Using ThingWorx with LLMs

Best Practice V1.1

Abstract

This document outlines best practices for integrating large language models (LLMs), specifically ChatGPT-4o, with ThingWorx, PTC's Industrial IoT platform. It emphasizes practical application through a detailed walkthrough of an example ThingWorx project.

David Kessler & Brad Cline
dkessler@ptc.com, bcline@ptc.com

Contents

Best Practice Intended Use 2
Introduction 2
Use Case 2
Intended Persona/Process 3
Intended Business Value 3
Prerequisites and Caveats 4
Overview of Best Practice 5
Data Preparation 5
Introduction 5
Demo Application Components 6
Project: “PTCDTS.TWXLLM” 6
Importing & Using the Demo Application 10
Link the Large Language Model (LLM) 13
Introduction 13
Set up LLM account access 13
Evaluate the Results 15
Validating and Governing LLM-Generated Insights in an IoT Application 15
Conclusion 17
Release Notes 18

Best Practice Name *Using ThingWorx with LLMs*	**Version** *V1.1*	**Version Date** *5/30/2025*	**Author** *David Kessler & Brad Cline* *Document authored by PTC, with AI assistance*
Software Versions Tested *ThingWorx 9.5.0 & above*	**Page** 1	**Please Send Feedback to** *dkessler@ptc.com, bcline@ptc.com*	**Reviewer**

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-4o, 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.

Use Case

This document focuses on a demo application that integrates ThingWorx with ChatGPT-4o. 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 a specific LLM service, the application and concepts it illustrates can be easily adapted to work with other data stores and other LLMs. The demo data used by the application is stored in a ThingWorx Data Table, for the sake of easy importing. The process covers extracting data from the Data Table, preparing it, and creating prompts that let ThingWorx talk with the LLM to deliver useful, natural language answers.

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 key rule is that the LLM should only provide suggestions and insights; it should 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-4o, 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. 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.

This document is intended for users working with ThingWorx 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

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

Introduction

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 one store, but the concepts and components illustrated in this paper can be expanded to accommodate more sources of data. The fields in the demo data used for this application 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

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: This demo uses a Data Table to make it easier for the user to import a single ThingWorx project and have a working ThingWorx/LLM application. 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”

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.
II. "GPTUrl" (String, persisted): Unique URL to the LLM API that the application will use to talk to the LLM.

Services

I. "CreatePrompt": This service takes as input the natural language 'userQuestion' entered by the user in the "PTCDTS.TWXLLM.DataQuery_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 to 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 must have 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 should be 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 will be 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. The LLM should be infer what the user is trying to ask for, and form the proper query, aggregate parameters, and answer preamble to address the user's question.
   - *Data table information*. This should contain the names of the columns in the Data Table, as well as their corresponding data types. This will help 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 will be 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 should be 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. It's also helpful to pass the types of aggregations that can be used as part of the parameters object.
   - *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 should be 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 may be 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 pairs of delimiters. These can be characters such as '<' and '>'. The 'ProcessQuestion' service will extract the different pieces from the pairs of delimiters and handle them appropriately.

II. "GetExampleQuestion": 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 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 the "CreatePrompt" service, 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 where the data for this demo resides. 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. "ProcessQuestion": This is the main service that orchestrates the flow of data and information through all the other services. It takes as input the string "userQuestion" that was entered by the user in the Mashup. This is then passed to the "CreatePrompt" 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 Mashup.

2. Data Table: "PTCDTS.TWXLLM.ExampleData_DT"

This Data Table is the ThingWorx entity that holds data for 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.DataQuery_MU"

This Mashup is the user interface for the application.

From the Data Table:

- It calls the 'GenerateRandomDemoData' service when 'Prepare Data' is clicked.
- It invokes the 'GetDataStatus' service by means of an Auto Refresh function that runs every second.

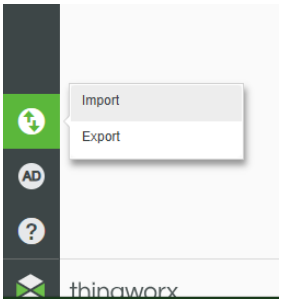
From the Manager Thing:

- It runs the 'GetExampleQuestion' service when the Mashup loads and displays the result in the dropdown.
- It calls the 'ProcessQuestion' service when 'Submit' is clicked.

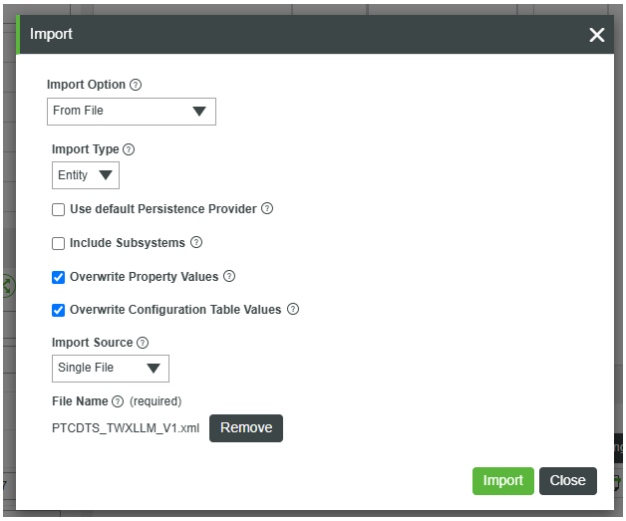
Importing & Using the Demo Application

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

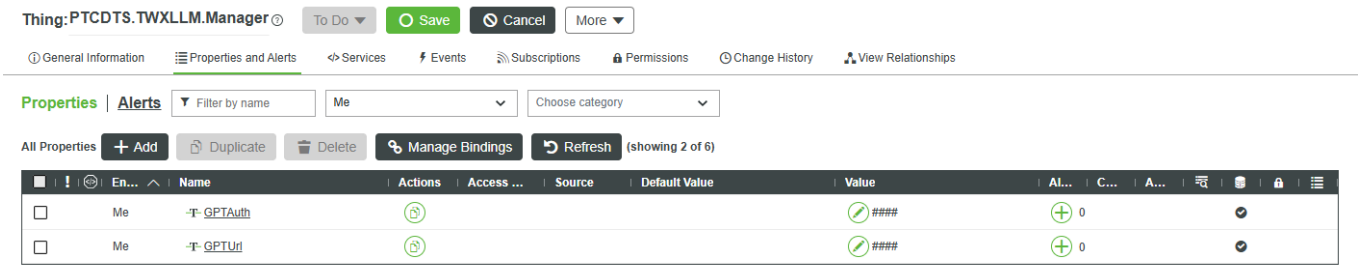
1.) Use the “Import” button along the bottom left of ThingWorx Composer



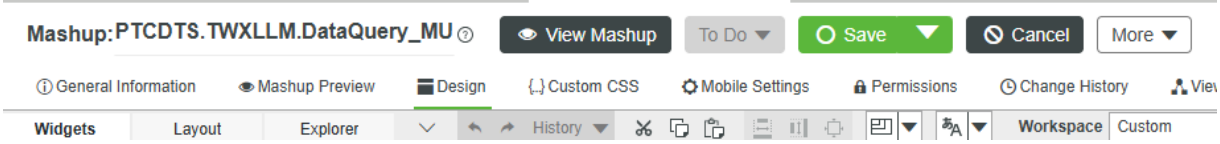
2.) Choose the “From File” import option and “Entity” import type. Then browse to the location where the “PTCDTS_TWXLLM_V1.xml” file is stored. Select the file and then click “Import”



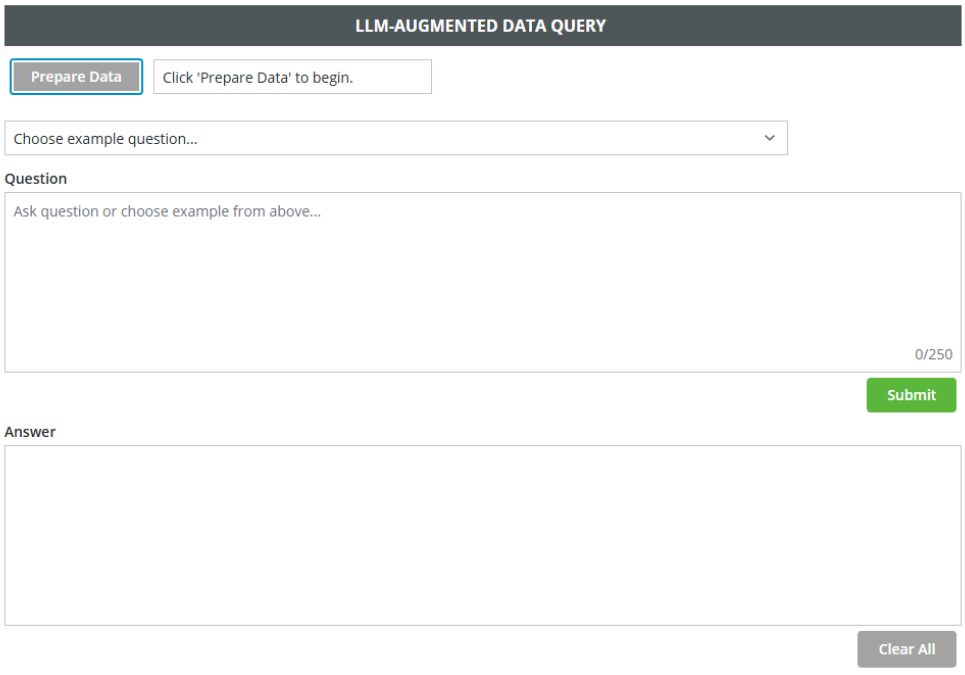
3.) Open the Manager Thing “PTCDTS.TWXLLM.Manager” and click the “Properties and Alerts” tab. Edit the values for the “GPTAuth” and “GPTUrl” properties. These are the values obtained when setting up the LLM account.



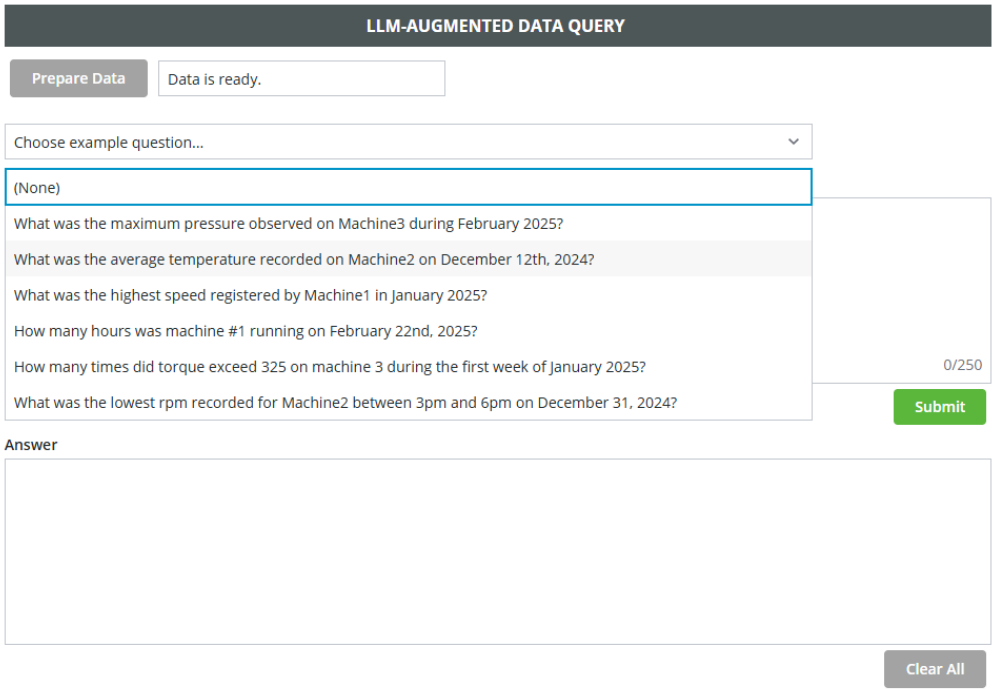
4.) Open the Mashup “PTCDTS.TWXLLM.DataQuery_MU” and click “View Mashup”.



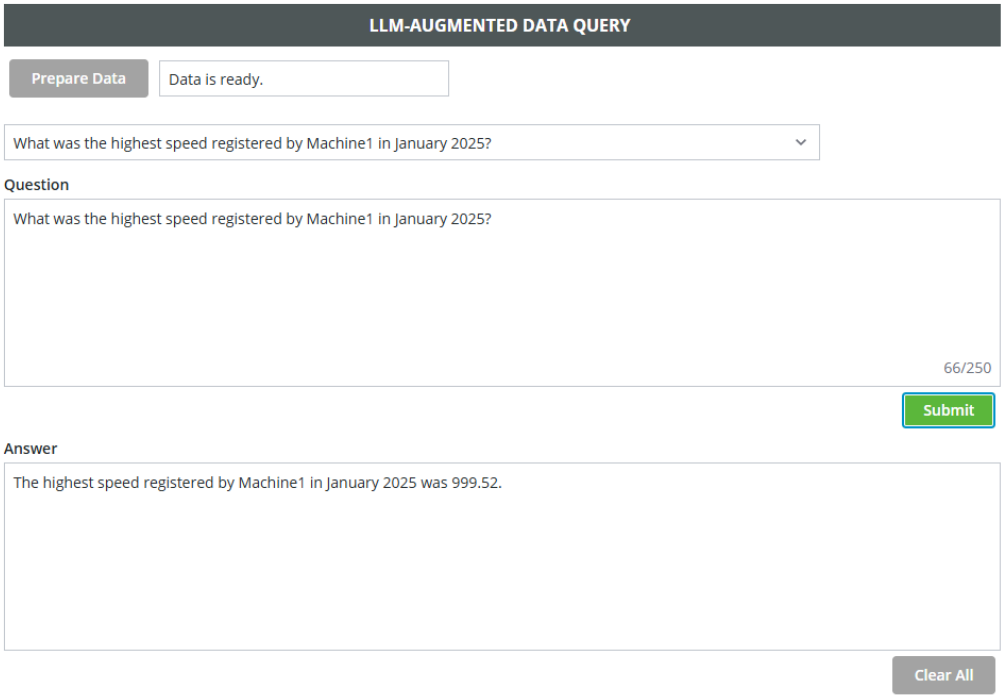
5.) Click “Prepare Data” along the top left of the Mashup, and wait for the preparation to complete.



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



7.) Submit the question and wait a couple seconds for a response.



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

Best Practice Name *Using ThingWorx with LLMs*	**Version** *V1.1*	**Version Date** *5/30/2025*	**Author** *David Kessler & Brad Cline* *Document authored by PTC, with AI assistance*
Software Versions Tested *ThingWorx 9.5.0 & above*	**Page** 18	**Please Send Feedback to** *dkessler@ptc.com, bcline@ptc.com*	**Reviewer**