried by the application has been formatted and cleansed according to the organization's data standards and contains the data end-users will want to ask questions about.

Multiple data stores may be called to provide data to a TWX/LLM application. This demo application only relies on two stores, but the concepts and components illustrated in this paper can be expanded to accommodate more sources of data.

One store resides in a Data Table and is used by the "Data Insights" portion of the application. The fields in this data are:

- "id" (string): unique identifier for each record
- "DateTime" (datetime): hourly value from 12am November 1st, 2024 – 12am March 1st, 2025 with no missing gaps in time
- "asset" (string): has values "Machine1", "Machine2", and "Machine3"
- "status" (string): has values "running" and "idle". Assets are running 80% of the time.
- "pressure" (numeric): has values between 50 and 150, ex. psi
- "temperature" (numeric): has values between 20 and 120, ex. °C
- "speed" (numeric): has values between 100 and 1000, ex. m/s
- "vibration" (numeric): has values between 0 and 5, ex. mm/s
- "voltage" (numeric): has values between 210 and 240, ex. volts
- "flow" (numeric): has values between 10 and 100, ex. liters/min
- "torque" (numeric): has values between 100 and 500, ex. Nm
- "rpm" (numeric): has values between 500 and 1500
- "power" (numeric): has values between 50 and 500, exc. kW

The other store comes in the form of a ThingWorx Analytics Dataset which is created by the application using the "DEMO_BEANPRO_DATA.csv" and "DEMO_BEANPRO_METADATA.json" files that the user uploads to the project repository. This data is used by the "Analytics Insights" portion of the application. The fields in this data are:

- "AvgMaintenanceTimePrior1-30Days" (numeric): has values between 0 and 12.55
- "AvgMaintenanceTimePrior31-60Days" (numeric): has values between 0 and 9.0
- "AvgMaintenanceTimePrior61-90Days" (numeric): has values between 0 and 9.7
- "AvgTechnicianTenurePriorMonth" (numeric): has values between 0 and 41.56
- "UserErrorAlertCodesPrior1-30Days" (numeric): has values between 0 and 5
- "GrinderErrorOccurrence" (Boolean): ***Used as the goal variable in this demo***
- "PumpErrorOccurrence" (Boolean)
- "HeatingElementErrorOccurrence" (Boolean)
- "ServiceDeskCallsPrior1-30Days" (numeric): has values between 0 and 12
- "GrinderType" (categorical): has values "Integrated" and "Satellite"
- "PumpHeadType" (categorical): has values "Bolt Flange", "Splined Driving Rod", and "Twist Flange"
- "identifier" (string): unique identifier for the record
- "HopperJammedAlertCodesPrior1-30Days" (numeric): has values between 0 and 10
- "Latitude" (numeric): has values between -14.27 and 71.28
- "Longitude" (numeric): has values between -170.7 and 158.26
- "ServiceMaintenanceCallsPrior1-30Days" (numeric): has values between 0 and 11
- "MechanicalAlertCodesPrior1-30Days" (numeric): has values between 0 and 9
- "NoProblemAlertCodesPrior1-30Days" (numeric): has values between 0 and 6

- “OtherAlertCodesPrior1-30Days” (numeric): has values between 0 and 8
- “Region” (categorical): has values “CANADA”, “CENTRAL”, “EASTERN”, “LATIN AMERICAN”, “NORTH CENTRAL”, “NORTH EASTERN”, “PACIFIC ISLANDS”, “SOUTHERN”, and “WESTERN”
- “PriorityCodesLvls0-2Prior1-30Days” (numeric): has values between 0 and 16
- “PriorityCodesLvls3-5Prior1-30Days” (numeric): has values between 0 and 4
- “PriorityCodesLvls6-9Prior1-30Days” (numeric): has values between 0 and 2
- “AvgDailyHeatingTempPrior7Days” (numeric): has values between 86.7 and 95.9
- “AvgDailyPressurePrior7Days” (numeric): has values between 8 and 11.8
- “AvgDailyCyclesPrior7Days” (numeric): has values between 218 and 483
- “ AvgBoilerWaterLevelPrior7Days” (Boolean)
- “AvgDailyCleaningsTempPrior7Days” (numeric): has values between 0 and 5
- “HeatingTempVarianceFromLastMonth” (numeric): has values between -4.91 and 4.53
- “PressureVarianceFromLastMonth” (numeric): has values between -12.9 and 22.34
- “CyclesVarianceFromLastMonth” (numeric): has values between -27.12 and 37.5
- “BoilerWaterLevelVarianceFromLastMonth” (numeric): has values between -100 and 334.78
- “CleaningsVarianceFromLastMonth” (numeric): has values between -100 and 100
- “record_purpose” (categorical): has values “scoring” and “training”. *This version of the project does not use this field, and so it is marked as “informational” in ThingWorx Analytics so as to not be part of any analysis.*

What’s New in V2

V2 of the ThingWorx/LLM Accelerator introduces improvements that make the system more capable, transparent, and aligned with modern design principles. The most visible change is the new integration with ThingWorx Analytics (TWXA). With TWXA, users can move beyond simple queries and ask deeper questions about Signals, Profiles, datasets, and predictive model performance. This allows more meaningful exploration of what drives machine behavior and how reliable the trained models are. TWXA must be installed for these advanced insights, but environments without it can still use the original V1 “Data Insights.”

Users can now ask questions such as:

- “What can we learn from Signals in general?”
- “What datasets have been created for analytics purposes?”
- “Tell me more about the Bean Pro dataset.”
- “What are the top factors driving grinder failure in the Bean Pro machines?”
- “If I wanted to reduce likelihood of having grinder problems, what grinder type should I install in my machines?”
- “How good is the model that was trained on the Bean Pro dataset?”

The new version also improves extensibility and interpretability. Services are organized into modular components, so it is easier to add or adjust functionality. At the same time, V2 increases transparency by showing why certain services were chosen. This change introduces more complexity, since analytics questions now involve two LLM calls rather than one, but the result is richer and more reliable answers. Where V1 relied on five services, V2 now includes seventeen.

The execution process has also been rethought. In V1, a single prompt combined the user's question with data context, and the LLM returned everything in one step. V2 takes a different approach. It first sends a router prompt enriched with analytics context. The LLM responds with a list of services that should be run. ThingWorx executes those services, compiles the results into a second prompt, and then sends that to the LLM for the final answer. Separating service discovery from response generation leads to smarter choices and higher quality outputs.

At the core of this design is a router-based architecture. The router lets the LLM choose from up to twelve analytics services, running as many as needed depending on the question. This reduces wasted processing, cuts down costs, and ensures only the most relevant information is passed back. Each service is clearly defined, giving developers more control and users more confidence.

Finally, the mashup has been updated with a fresh design based on the PTC corporate template. The new interface improves usability and consistency, making insights easier to consume. Taken together, these changes make V2 a stronger, more flexible, and future-ready platform for bringing LLMs into ThingWorx environments.

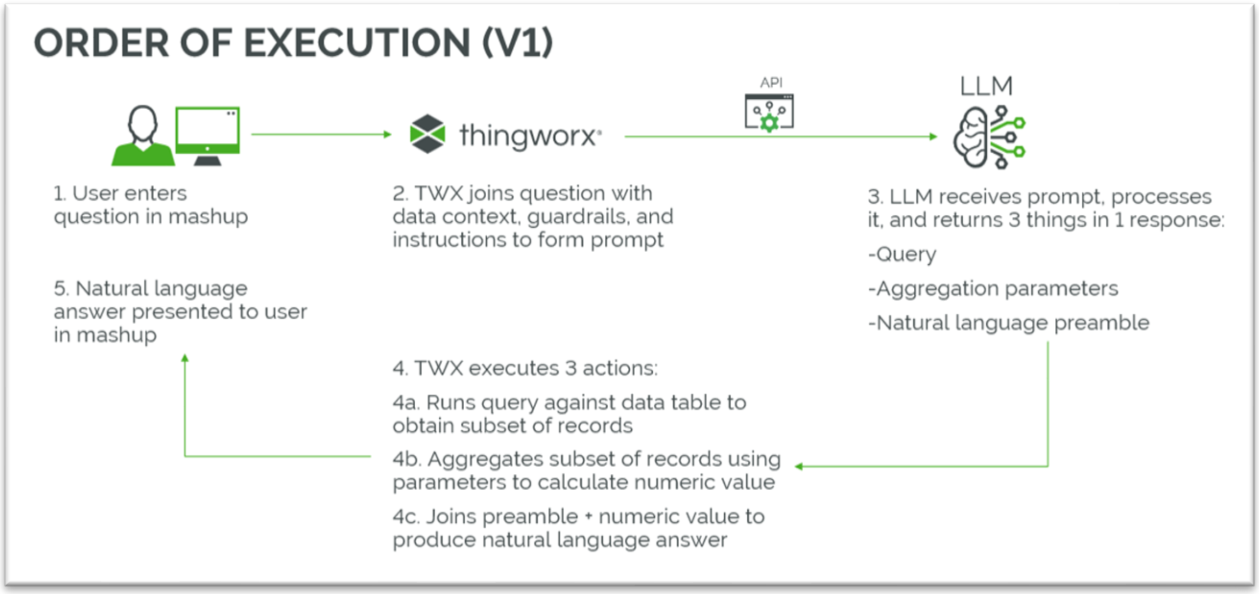

ORDER OF EXECUTION (V1)
thingworx
API
LLM
1. User enters question in mashup
2. TWX joins question with data context, guardrails, and instructions to form prompt
3. LLM receives prompt, processes it, and returns 3 things in 1 response:
-Query
-Aggregation parameters
-Natural language preamble
4. TWX executes 3 actions:
4a. Runs query against data table to obtain subset of records
4b. Aggregates subset of records using parameters to calculate numeric value
4c. Joins preamble + numeric value to produce natural language answer
5. Natural language answer presented to user in mashup

Figure 1: Order of Execution (V1)

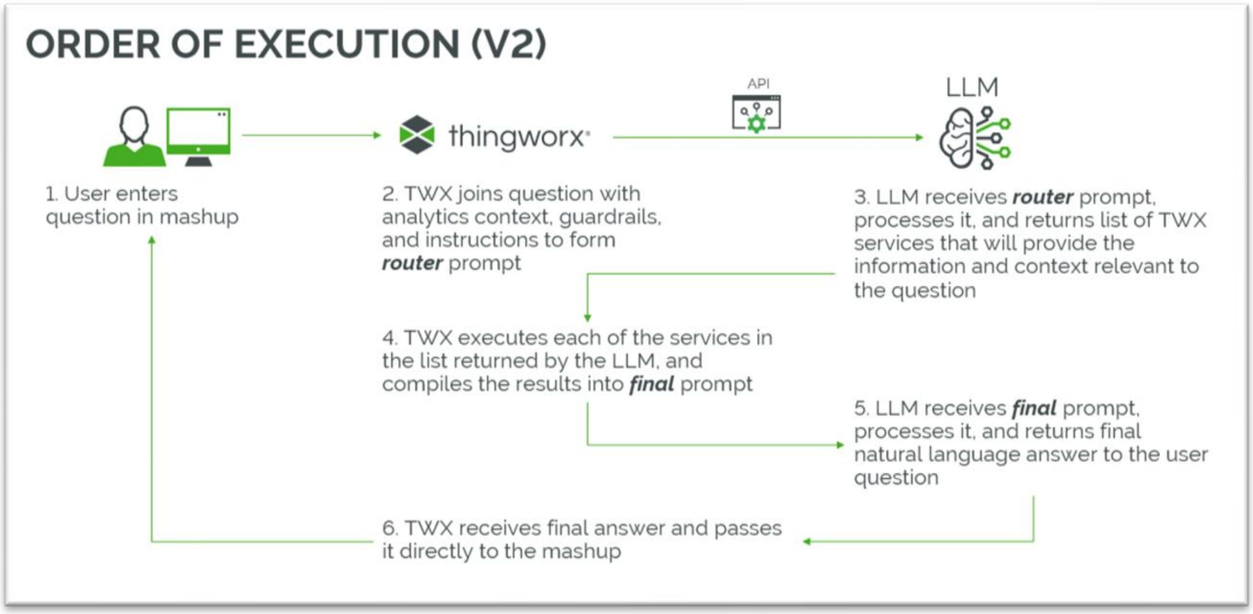

ORDER OF EXECUTION (V2)
thingworx
API
LLM
1. User enters question in mashup
2. TWX joins question with analytics context, guardrails, and instructions to form ***router*** prompt
3. LLM receives ***router*** prompt, processes it, and returns list of TWX services that will provide the information and context relevant to the question
4. TWX executes each of the services in the list returned by the LLM, and compiles the results into ***final*** prompt
5. LLM receives ***final*** prompt, processes it, and returns final natural language answer to the user question
6. TWX receives final answer and passes it directly to the mashup

Figure 2: Order of Execution (V2)

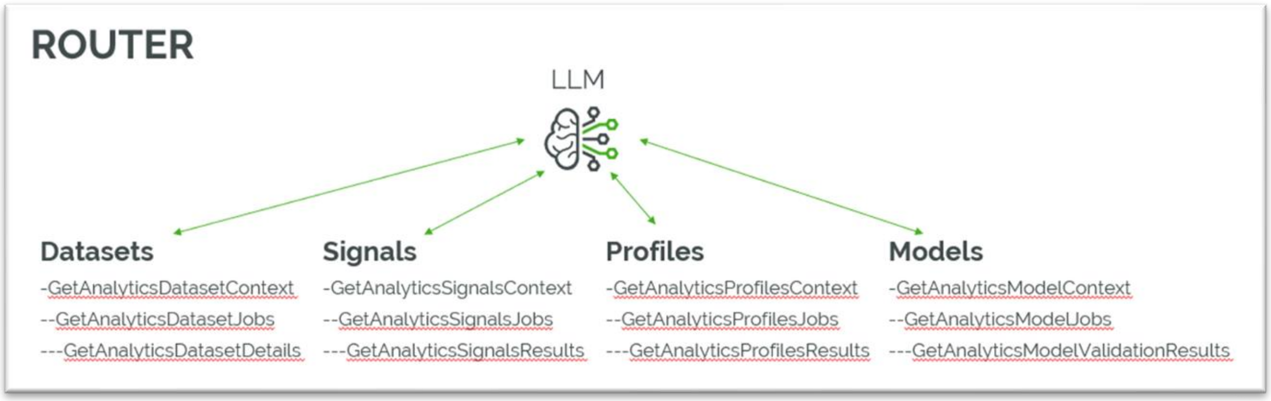

ROUTER
LLM
Datasets
-GetAnalyticsDatasetContext
--GetAnalyticsDatasetJobs
---GetAnalyticsDatasetDetails
Signals
-GetAnalyticsSignalsContext
--GetAnalyticsSignalsJobs
---GetAnalyticsSignalsResults
Profiles
-GetAnalyticsProfilesContext
--GetAnalyticsProfilesJobs
---GetAnalyticsProfilesResults
Models
-GetAnalyticsModelContext
--GetAnalyticsModelJobs
---GetAnalyticsModelValidationResults

Figure 3:Router Architecture

Demo Application Components

To accompany this document, a demo application in the form of a ThingWorx project named "PTCDTS.TWXLLM" has been created. This project may be imported into an environment running ThingWorx 9.5.0 or above, but it is not required to import the project. This document discusses the components of the project so you may recreate them to build a similar application.

The application discussed here leverages the abilities of ThingWorx to communicate with outside systems through APIs and other connection mechanisms. The application also leverages the natural language processing and coding abilities of LLMs. Together ThingWorx with LLMs can provide fast and reliable answers to the user easily and efficiently.

In this section you will see the ThingWorx entities that will need to be created in your environment. These entities collectively compose the application that integrates ThingWorx with an LLM.

Project: "PTCDTS.TWXLLM"

This project contains all the ThingWorx entities that compose the application. The entities and their associated properties or services may be seen in their entirety after importing the project into a sandbox environment running ThingWorx 9.5.0 or greater.

Note: The "data insights" portion of this demo uses a Data Table. The "analytics insights" portion uses a data CSV and metadata JSON file that the user uploads manually. In practice, the user may have their data in other databases or stores. The structure of this demo and the way it works can be replicated for those other cases.

The entities contained in the project are:

1. Thing: "PTCDTS.TWXLLM.Manager"
2. Data Table: "PTCDTS.TWXLLM.ExampleData_DT"
3. Data Shape: "PTCDTS.TWXLLM.ExampleData_DS"
4. Data Shape: "PTCDTS.TWXLLM.ExampleQuestion_DS"
5. Mashup: "PTCDTS.TWXLLM.DataQuery_MU"
6. Repository: "PTCDTS.TWXLLM.FileRepository"

1. Thing: " PTCDTS.TWXLLM.Manager"

This Thing is the main ThingWorx entity that powers the ThingWorx/LLM application. This Thing uses the "GenericThing" as the "Base Thing Template".

Properties

I. "GPTAuth" (String, persisted): Unique authorization token generated when setting up LLM account. This must be entered by the user before attempting to interact with the mashup.
II. "GPTUrl" (String, persisted): Unique URL to the LLM API that the application will use to talk to the LLM. This must be entered by the user before attempting to interact with the mashup.
III. "DemoDatasetJobId", "DemoModelJobId", "DemoProfilesJobId", "DemoSignalsJobId" (String, persisted): Unique job identifiers that are generated by the application when "Prepare Analytics" is executed.
IV. "TWXAnalyticsServerName" (String, persisted): Name of the ThingWorx Analytics server. This must be entered by the user before attempting to interact with the mashup. The name can be found by searching for the

"...._DataThing" that is created in the environment when ThingWorx Analytics is installed. The exact value to be entered as the "TWXAnalyticsServerName" is the string of characters leading up to the underscore "_".

Services

There are two categories of services related to this application. The "Data (V1)" category is for all services related to the "Data Insights" tab of the Mashup. These services are simpler and fewer and are perhaps easier for the user to begin using quickly. The "Analytics (V2)" category is for those related to the "Analytics Insights" tab. These services are more complicated and numerous.

Data (V1)

I. "CreateDataPrompt": This service takes as input the natural language "userQuestion" entered by the user in the "PTCDTS.TWXLLM.Insights_MU" Mashup. The service adds the necessary context and information for the LLM to generate a data query, aggregate parameters, and a natural language preamble for the final answer.

   Prompt engineering is the process of crafting specific inputs (prompts) to elicit desired outputs from generative AI models. This ensures effective human-AI communication to maximize the potential of these models. There's a lot of skill and nuance that goes into prompt engineering, and that goes beyond the scope of this document. However, there are a few key points illustrated below. For more information check out OpenAI's best practices for prompt engineering.

   The prompt sent to the LLM for processing has a few components:

   - *Request for query.* The JSON query generated by the LLM will be executed by ThingWorx against the Data Table to retrieve the raw data that is relevant to the question posed by the user. To accompany the request is an example query to help ensure that the new query generated by the LLM is of the proper format and contains the correct information.
   - *Natural language question asked by the user*. This is passed word-for-word to the LLM and will most likely be imperfect. For example, the user may accidentally enter typos or ask about assets (machines) - or their properties - that don't exist. This highlights the need for the other prompt components.
   - *Data table information*. This contains the names of the columns in the Data Table, as well as their corresponding data types. This helps the LLM understand the context of the data to properly form a valid query for ThingWorx to execute. It's also beneficial to include other context of the data such as a list of the assets, what range of time the data covers, as well as the frequency of the data.
   - *Request for aggregate parameters*. The JSON object generated by the LLM is used by ThingWorx as part of a service call to aggregate the raw queried into the numeric part of the final answer requested by the user. An example aggregate parameter JSON object is sent along to the LLM to help ensure that the new object generated by the LLM is of the proper format and contains the correct information. The possible types of aggregations that can be used are also sent.
   - *Request for answer preamble.* A preamble is an incomplete natural language answer to the question asked by the user. It contains only the words which will later be joined with the calculated numerical answer from the ThingWorx aggregation to form the complete answer displayed to the user in the Mashup.

- *Guardrails*. To discourage the user from asking questions that aren't relevant to the data and to help avoid erroneous answers, part of the prompt is dedicated to guardrails. It's helpful to give the LLM at least a small list of criteria by which the LLM can decide if a user question is valid or not. If the LLM deems the question to be invalid, it is instructed to return a retry question to the user.
- *Request for Delimiters*. Whenever a response from an LLM contains multiple pieces that need to be processed separately, the prompt should instruct to put those pieces within delimiters. These can be characters such as "<" and ">". The "ProcessDataQuestion" service extracts the different pieces from the pairs of delimiters and handles them appropriately.

II. "GetExampleDataQuestion": This service takes no input but returns an infotable of example questions that the user may ask of the data. This infotable is passed to a dropdown menu in the "Data Insights" tab of the Mashup. The user has the option to choose one of the example questions from the menu and submit it as-is or modify if they like. One example question is "What was the highest speed registered by Machine1 in January 2025?".

III. "GetLLMTextResponse": This service takes as input the "prompt" that was created by either the "CreateDataPrompt" service or "CreateAnalyticsPrompt", as well as a numeric "top_p" value. These are passed to the LLM, and the text response is requested. The string returned by the LLM is then output by the service.

ChatGPT has parameters "top_p" and "temperature" that users can modify to suit their use case. For "top_p", values closer to 0 will encourage ChatGPT to provide more consistent answer and are better suited for coding use cases. Values closer to 1 are better suited for writing and other creative use cases. This application requires accurate and consistent code from the LLM, so a value of 0.1 is used. See this OpenAI Developer Community post for more information.

IV. "ProcessData": This service takes the inputs "aggregateParams" and "query", which are both JSON objects written by the LLM. It passes the query to the "QueryDataTableEntries" service on the Data Table. The data returned by the query is then aggregated using the aggregate parameters. The single numeric result is finally rounded to two decimal places and output as a string.

V. "ProcessDataQuestion": This is the main service that orchestrates the flow of data and information through all the other "Data (V1)" services. It takes as input the string "userQuestion" that was entered by the user in the "Data Insights" tab of the Mashup. This is then passed to the "CreateDataPrompt" service, which returns the full string prompt. The prompt is then passed to the "GetLLMTextResponse" service. The "query", "aggregateParams", and answer preamble are extracted from the response using the delimiters. The "aggregateParams" and "query" are sent to the "ProcessData" service. The resulting answer is joined with the answer preamble and output as a string. This is the final natural language answer the user sees in the "Data Insights" tab of the Mashup.

Analytics (V2)

I. "CreateAnalyticsListPrompt": This service contains information and context about the types of results that the other "Analytics (V2)" services provide. The result of this service is a prompt that is sent to the LLM along with the user question. The LLM is asked to analyze the question and decide which services will provide the right information to answer the question. Some guardrails are put in place so that if the user question can't be addressed by the relevant information produced by these services, then an "invalid question" flag is returned from the LLM.

   This is the first service executed by the "ProcessAnalyticsQuestion" service and dictates how the rest of the ThingWorx/LLM processing occurs to answer the user question.

   This service essentially helps the LLM figure out what it needs from ThingWorx to answer the question. By dividing the types of content across multiple services, we greatly reduce the amount of material that is ultimately sent to the LLM for consideration in answering the user question. This architecture achieves several things:

   - Reduced AI cost by limiting the total number of tokens sent to the LLM
   - Reduced AI processing time by limiting the amount of information the LLM must sift through and consider for inclusion in the final response to the user
   - Increased chance of an accurate and thoughtful answer from the LLM by only including relevant information
   - Increased information control within the ThingWorx application by separating information into different services which can easily be restricted or allowed as needed

II. "GetAnalyticsDatasetContext", "GetAnalyticsModelContext", "GetAnalyticsProfilesContext", "GetAnalyticsSignalsContext": These four services all function in the same way. They don't take any inputs or do any processing. All that they do is return a string of context about their respective Analytics jobs. For example, "GetAnalyticsDatasetContext" tells what is contained in an analytics dataset, and what fields or options are part of a dataset. The context that these services provide help to answer more conceptual questions like "So what?" or "How is this significant?". This context may be packaged later with other information and sent to the LLM for processing into the final answer to the user question.

III. "GetAnalyticsDatasetJobs", "GetAnalyticsModelJobs", "GetAnalyticsProfilesJobs", "GetAnalyticsSignalsJobs": These four services all function the same way. They don't take any inputs. They simply interact with their respective Analytics Server Microservice to retrieve the lists of all jobs that have been executed. For example, "GetAnalyticsSignalsJobs" retrieves a list of all the Signals jobs that have been run from the "..._SignalsThing" Microservice. The services then package the list of jobs along with descriptions of the fields that are inside the jobs. This information helps to answer questions related to all jobs that have been run, as well as produce job Ids that can be used by "...Results" or "...Details" services. This information may be packaged later with other information and sent to the LLM for processing into the final answer to the user question.

IV. “GetAnalyticsDatasetDetails”, “GetAnalyticsModelValidationResults”, “GetAnalyticsProfilesResults”, “GetAnalyticsSignalsResults”: These four services all function the same way. They all take as input a job id and then use it to retrieve the results for that job from their respective Microservice. For example, “GetAnalyticsProfilesResults” take a Profiles job Id, and retrieves the results from the “...ProfilingThing” Microservice. These services may package the results with extra context for the results. This information may be packaged later with other information and sent to the LLM to process it into the final answer to the user question.

V. “GetAnalyticsStatuses”: This service determines if the four analytics jobs (dataset, Signals, Profiles, model) completed successfully and are ready for use in the application. It does this by checking the four relevant Microservices to see if these jobs are in the “COMPLETED” state. It outputs a string message that is displayed to the user in the “Analytics Insights” tab that tells the state of the four jobs.

VI. “GetExampleAnalyticsQuestion”: This service takes no input but returns an infotable of example questions that the user may ask of the analytics jobs. This infotable is passed to a dropdown menu in the “Analytics Insights” tab of the Mashup. The user has the option to choose one of the example questions from the menu and submit it as-is or modify if they like. One example question is “How does grinder type affect the goal variable in the Signals job?”.

VII. “ProcessAnalyticsQuestion”: This is the main service that orchestrates the flow of data and information through all the other “Analytics (V2)” services. It takes as input the string “userQuestion” that was entered by the user in the “Analytics Insights” tab of the Mashup. This is then passed to the “CreateAnalyticsListPrompt” service, which creates the “listPrompt”. This prompt is then sent to the LLM, and a list of “Analytics (V2)” services capable of producing information relevant to the user question is returned. Each “Analytics (V2)” service that was in the list is then executed, and the resulting information is joined together with some guardrails into a final prompt. This final prompt is then sent to the LLM for processing, and the response is presented in the answer box of the “Analytics Insights” tab.

VIII. “RunDemoAnalytics”: This service takes care of running all the analytics jobs that will be used by the demo. It first creates a ThingWorx Analytics dataset using the “DEMO_BEANPRO_DATA.csv” and “DEMO_BEANPRO_METADATA.json” files that the user uploaded to the “PTCDTS.TWXLLM.FileRepository” repository. This dataset is given the name “Demo_Beanpro_Dataset”. The service waits until the dataset is completed before moving on to running a Signals job on that dataset. This Signals job is assigned the name “Demo_Beanpro_Signals”. Once Signals are complete, Profiles are run on the same dataset. Profiles are given the name “Demo_Beanpro_Profiles”. Once Profiles are done, a predictive model is trained on the dataset. The model is given the name “Demo_Beanpro_Model” and is trained using the “NEURAL_NET” machine learning algorithm, with the validation holdout percentage set to 20%.

2. Data Table: “PTCDTS.TWXLLM.ExampleData_DT”

This Data Table is the ThingWorx entity that holds data for the “Data Insights” portion of this demo.

Services

I. “GenerateRandomDemoData”: This service generates random data used for this demo. It first creates an empty infotable using the “PTCDTS.TWXLLM.ExampleData_DS” Data Shape. Then it establishes a start datetime value of November 1st, 2024 and the demo assets to be “Machine1”, “Machine2”, and “Machine3”. Next it creates an hourly entry for each of the machines until it reaches an end datetime value of March 1st, 2025. In total there are 8640 entries.

   This service only needs to be run once to populate the Data Table. The “Prepare Data” button from the Mashup calls this service. If the user tries to submit a question before this button is pushed and the data is prepared, they will be prompted to prepare the data.

II. “GetDataStatus”: This service determines if the data in the Data Table is ready for use in the application. It does this by checking if the correct number of entries exist. It outputs a string message that is displayed to the user that shows the progress of the data preparation.

3. Data Shape: “PTCDTS.TWXLLM.ExampleData_DS”

This Data Shape is used by the Data Table to define the field names and (data types) in the table. See the Data Preparation section above for the complete list.

4. Data Shape: “PTCDTS.TWXLLM.ExampleQuestion_DS”

This Data Shape is used by the Manager Thing to accompany the infotable that is produced by the “GetExampleQuestion” service. It only has one field:

- “question” (string): example question to be displayed to user in the Mashup

5. Mashup: “PTCDTS.TWXLLM.Insights_MU”

This Mashup is the user interface for the application.

From the Data Table (“Data Insights” tab):

- Calls “GenerateRandomDemoData” when “Prepare Data” is clicked.
- Invokes “GetDataStatus” when the mashup is loaded, and by means of an Auto Refresh function that runs every second.

From the Manager Thing (“Data Insights” tab):

- Runs “GetExampleDataQuestion” when the Mashup loads and displays the result in the dropdown.
- Calls “ProcessDataQuestion” service when “Submit” is clicked.

From the Manager Thing (“Analytics Insights” tab):

- Executes “GetExampleAnalyticsQuestion” when the Mashup loads and displays results in the dropdown.
- Invokes “GetAnalyticsStatuses” when the mashup is loaded, and by means of an Auto Refresh function that runs every second.
- Calls “ProcessAnalyticsQuestion” service when “Submit” is clicked.
- Executes “RunDemoAnalytics” when “Prepare Analytics” is clicked.

6. Repository: “PTCDTS.TWXLLM.FileRepository”

This repository will hold the demo Bean Pro Espresso data CSV and its associated metadata JSON file.

Importing & Using the Demo Application

Follow these steps to get up and running with the demo.

1.) Use the “Import” button along the bottom left of ThingWorx Composer

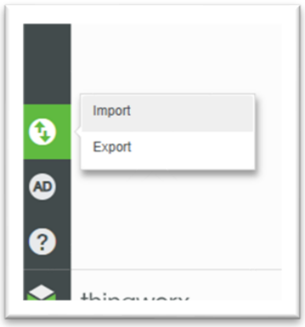
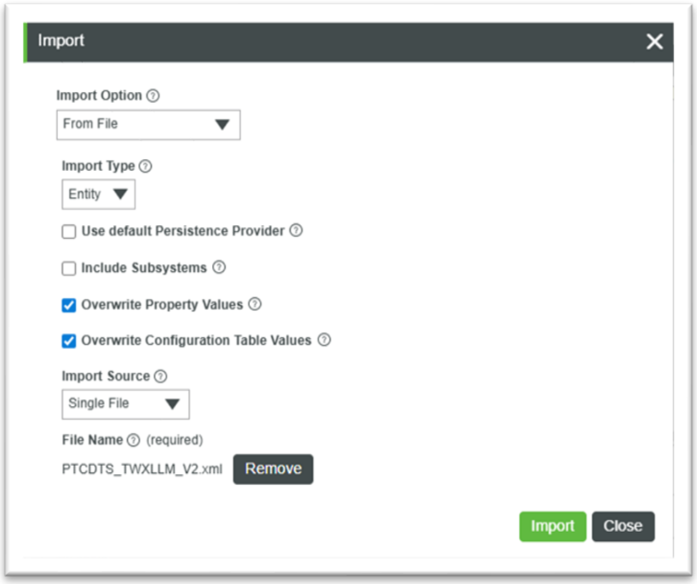


Figure 4: Import Button

2.) Choose the “From File” import option and “Entity” import type. Then browse to the location where the “PTCDTS_TWXLLM_V2.xml” file is stored. Select the file and then click “Import”


Import

Import Option: From File

Import Type: Entity

☐ Use default Persistence Provider

☐ Include Subsystems

☑ Overwrite Property Values

☑ Overwrite Configuration Table Values

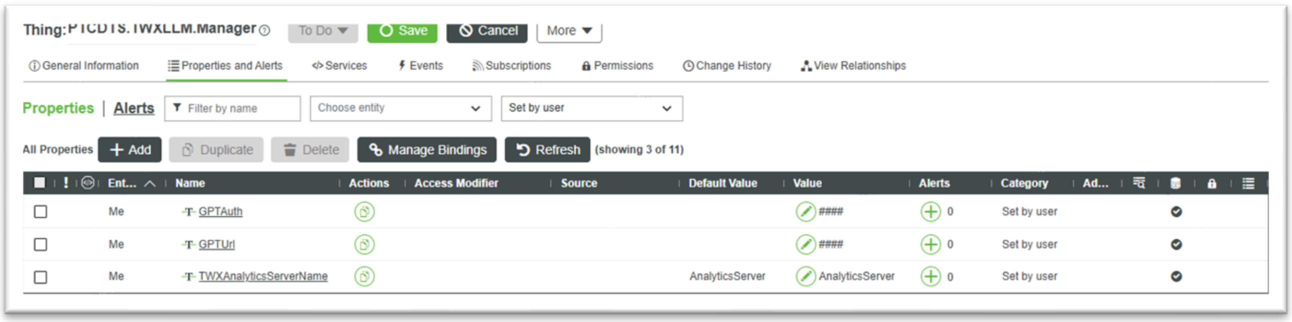
Import Source: Single File

File Name (required): PTCDTS_TWXLLM_V2.xml

Figure 5: Entity Selection

Best Practice Name	Version	Version Date	Author
Using ThingWorx with LLMs	*V2.0*	*6/30/2025*	*David Kessler* *Document authored by PTC, with AI assistance*
Software Versions Tested	**Page**	**Please Send Feedback to**	**Reviewer**
ThingWorx 9.5.0 & above	18	*dkessler@ptc.com*	

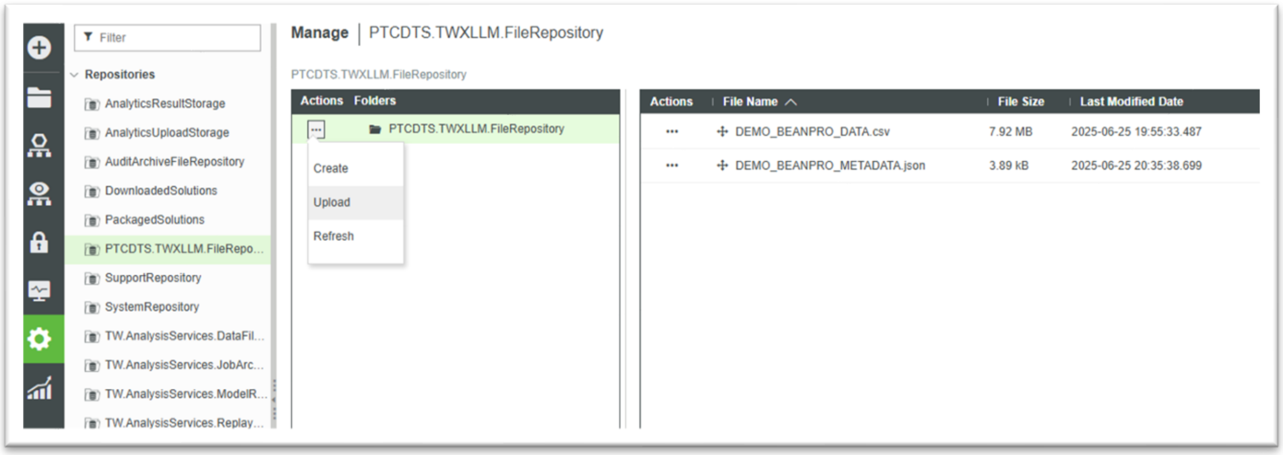
3.) Open the Manager Thing "PTCDTS.TWXLLM.Manager" and click the "Properties and Alerts" tab. Enter the values for "GPTAuth" and "GPTUrl" obtained when setting up the LLM account. Enter the name of the ThingWorx Analytics server for "TWXAnalyticsServerName". The name can be found by searching for the "…._DataThing" that is created in the environment when ThingWorx Analytics is installed. The exact value to be entered as the "TWXAnalyticsServerName" is the string of characters leading up to the underscore "_".



Ent...	Name	Actions	Access Modifier	Source	Default Value	Value	Alerts	Category	Ad...
Me	GPTAuth					####	0	Set by user	
Me	GPTUrl					####	0	Set by user	
Me	TWXAnalyticsServerName				AnalyticsServer	AnalyticsServer	0	Set by user	

Figure 6: Setting Credentials

4.) Upload the "DEMO_BEANPRO_DATA.csv" and "DEMO_BEANPRO_METADATA.json" files to the "PTCDTS.TWXLLM.FileRepository" repository.


Filter
Repositories
AnalyticsResultStorage
AnalyticsUploadStorage
AuditArchiveFileRepository
DownloadedSolutions
PackagedSolutions
PTCDTS.TWXLLM.FileRepo...
SupportRepository
SystemRepository
TW.AnalysisServices.DataFil...
TW.AnalysisServices.JobArc...
TW.AnalysisServices.ModelR...
TW.AnalysisServices.Replay...
Manage | PTCDTS.TWXLLM.FileRepository
PTCDTS.TWXLLM.FileRepository
Actions Folders
PTCDTS.TWXLLM.FileRepository
Create
Upload
Refresh

Actions	File Name	File Size	Last Modified Date
•••	DEMO_BEANPRO_DATA.csv	7.92 MB	2025-06-25 19:55:33.487
•••	DEMO_BEANPRO_METADATA.json	3.89 kB	2025-06-25 20:35:38.699

Figure 7: Upload Analytics Files

Best Practice Name	Version	Version Date	Author
Using ThingWorx with LLMs	*V2.0*	*6/30/2025*	*David Kessler* *Document authored by PTC, with AI assistance*
Software Versions Tested *ThingWorx 9.5.0 & above*	**Page** 19	**Please Send Feedback to** *dkessler@ptc.com*	**Reviewer**

5.) Open the Mashup “PTCDTS.TWXLLM.Insights_MU” and click “View Mashup”.

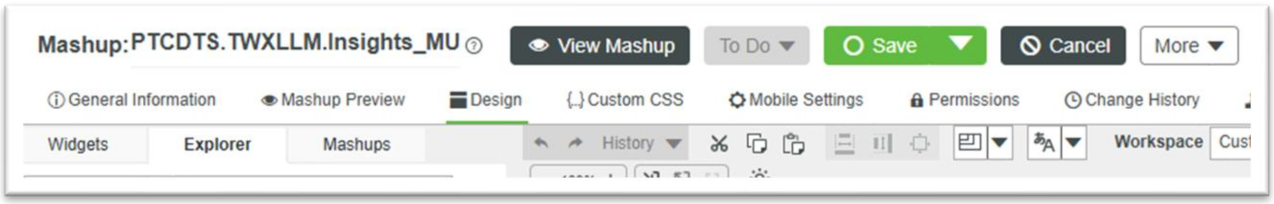


Figure 8: View Mashup

6.) Click “Prepare Data” in the “Data Insights” tab of the Mashup.

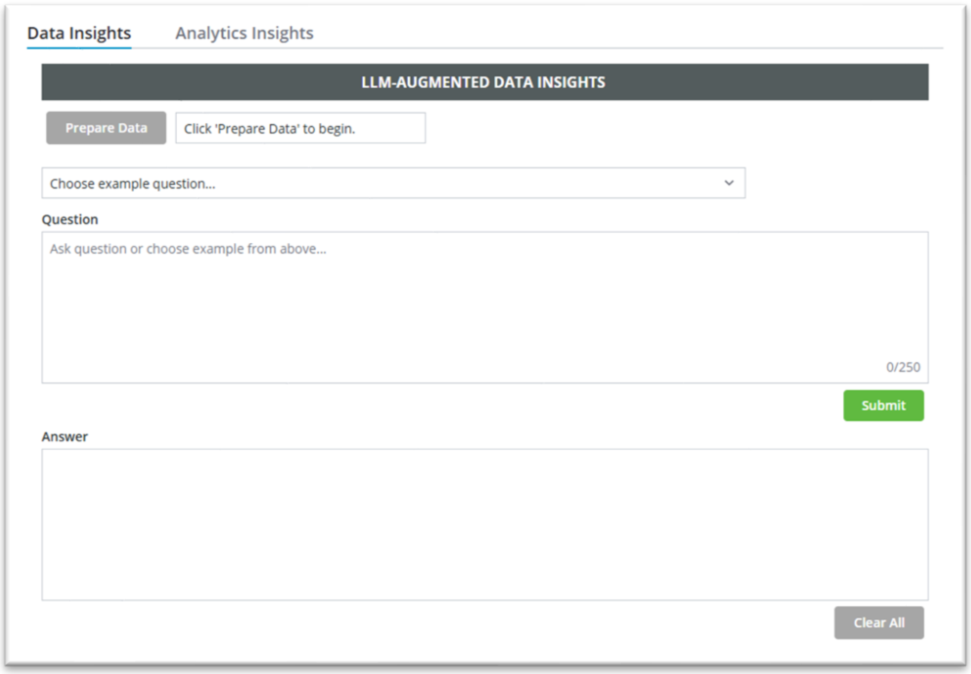
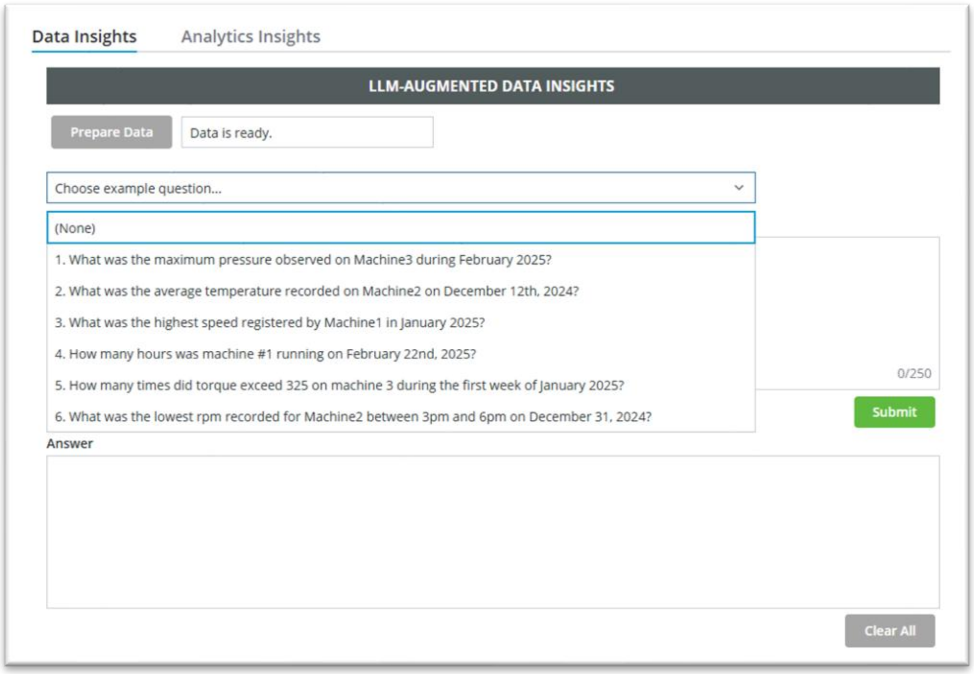

Data Insights
Analytics Insights
LLM-AUGMENTED DATA INSIGHTS
Prepare Data
Click 'Prepare Data' to begin.
Choose example question...
Question
Ask question or choose example from above...
0/250
Submit
Answer
Clear All

Figure 9: Prepare Data

7.) Choose one of the example questions from the dropdown or enter a custom similar question.



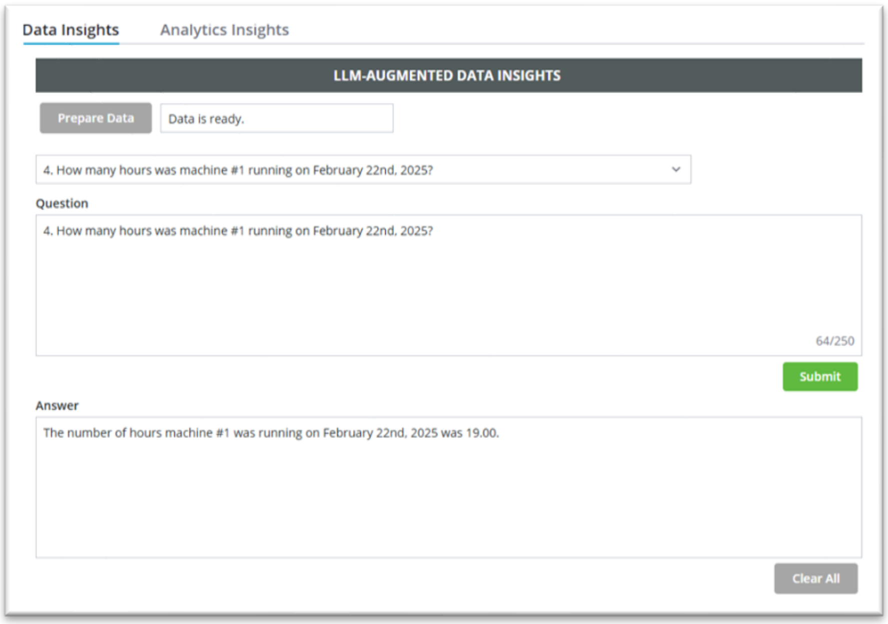
Data is ready.

Choose example question...

(None)
1. What was the maximum pressure observed on Machine3 during February 2025?
2. What was the average temperature recorded on Machine2 on December 12th, 2024?
3. What was the highest speed registered by Machine1 in January 2025?
4. How many hours was machine #1 running on February 22nd, 2025?
5. How many times did torque exceed 325 on machine 3 during the first week of January 2025?
6. What was the lowest rpm recorded for Machine2 between 3pm and 6pm on December 31, 2024?

Figure 10: Example Data Insights Question

8.) Submit the question and wait a few seconds for a response.



Data is ready.

4. How many hours was machine #1 running on February 22nd, 2025?

Question

4. How many hours was machine #1 running on February 22nd, 2025?

Answer

The number of hours machine #1 was running on February 22nd, 2025 was 19.00.

Figure 11: Submit Question

9.) Click “Prepare Analytics” in the “Analytics Insights” tab of the Mashup.

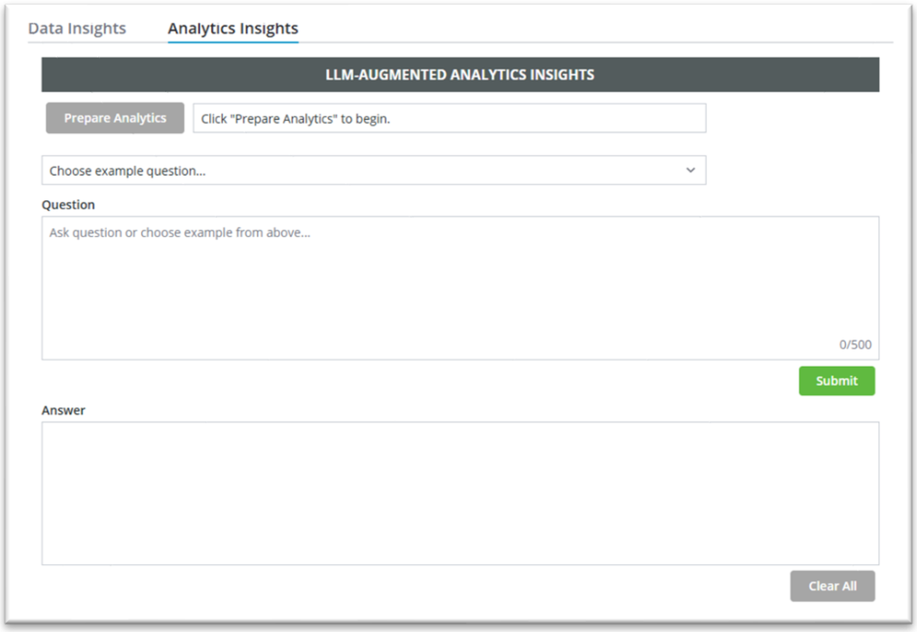
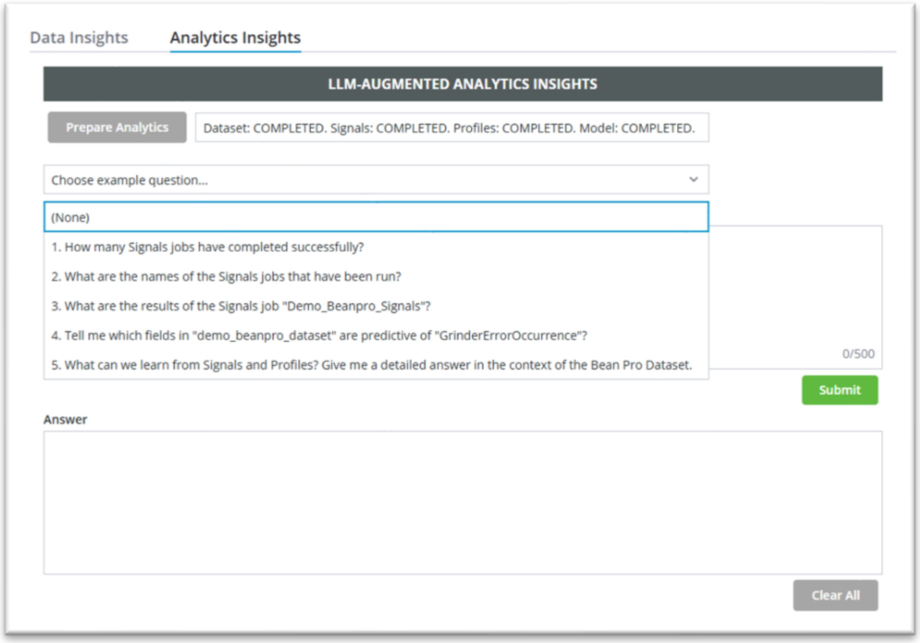

Data Insights
Analytics Insights
LLM-AUGMENTED ANALYTICS INSIGHTS
Prepare Analytics
Click "Prepare Analytics" to begin.
Choose example question...
Question
Ask question or choose example from above...
0/500
Submit
Answer
Clear All

Figure 12: Prepare Analytics

10.) Choose one of the example questions from the dropdown or enter a custom similar question.

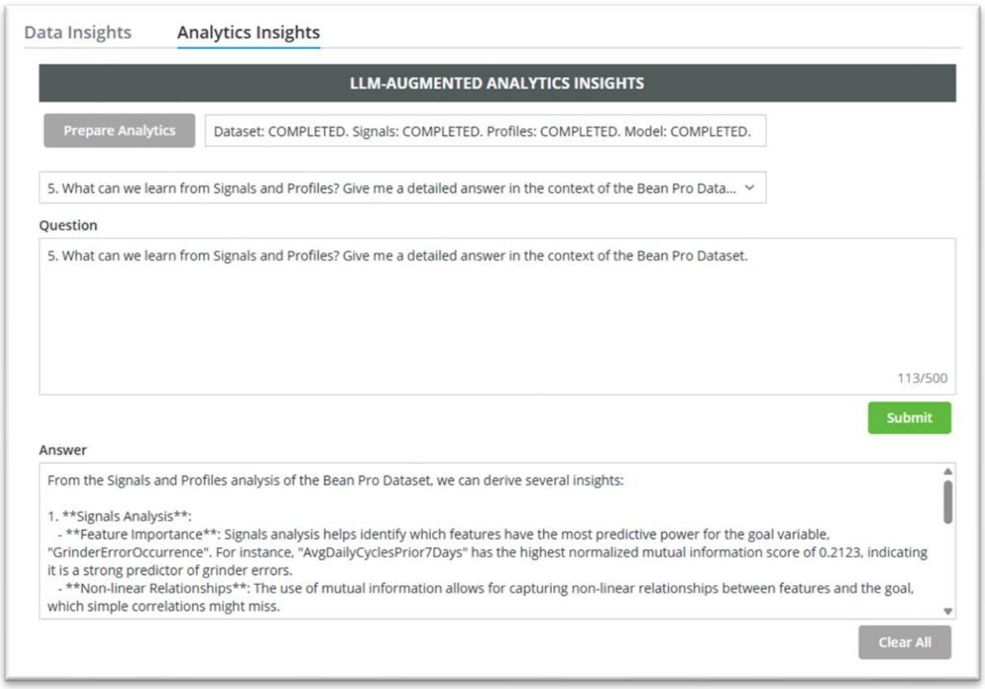


Dataset: COMPLETED. Signals: COMPLETED. Profiles: COMPLETED. Model: COMPLETED.

(None)
1. How many Signals jobs have completed successfully?
2. What are the names of the Signals jobs that have been run?
3. What are the results of the Signals job "Demo_Beanpro_Signals"?
4. Tell me which fields in "demo_beanpro_dataset" are predictive of "GrinderErrorOccurrence"?
5. What can we learn from Signals and Profiles? Give me a detailed answer in the context of the Bean Pro Dataset.

Figure 13: Example Analytics Insights Question

11.) Submit the question and wait a few seconds for a response.



Dataset: COMPLETED. Signals: COMPLETED. Profiles: COMPLETED. Model: COMPLETED.

5. What can we learn from Signals and Profiles? Give me a detailed answer in the context of the Bean Pro Data...

Question

5. What can we learn from Signals and Profiles? Give me a detailed answer in the context of the Bean Pro Dataset.

113/500

Answer

From the Signals and Profiles analysis of the Bean Pro Dataset, we can derive several insights:

1. **Signals Analysis**:
 - **Feature Importance**: Signals analysis helps identify which features have the most predictive power for the goal variable, "GrinderErrorOccurrence". For instance, "AvgDailyCyclesPrior7Days" has the highest normalized mutual information score of 0.2123, indicating it is a strong predictor of grinder errors.
 - **Non-linear Relationships**: The use of mutual information allows for capturing non-linear relationships between features and the goal, which simple correlations might miss.

Figure 14: Submit Question

Link the Large Language Model (LLM)

Introduction

There are many LLMs - both closed and open source - that can be used as part of an IIoT application. However, not all models are created equally in terms of security, speed, cost, and quality. Some LLMs can be hosted on-prem, while others are only available through cloud-based API usage. The purpose of this paper is to focus on integrating an LLM into an IIoT application, and so a recommendation is here made for an LLM that will satisfy many use cases.

Set up LLM account access

Integrating an LLM into an IoT platform introduces important security and data governance considerations. Best practices for setting up and using an LLM service with ThingWorx include:

• Use Enterprise-Grade LLM Services: Prefer an LLM deployment that offers strong privacy guarantees. For instance, using Microsoft Azure OpenAI Service (Azure ChatGPT) ensures that your prompts and data are not used to retrain the public model. Azure OpenAI provides the same advanced GPT models with the security and enterprise compliance of Azure. This means data is processed within Microsoft's secure cloud, with encryption at rest and in transit, and no data is retained beyond a short window for abuse monitoring. Such services also allow features like private networking (VNET integration) to keep API calls off the public internet. Using Azure or similar enterprise LLM services can protect proprietary IoT data (like machine performance stats or product yield figures) from leaking into public AI models.

• API Key Management: Treat LLM API keys or credentials as highly sensitive. It is a good idea to store them in ThingWorx's Security Management Tool or in an encrypted configuration file on the server, rather than hard coding in ThingWorx services. Only the server administrators and the LLM connector service should have access to these keys. Rotate keys periodically and monitor usage. If using Azure OpenAI, leverage Azure's identity and access management (e.g., Managed Identity or Key Vault) to supply credentials to ThingWorx connectors securely.

• Data Minimization & Anonymization: Only send the necessary data to the LLM. Avoid including raw personally identifiable information (PII) or sensitive business data in prompts if not needed. For example, instead of sending a full maintenance log with employee names or exact timestamps, pre-process it to only include the fault descriptions and durations. If sending any potentially sensitive data (like production rates or defect counts), consider anonymizing identifiers. Be prudent about what IoT data is exposed to the LLM – even if the LLM service is secure, any data in a prompt could theoretically be seen by the AI provider's systems. Thus, follow company data governance policies: classify data and restrict sending of any confidential or export-controlled information to third-party APIs. In Defense/Aerospace contexts, this likely means using a fully on-premises LLM or one in a secure cloud enclave for any mission-critical or classified data.

• Network Security: If ThingWorx is on-premises, ensure outbound connections to the LLM API go through approved network routes (VPN or secure gateway). Use HTTPS/TLS for all API calls. Azure OpenAI allows binding the service to a private endpoint; if possible, use that so the traffic doesn't traverse the public internet. Configure firewalls to only allow ThingWorx to communicate with the specific LLM API hosts. This reduces exposure in case an attacker compromised the ThingWorx server – they couldn't misuse the network to call arbitrary external services.

• LLM Account and Permissions: Limit the LLM account to only the needed capabilities and quotas. For example, if using Azure, apply resource locks or quota limits to prevent runaway costs or abuse. Do not reuse the same LLM API account for non-IoT purposes; segregate it to this integration so monitoring is easier. Keep an audit log of all prompts sent to the LLM and responses received (excluding sensitive data) – this will help in reviewing what data was shared and detecting any unusual usage.

• Compliance and Legal: Check compliance requirements in your industry. Life Sciences and Healthcare might have HIPAA or other regulations; ensure that using an LLM (which might temporarily store prompts) does not violate any rules. Azure OpenAI, for instance, guarantees data retention of prompts for only 30 days for abuse detection and then deletion – such details might help in a compliance review. Document how the LLM is used (e.g., it's not making autonomous decisions, only recommendations) to alleviate concerns from regulators or customers.

By setting up the LLM integration with these security practices, companies can reap the AI benefits while safeguarding their IoT data and intellectual property. The document will highlight a case example – e.g., a manufacturer using Azure OpenAI through ThingWorx to answer production questions, with all network traffic staying within the Azure cloud and no data being retained by the LLM service beyond the immediate query. This provides peace of mind that trade secrets (like detailed process parameters) won’t leak, addressing one of the main pitfalls of using services like ChatGPT in enterprise settings.

1. Benefits of OpenAI for Microsoft Azure

Microsoft Azure OpenAI offers enterprise-grade security, compliance, and seamless integration for manufacturing operations. It ensures data security with encryption and prevents proprietary information from being used for model training. Compliance with global standards like GDPR and ISO 27001 helps regulated industries meet strict mandates while minimizing risks. Azure also provides data residency options, allowing manufacturers to control where their data is stored to meet regional regulations. Its deep integration with the Microsoft ecosystem facilitates AI-driven solutions for operational efficiency. Additionally, 24/7 enterprise support and SLAs ensure system reliability, while customization and scalability allow manufacturers to fine-tune AI models securely to optimize processes and enhance predictive maintenance.

2. Why Protecting Data is Crucial When Using LLMs

- **Preservation of Competitive Advantage:**
  Manufacturing data often includes unique production processes, maintenance strategies, and machine configurations. If this sensitive data were exposed or misused, competitors could gain insights that erode your competitive edge.
- **Regulatory Compliance & Risk Management:**
  Global factories must comply with stringent data protection regulations (such as GDPR or local data privacy laws). A data breach or mishandling can lead to severe financial penalties and reputational damage.
- **Operational Integrity:**
  Unprotected data can lead to unauthorized access or manipulation, potentially disrupting critical manufacturing operations. Secure data practices help ensure that your production systems run smoothly, contributing to improved overall equipment effectiveness (OEE) and reduced unplanned downtime.
- **Trust and Safety:**
  Protecting proprietary information fosters trust among stakeholders, customers, and partners. It demonstrates your commitment to maintaining the confidentiality, integrity, and availability of critical business data.

Evaluate the Results

Validating and Governing LLM-Generated Insights in an IoT Application

While LLMs can generate human-like answers and helpful insights, it's crucial to validate the outputs within the IoT context to ensure accuracy and reliability. IoT applications often involve critical operations where decisions must be correct and timely. This section outlines methods to verify and govern LLM responses:

• Cross-Check with Source Data: Whenever an LLM provides a numeric answer or specific statistic from IoT data, have ThingWorx verify that against the actual database/value stream. For example, if the LLM says "Compressor 3's max pressure was 250 psi on Jan 5", the system can programmatically check that value via a ThingWorx query to ensure it's not hallucinated. This can be done by designing the LLM prompts such that the LLM returns a structured answer (e.g., JSON with fields), making it easier for ThingWorx to parse and validate the content before showing it to the user. Any discrepancies or low-confidence answers should be flagged. Essentially, never blindly trust an LLM's output when exact data is available to confirm. Use the LLM for language and analysis but use ThingWorx for factual accuracy.

• Feedback Loop and Human Oversight: Incorporate a feedback mechanism where users can mark LLM answers as correct or incorrect, which is logged. In a manufacturing setting, if the LLM recommends adjusting a machine calibration and an engineer notices it's based on a misinterpretation, they should have an easy way to provide that feedback. This can trigger a review of the prompt design or data provided to the LLM. Keep a human-in-the-loop especially for high-impact recommendations – for instance, an LLM might draft a maintenance plan, but a reliability engineer should approve it before execution. Over time, this oversight helps the LLM integration "learn" (even if the LLM model itself isn't retraining, the development team can refine prompts or add rules based on repeated errors).

• Prompt Engineering to Reduce Errors: Craft prompts that guide the LLM to be accurate and to show its reasoning or uncertainty. For example, you might prompt: *"If the data is insufficient to answer, respond with 'Unknown'."* or *"List the top 3 factors with reasoning from the data."* This can prevent the LLM from guessing or making up data when the IoT data is unclear. By having the LLM explain the basis of its answer (which sensor readings or thresholds led to a conclusion), you make it easier to spot if it's drawing correct inferences. Some prototypes even use *chain-of-thought* prompting or knowledge graph validation to ensure the LLM's statements align with known facts. In this document, we can't dive deep into AI theory, but we will recommend that the implementers iterate on prompt strategies to achieve reliable outputs.

• Limit the Decision Autonomy: Use LLM outputs as recommendations or insights, not final commands, in critical systems. For instance, an LLM can suggest that a machine be shut down for inspection, but it should not directly trigger the shutdown through ThingWorx without human confirmation (unless thoroughly tested and approved). This layered approach aligns with safety standards in industries like aerospace and life sciences, where AI suggestions must undergo verification. By designing the IoT application to treat LLM responses as one input (alongside sensor alarms, etc.), you avoid over-reliance on an AI that might occasionally err.

• Monitoring and Improvement: Continuously monitor the performance of the LLM integration. Track metrics like the percentage of questions answered correctly, the time saved in analysis, reduction in downtime or quality issues after implementing LLM suggestions, etc. Also monitor the LLM's usage patterns for any anomalies (if it starts giving odd answers or if response times increase, for example). Periodic reviews of the LLM's output by domain experts (perhaps monthly audits of a sample of interactions) can ensure it remains aligned with reality. As better models or new features (like domain-specific LLMs or tools) become available, plan for updates to the LLM_Manager and prompts.

Conclusion

Integrating large language models with ThingWorx can unlock powerful new capabilities – from intuitive data querying to intelligent maintenance advisors – across industrial sectors. With connectivity to virtually any data source (OPC UA servers, SQL databases, historians via REST/API integration), ThingWorx provides the foundation of reliable IoT data and device control, while LLMs add a reasoning and natural language layer on top. This convergence can augment human expertise, enabling domain experts to interact with their IoT data more effectively and make informed decisions faster.

The outlined best practices emphasize that success lies not just in the technology, but in careful design: ensuring the right data feeds, creating the necessary ThingWorx entities, securing the AI services, and validating the outcomes. By following this blueprint, organizations in manufacturing, aerospace, electronics, automotive, and life sciences can confidently deploy LLM-powered IoT applications that drive higher productivity, better quality, and reduced downtime, all while keeping their data secure and their trust in AI well-founded.

Release Notes

Version 1.0 March 31, 2025
Initial Release

Version 1.1 May 29, 2025
Fixes:

- Broken reference link in Table of Contents
- Missing Best Practice name in footer

Changes:

- “Software Versions Tested” now ThingWorx 9.5.0 & above (was ThingWorx 9.3.7 & above)
- “White paper” now referred to as “document”
- Added disclaimer in “Prerequisites and Caveats” section
- Added “Document authored by PTC, with AI assistance” in footer

Version 2.0 June 30, 2025
Changes:

- Added new material relating to “Analytics Insights”
- Modified demo application to have many new services
- Functionality from previous version now referred to as “Data Insights”
- Modified names and code from previous services to indicate clarify responsibilities
- Modified Mashup to have two tabs; “Data Insights” and “Analytics Insights”
- Recommended LLM model updated from “GPT-4o” to “GPT-4.1”