THOMAS MORE

Aftersales AI Assistance

Realization document

PASCHAL CHUKWUBUIKEM IFEWULU

Student Bachelor Of Applied Computer Science

FEB 23RD 2026 – MAY 22ND 2026

Table of Contents

1. Glossary

Term / Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
Azure AI Search	Microsoft cloud search service used for indexing and retrieval
Azure Blob Storage	Cloud storage service used to store documents, videos, and processed files
Azure OpenAI	Microsoft Azure service used to access OpenAI language models
Bafang	Older electric drive system supplier used in some MAX bike configurations
BI	Business Intelligence
Chunk	Smaller section of a document used during retrieval and indexing
CSV	Comma-Separated Values file format used for structured data storage
E-MAX	Electric version of the MAX bike platform
FastAPI	Python backend framework used for the API and chatbot services
GPT	Generative Pre-trained Transformer language model
GPT-4.1	OpenAI language model tested during development for classification tasks
GPT-5	Main language model used for retrieval reasoning and response generation
Metadata	Structured information attached to indexed content to improve retrieval filtering
Mivice	Newer electric drive system supplier used in recent MAX bike configurations
OCR	Optical Character Recognition used for extracting text from video frames
Odoo	ERP platform used by Ahooga for ticket and operational management
RAG	Retrieval-Augmented Generation, an AI architecture combining retrieval and generation
Semantic Search	Search method based on contextual meaning rather than exact keywords
Tesseract	OCR engine used during video frame text extraction
Vector Search	Retrieval method based on embedding similarity
Embeddings	Numerical vector representations of text used for semantic retrieval

A-MAX	Non-electric version of the MAX bike platform
Arthur Interface	Internal interface used for rule management and validated response control
Drivetrain	Gear transmission configuration used on the bike
Alfine	Drivetrain configuration mainly associated with the Mivice system
Nexus	Drivetrain configuration commonly associated with the Bafang system
Derailleur	Drivetrain configuration mainly associated with the Mivice system

***Table 1**. Glossary of technical terms, abbreviations, and system-specific terminology used throughout the project.*

2. Introduction

This document describes the realization and development of the internship assignment carried out at Ahooga between 24 February 2026 to 22 May 2026. Ahooga is a Brussels-based company that designs and sells compact folding bikes and electric bikes focused on urban mobility.

During the first two weeks of the internship, a diagnostic and research phase was carried out to identify areas within the company where Artificial Intelligence could provide value. Several departments and workflows were analysed, and multiple opportunities for AI integration were identified. The after-sales department was selected as the main focus due to the high number of repetitive support tasks, the large amount of technical knowledge available, and the potential operational impact within the timeframe of the internship.

Based on these findings, the assignment was defined as the development of an AI-based after-sales assistant. The goal of the system is to support the after-sales department by improving how technical knowledge is accessed and used across different company knowledge sources. These sources include company documentation, manuals, videos, website information, and structured support ticket data extracted from Odoo.

The final implemented system focuses primarily on dealer support for the Ahooga MAX bike platform. The assistant is designed to help dealers access troubleshooting information more efficiently, provide structured technical guidance, and improve the consistency and speed of after-sales support.

This document builds further on the previously defined project plan and focuses on the realization and implementation of the solution. It describes the analysis performed before development, the technologies and architectural decisions that were made, and the implementation of the different parts of the system. In addition, the document explains how the assistant was tested, improved through continuous feedback, and prepared for further use and future expansion within the company.

The development process followed an iterative approach, where an early working version of the assistant was continuously improved through testing, debugging, validation sessions, and feedback from the after-sales department.

The assignment was carried out under the supervision of Hendrik Winkelmans (company mentor), with technical support from Steve and continuous operational feedback from Arthur Mesyngier, who is responsible for after-sales operations. Academic supervision was provided by Brent Pulmans from Thomas More.

3. Analysis

Before the development of the system started, an analysis phase was carried out to determine the most suitable approach, technologies, and architecture for the project. The objective was not only to build a functional chatbot, but to develop a reliable support system that could realistically be used within the after-sales operations of the company.

The initial project requirements and business objectives were defined together with the company supervisors based on the needs of the after-sales department. The technical analysis, experimentation, architectural design, and implementation decisions were carried out during the internship by the intern, with regular discussions and feedback from company mentors to validate the chosen direction.

Since the company already worked extensively within the Microsoft ecosystem, one of the first decisions was whether to use existing Microsoft services or introduce external platforms. After evaluating different possibilities, the decision was made to build the system mainly within the Azure environment. This approach simplified integration with existing company tools and also made the system easier to maintain and continue after the internship.

Another important part of the analysis phase was determining how the AI should use company knowledge. Two approaches were considered. The first option was directly training a model with company data and documentation. The second option was using a retrieval-based approach where the AI searches through structured company knowledge before generating an answer.

The retrieval-based approach (RAG – Retrieval Augmented Generation) was selected because it matched the requirements of the project better. Company knowledge changes regularly, new documents are added over time, and troubleshooting information continuously evolves. Retraining a model each time information changes would not have been practical within the context of the project.

Instead, the selected approach allows knowledge to be structured, indexed, and retrieved dynamically when a question is asked. This also provides more control over the generated responses since the answers are based on retrieved company information rather than only on the model's general knowledge. In addition, the approach made it easier to combine multiple knowledge sources such as manuals, technical documentation, processed video content, and structured ticket data. Website integration was also explored during development but was not fully implemented within the internship timeframe.

The architecture of the system was also analysed carefully before implementation. During the early stages of the project, tools such as Microsoft Copilot Studio and Power Automate were considered because they provide fast low-code integration possibilities. However, after testing

and experimentation, it became clear that more control was needed over the behaviour of the assistant, the retrieval logic, and the response generation process.
Because of this, the final implementation moved towards a more customized architecture using a FastAPI backend connected directly to Azure services. This provided much better control over:

- prompt management
- retrieval flow
- response formatting
- audience handling
- debugging and testing
- system behaviour adjustments

The final system uses several Azure services and supporting technologies that work together as one architecture.

Azure Blob Storage is used to store files such as manuals, PDF documents, images, and videos. Instead of processing files directly from local storage, the ingestion process was later redesigned so that documents and videos are retrieved directly from Blob Storage before processing and indexing. This improved the organization of the system and better reflected a cloud-based workflow.

Azure AI Search is used as the retrieval layer of the system. The indexed knowledge is stored here in structured form and is retrieved when the assistant receives a question. The system uses metadata and filtering to improve retrieval quality and reduce incorrect matches between different bike systems and components.

Azure OpenAI is responsible for generating the final responses. During the final stages of the project, the system was migrated to GPT-5 in order to improve reasoning quality, troubleshooting performance, and response structure. This significantly improved the quality of the generated answers compared to earlier versions of the assistant.

FastAPI is used as the backend layer of the system. This became one of the most important parts of the architecture because it acts as the central logic layer of the assistant. The backend handles:

- prompt management
- retrieval orchestration
- audience handling
- conversation flow
- integration between services
- response generation logic

In addition to the chatbot backend, an internal API layer was also developed during the project. This layer acts as a centralized connection point between company systems and future AI-related functionalities. The chatbot itself, the Arthur control interface, and Odoo integrations are connected through this API layer.

One of the objectives of this API structure was also to expose and organize Odoo fields and operational data in a reusable way. This allows the company to later reuse the same infrastructure for reporting, automation, or future AI applications beyond the chatbot itself.

Another important analysis topic was the handling of support tickets from Odoo. Initially, the goal was to directly integrate tickets into the knowledge base. However, after analysing the data quality, it became clear that many tickets contained duplicated information, incomplete troubleshooting steps, or inconsistent formatting.
Because of this, a cleaning and validation process was introduced. Tickets are first extracted, structured, and filtered before being considered for ingestion into the knowledge base. A flagging mechanism was also added to identify unreliable or duplicate cases. By the end of the internship, the ticket extraction and structuring process was completed, although the validated tickets had not yet been fully ingested into Azure AI Search.

The project also required analysis regarding the use of video content as a knowledge source. Many of the available support videos did not contain usable speech or transcripts, which made normal transcription approaches ineffective. To solve this, a custom video-processing workflow was designed where frames were extracted from videos at intervals and processed using text extraction techniques. The extracted information was then cleaned, summarized, and converted into structured text files before ingestion into the knowledge base.

Another important design decision involved audience separation. During development, it became clear that dealers and end customers require different levels of technical information. Customers should not receive detailed repair instructions or advanced troubleshooting procedures, while dealers require more technical operational guidance.

Different audience approaches were tested during the project. However, after evaluation and testing sessions with the company, the final implemented system focused primarily on dealer support for the MAX bike platform. This allowed the assistant to provide more technical and detailed responses while improving reliability and reducing ambiguity.

Overall, the analysis phase and the continuous evaluations during development played an important role in shaping the final architecture of the system. Many technical decisions evolved during the internship based on testing results, operational feedback, and practical constraints encountered during implementation.

4. Implementation

4.1 System Overview

The system developed during this internship is an AI-based after-sales assistant designed to support Ahooga's after-sales activities. Its main purpose is to help handle technical questions more efficiently by providing fast and reliable answers based on company knowledge.

The assistant is mainly intended for use by the after-sales team and dealers. By reducing the time spent manually searching through documents, videos, and previous troubleshooting information, the system helps improve the speed and consistency of after-sales support.

The assistant focuses mainly on technical questions related to Ahooga bikes, especially the MAX platform. These questions often involve troubleshooting, component issues, repair procedures, or identifying the correct configuration of a bike. One of the main challenges identified during the internship was that many support requests are repetitive, while still requiring accurate and structured answers. The assistant helps reduce this repetitive workload while maintaining response quality.

An important complexity of the system is the different configurations within the MAX product family. The MAX platform includes both the A-MAX (non-electric) and the E-MAX (electric) models, which can use different gear systems such as Nexus, Alfine, and Derailleur. For the E-MAX bikes, the gear system is directly related to the motor supplier platform used. E-MAX bikes using the Nexus gear system are associated with the older Bafang system, while E-MAX bikes using Alfine or Derailleur systems use the newer Mivice system. Since these systems use different components and troubleshooting procedures, the assistant must correctly identify the bike configuration before generating a response.

Since troubleshooting procedures and technical guidance differ between these systems, the assistant includes logic to identify or clarify the correct bike configuration before generating a response. This helps reduce incorrect troubleshooting suggestions caused by mixing procedures from different systems.

The solution is built using a retrieval-based approach. Instead of relying only on pre-trained knowledge, the assistant retrieves relevant information from a structured knowledge base before generating a response. The active knowledge base currently includes company documentation, manuals, and processed instructional video content.

Website integration and structured ticket ingestion were also explored during the project. Odoo tickets were successfully extracted, cleaned, structured, and validated through a filtering and flagging process. However, due to the internship timeframe, the validated tickets were prepared for future ingestion rather than fully integrated into the knowledge base during the final implementation.

When a question is asked, the assistant retrieves the most relevant information from the available knowledge sources and uses this information to generate a structured answer. This helps ensure that responses remain aligned with company documentation and validated troubleshooting knowledge.

An important part of the system is the internal feedback interface used by the after-sales team. Through this interface, users can add new answers, correct existing responses, and adjust how the assistant behaves in specific situations. These adjustments are stored and prioritized by the system when similar questions are asked again, allowing the assistant to improve continuously through operational feedback.

From a technical perspective, the solution is organized into multiple layers, including a user interface, a backend layer responsible for the system logic, and cloud-based AI and retrieval services. In addition, a centralized internal API layer was developed to connect the chatbot, the feedback interface, Odoo integrations, and future company systems through a reusable architecture.

Overall, the developed assistant helps reduce manual work, improves response speed, and provides more consistent technical support for the after-sales department.

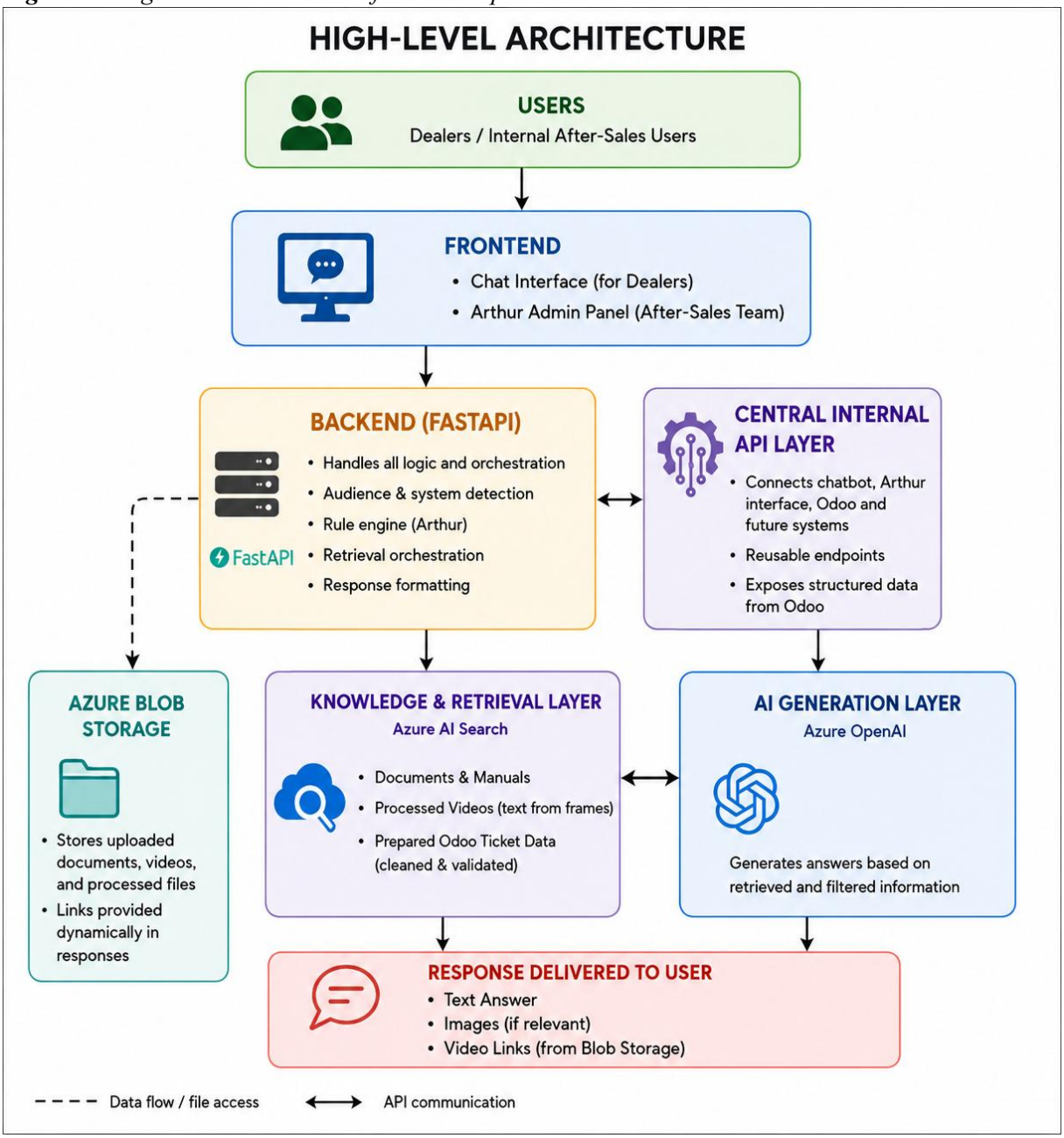
4.2 System Architecture

The system was designed as a controlled AI support assistant rather than a general-purpose chatbot. From the beginning of the project, one of the main objectives was to ensure that the responses generated by the assistant remained reliable, technically relevant, and aligned with Ahooga's internal troubleshooting knowledge.

During the early stages of development, different approaches and tools were explored, including low-code solutions such as Microsoft Copilot Studio. While these tools allowed fast prototyping, they provided limited control over the retrieval process, prompt structure, response behavior, and debugging capabilities. As the project evolved, it became clear that a more customizable architecture was required in order to achieve the expected response quality and system behavior.

For this reason, the final implementation was built around a custom backend architecture using FastAPI combined with Azure AI services. This architecture made it possible to control how information is retrieved, filtered, processed, and transformed into responses.

***Figure 1.**: High-level architecture of the developed AI assistant*



The final architecture is organized into multiple connected layers.

At the frontend level, the system includes the chatbot interface used by dealers and the internal control interface used by the after-sales department. The chatbot interface allows users to ask troubleshooting questions and receive technical guidance, while the internal interface allows the after-sales team to manage and improve the assistant without directly modifying the code.

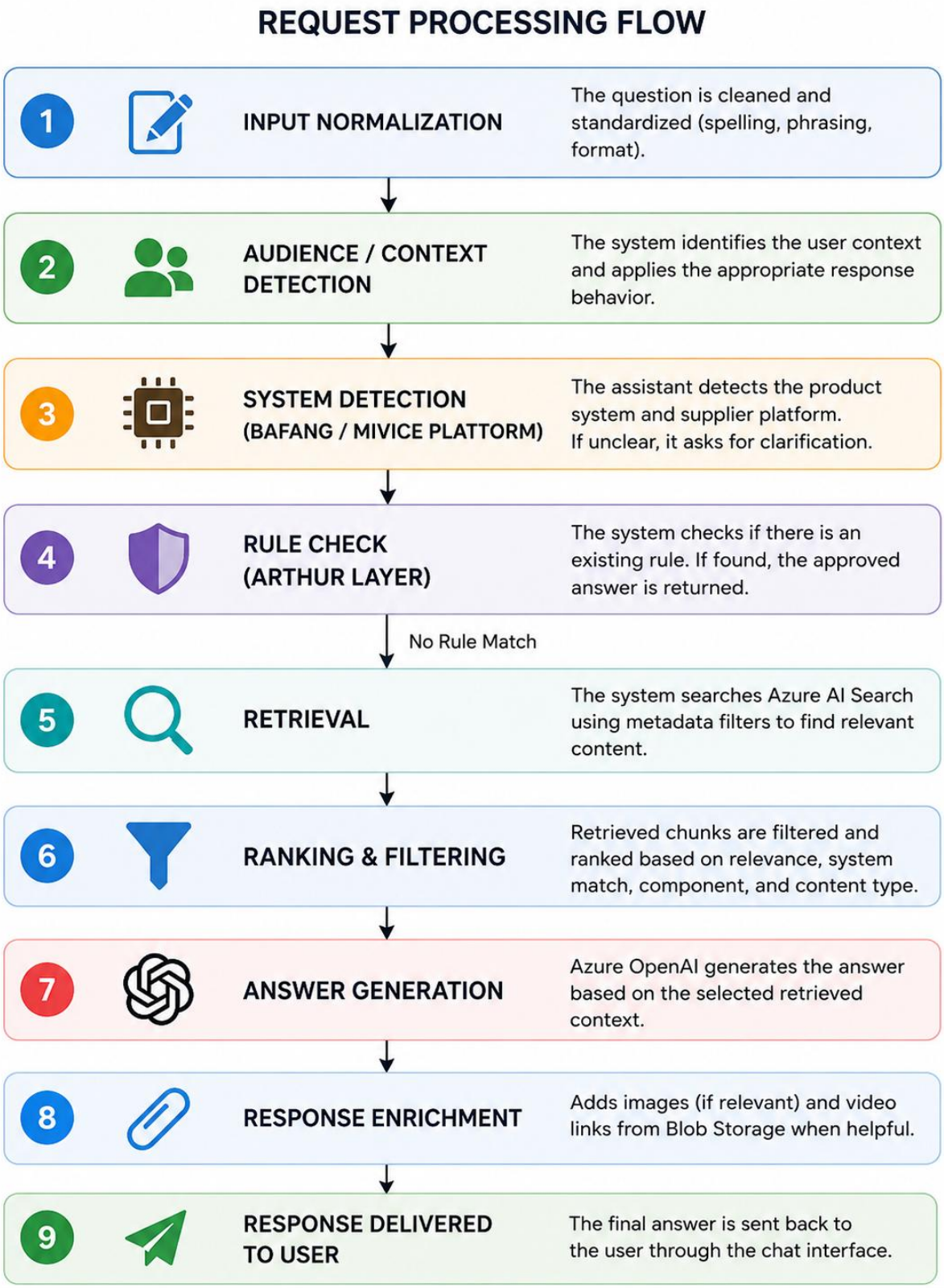
The backend layer acts as the central logic component of the system. This layer was implemented using FastAPI and became one of the most important parts of the architecture. The backend controls:

- prompt management

- conversation flow
- retrieval orchestration
- response formatting
- audience handling
- system detection logic
- communication between Azure services

The backend also handles the decision-making process before a response is generated. Instead of immediately sending every question directly to the language model, the request first passes through several validation and retrieval steps.

Figure 2. *Request processing flow from user question to final response*



One important part of this process is the handling of different bike configurations within the MAX product family. The MAX platform includes both A-MAX (non-electric) and E-

MAX (electric) models, which can use different gear systems such as Nexus, Alfine, and Derailleur.

For the E-MAX bikes, the gear system is directly related to the supplier platform used. E-MAX bikes using the Nexus gear system are associated with the older Bafang system, while E-MAX bikes using Alfine or Derailleur systems use the newer Mivice system. Since these systems use different components and troubleshooting procedures, the assistant must correctly identify the bike configuration before generating a response.

To reduce incorrect troubleshooting suggestions, the backend includes logic that attempts to detect the correct configuration automatically based on the user's question and context. If the system cannot determine the correct configuration reliably, the assistant asks for clarification before continuing.

The retrieval layer of the architecture is built using Azure AI Search. This service stores and indexes the knowledge used by the assistant. Instead of storing information as large documents only, the knowledge is processed into smaller chunks that can be searched and retrieved more efficiently.

The active knowledge base contains:

- technical manuals
- repair documentation
- instructional content
- processed video information

Each stored chunk also contains metadata used for filtering and retrieval. This metadata includes information such as:

- bike model
- system type
- document category
- technical component
- content source

This structure helps improve retrieval accuracy and reduces the risk of mixing information from incompatible systems.

Another important part of the architecture is the handling of support videos. A large number of the available videos did not contain usable transcripts or spoken explanations. Because of this, a separate processing workflow was designed during the project.

Instead of relying on audio transcription, frames were extracted from the videos at intervals. Text visible inside the frames was then extracted and cleaned before being converted into structured text summaries. These summaries were later processed and added to the knowledge base.

Example of video frame extraction and text processing workflow

Figure 3. Original instructional video files

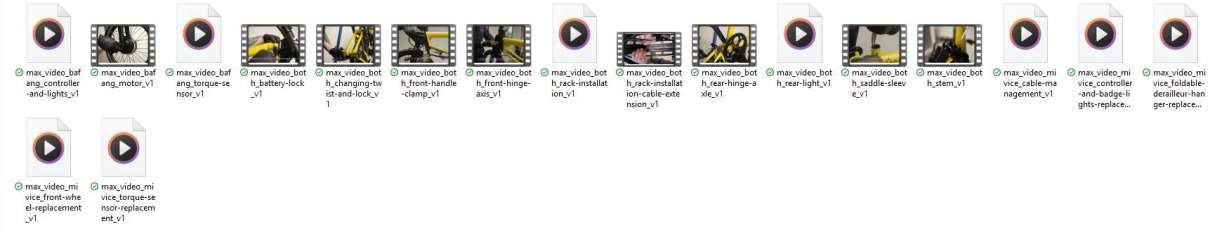


Figure 4. Extracted frames used for text processing

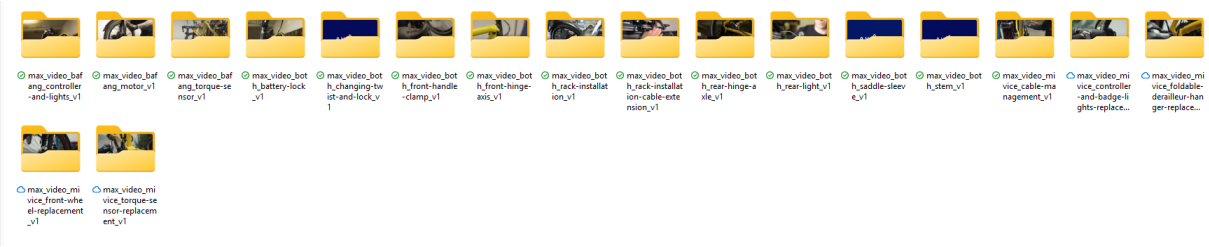
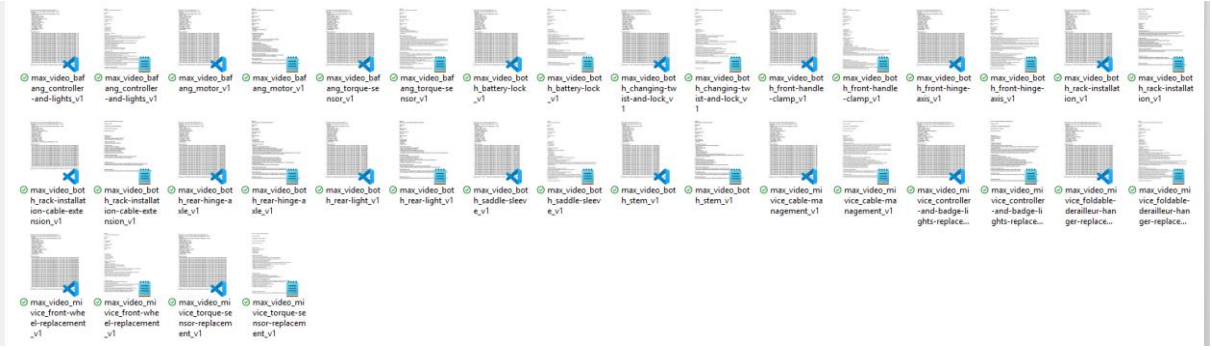


Figure 5. Structured TXT and CSV outputs generated for ingestion



This approach allowed video content to become searchable knowledge even when no usable audio transcription was available.

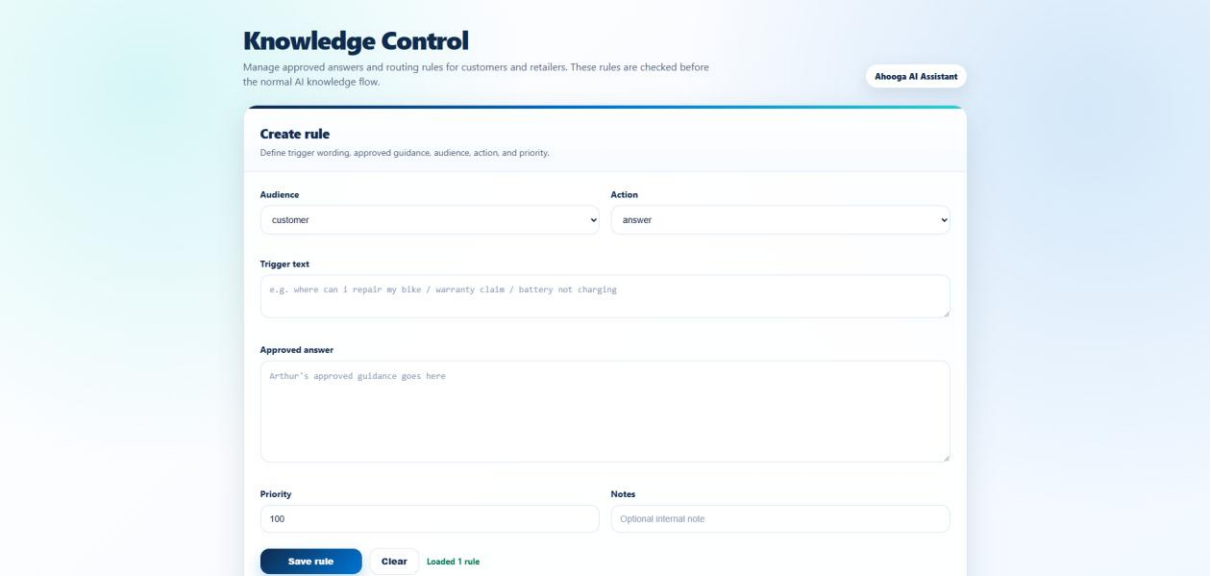
The architecture also includes an internal control layer referred to during development as the “Arthur interface.” This interface was created to allow the after-sales department to improve and control the assistant without requiring programming knowledge.

Through this interface, users can:

- add approved answers
- correct existing responses
- define response priorities
- improve assistant behavior for recurring questions

When similar questions are asked later, the assistant prioritizes this validated information before generating a response. This helped improve response consistency during testing and reduced the number of unreliable answers.

***Figure** 7. Internal after-sales control interface*



Another important part of the final implementation is the centralized internal API layer developed during the internship. Initially, the API was mainly created to support the chatbot backend and Odoo integration. However, during development, the structure evolved into a more reusable internal API architecture.

The final API layer acts as a centralized connection point between:

- the chatbot backend
- the Arthur interface
- Odoo integrations
- ticket extraction processes
- future company systems

The API layer also exposes and organizes selected Odoo fields and operational information in a reusable way. One of the goals of this structure was to allow the company to later reuse the same infrastructure for reporting, automation, or future AI-related applications.

Azure Blob Storage is used to store uploaded documents, processed files, images, and videos. During the final stages of the project, the ingestion process was improved so that files are processed directly from Blob Storage instead of local storage. This created a more organized cloud-based workflow and simplified the ingestion pipeline.

The generated responses are produced using Azure OpenAI. During the final optimization phase of the project, the system was migrated to GPT-5 in order to improve reasoning

quality, troubleshooting performance, and response structure. This significantly improved the reliability and quality of the generated answers compared to earlier versions of the assistant.

The overall architecture evolved continuously throughout the internship. Several parts of the system, including prompt structure, retrieval logic, response handling, and ingestion workflows, were redesigned multiple times based on testing sessions, debugging, and operational feedback from the after-sales department.

By the end of the internship, the system had evolved from an early prototype into a functional after-sales assistant capable of providing structured dealer-focused technical support using company knowledge and controlled retrieval logic.

4.3 Knowledge Base and Data Processing

One of the most important parts of the project was the design and preparation of the knowledge base used by the AI assistant. Since the quality of the assistant depended heavily on the quality of the retrieved information, a large part of the implementation focused on organizing, processing, and structuring the available company knowledge in a consistent way.

The knowledge base was designed to support technical after-sales questions related to the Ahooga MAX platform. The system combines multiple knowledge sources, including technical manuals, assembly documentation, troubleshooting guides, processed video content, and structured ticket information. Instead of relying only on general AI knowledge, the assistant retrieves information directly from these company-specific sources before generating a response. This approach improved the reliability and technical accuracy of the answers.

Knowledge Source Organization
To ensure that the retrieval system could work efficiently, the documents were first organized in Azure Blob Storage using a structured folder hierarchy. The storage structure was divided by knowledge type and bike platform. Separate containers were used for documents and processed video summaries.

The document storage container included categorized folders such as manuals, assembly guides, troubleshooting documentation, warranty information, and system-related documents. Inside these folders, the files were further grouped based on the MAX platform.

structure used in Azure Blob Storage for organizing the document knowledge base.

Figure 8.

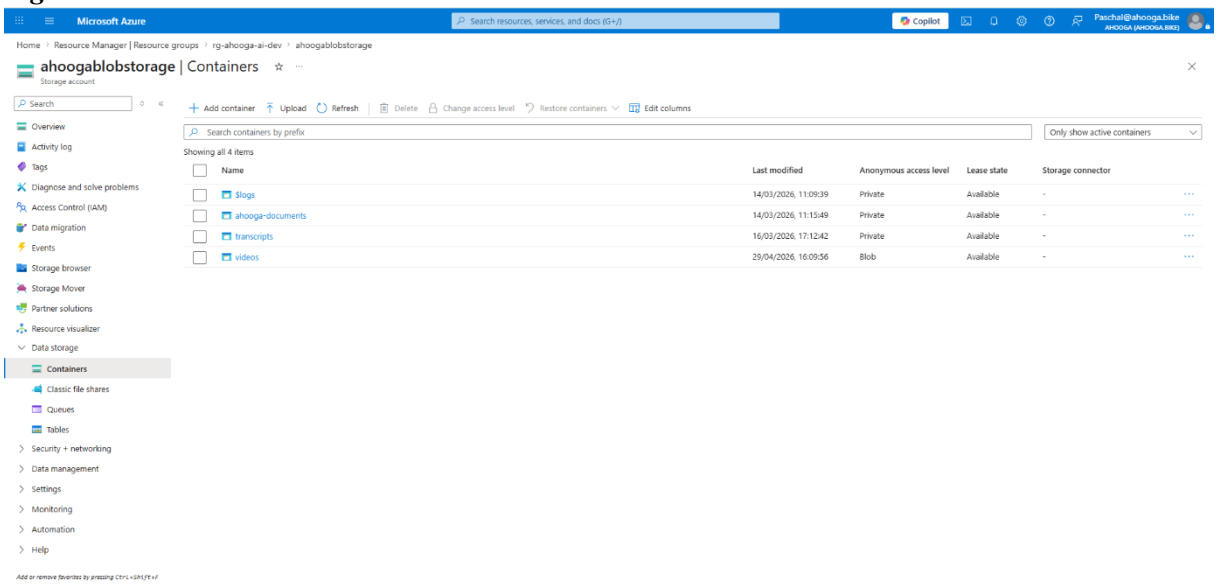


Figure 9.

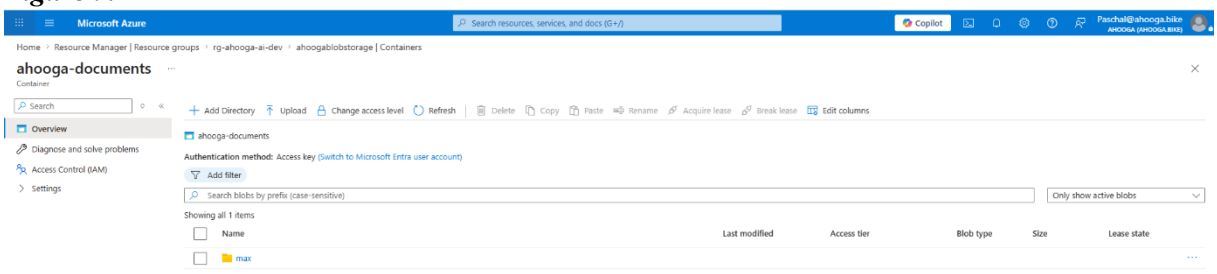
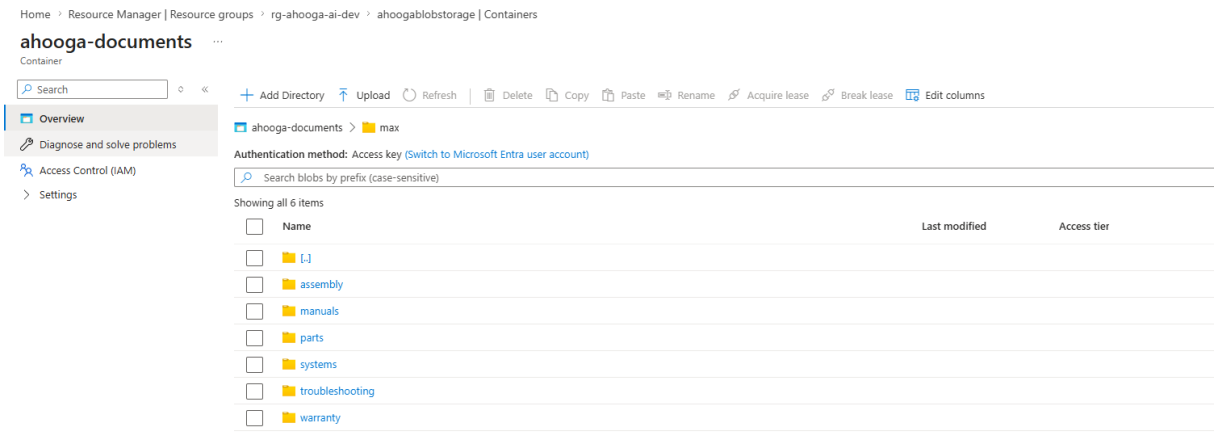


Figure 10.



A consistent naming strategy was also used for all uploaded files. This naming structure became an important part of the metadata system because the retrieval process later depended on these values for filtering and search optimization.

An example filename is shown below:

`max_assembly_bafang_controller_v1.pdf`

The different sections of the filename contain specific information about the document.

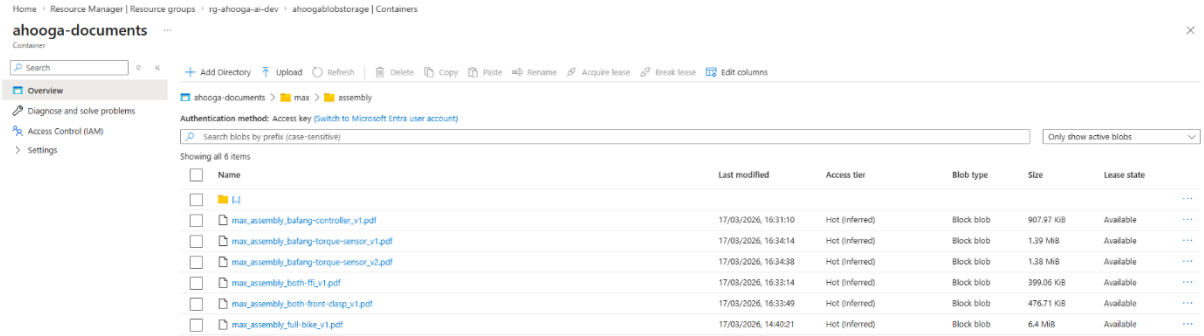
Naming Segment	Description
max	Bike platform
assembly	Document category
bafang	System supplier
controller	Related component
v1	Document version

***Table 2**. Explanation of the filename metadata structure used for indexed documents.*

This structure made it possible to automatically extract metadata during ingestion without manually defining information for every document. It also improved consistency across the knowledge base and simplified filtering during retrieval.

The same naming logic was used across the different document categories to maintain uniformity throughout the system.

***Figure 11**. an example of the document naming structure used in the storage container.*



The organization of the files became especially important because the assistant needed to distinguish between different systems and compatibility situations. For example, some technical procedures are specific to the older Bafang system, while others apply to the

newer Mivice system. By embedding this information directly into the metadata, the retrieval layer could filter results more accurately before generating a response.

Video Processing and Transcript Generation

Besides written documentation, instructional and repair videos were also integrated into the knowledge base. Since videos cannot be searched directly by the retrieval system, a preprocessing pipeline was created to convert the video content into searchable text.

The processing was performed locally before ingestion into Azure AI Search. The pipeline extracted frames from videos at fixed time intervals and applied OCR and text processing techniques to generate structured summaries.

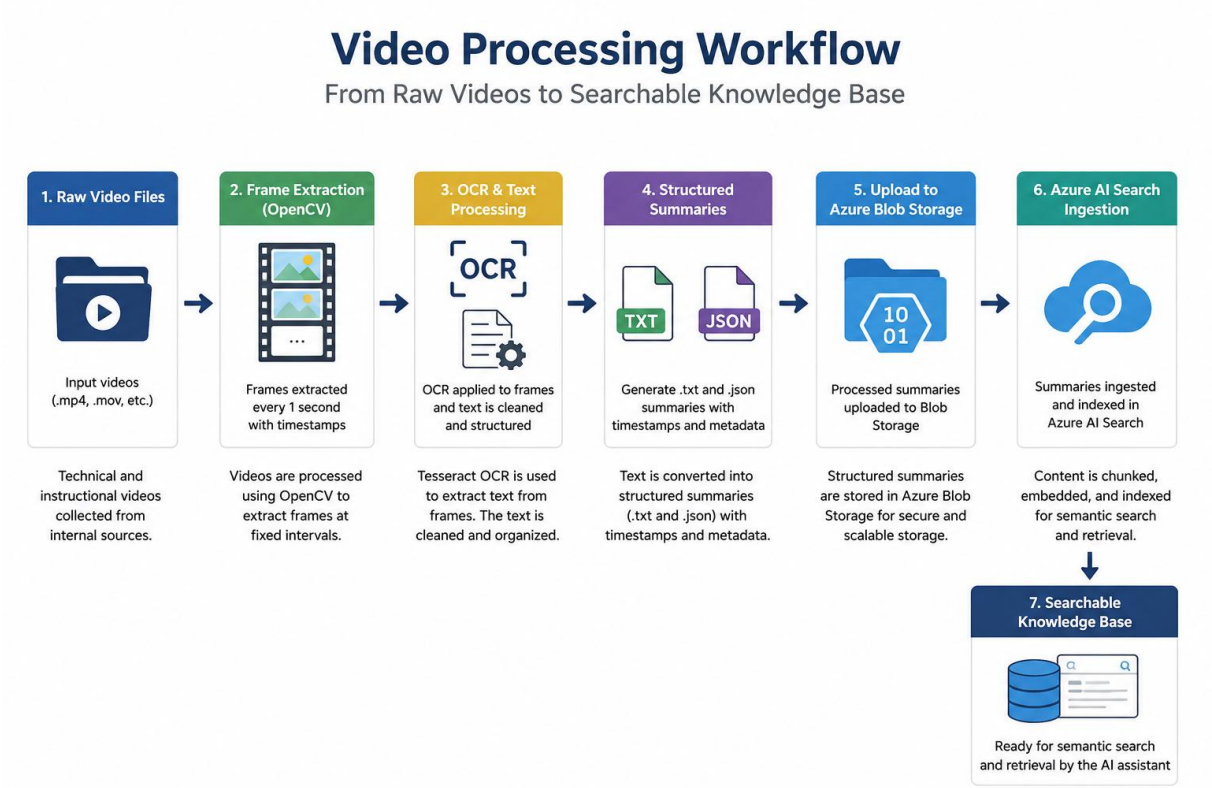
The frame extraction process was implemented using OpenCV. Frames were extracted every second and saved with timestamp-based filenames.

A simplified example of the frame extraction logic is shown below:

```
FRAME_EVERY_SECONDS = 1.0
frame_interval = max(1, int(fps * every_seconds))
output_name = f"{video_path.stem}_t{timestamp_seconds:07.2f}.jpg"
```

This approach made it possible to associate extracted text with specific moments inside the videos.

***Figure 12**. workflow used for video processing, including raw videos, extracted frames, and generated transcript outputs.*



After processing, structured .txt and .json summaries were generated for each video. These summaries were then uploaded to Azure Blob Storage and later ingested into Azure AI Search together with the written documentation.

The processed video summaries followed the same metadata structure used for the documents. This ensured that both documents and video-based knowledge could be retrieved using the same filtering logic.

the transcript storage structure used for processed video summaries.
Figure 13.

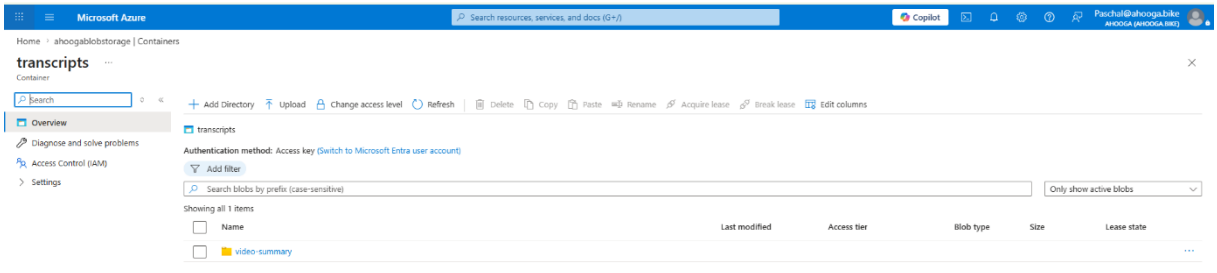


Figure 14.

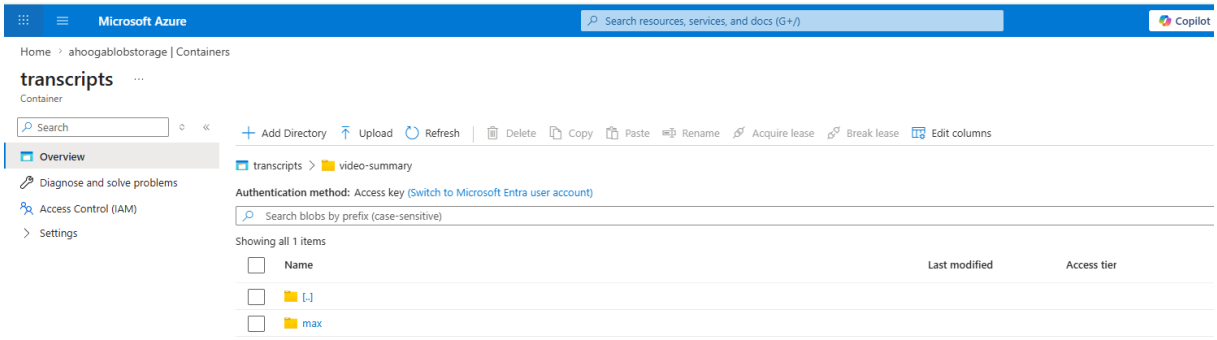
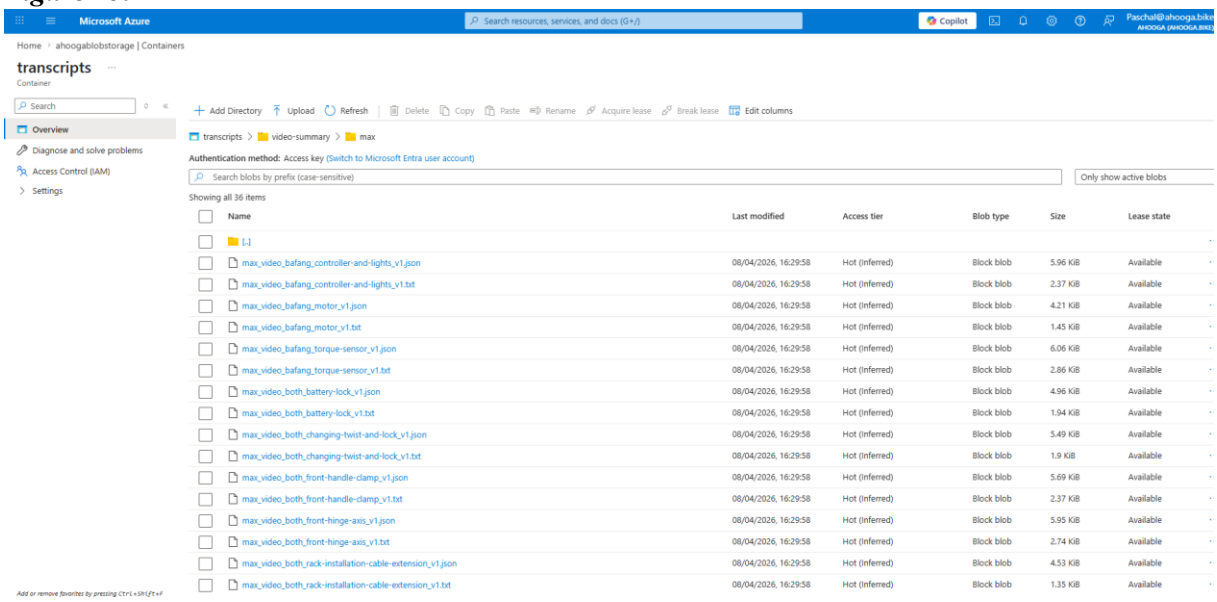


Figure 15.



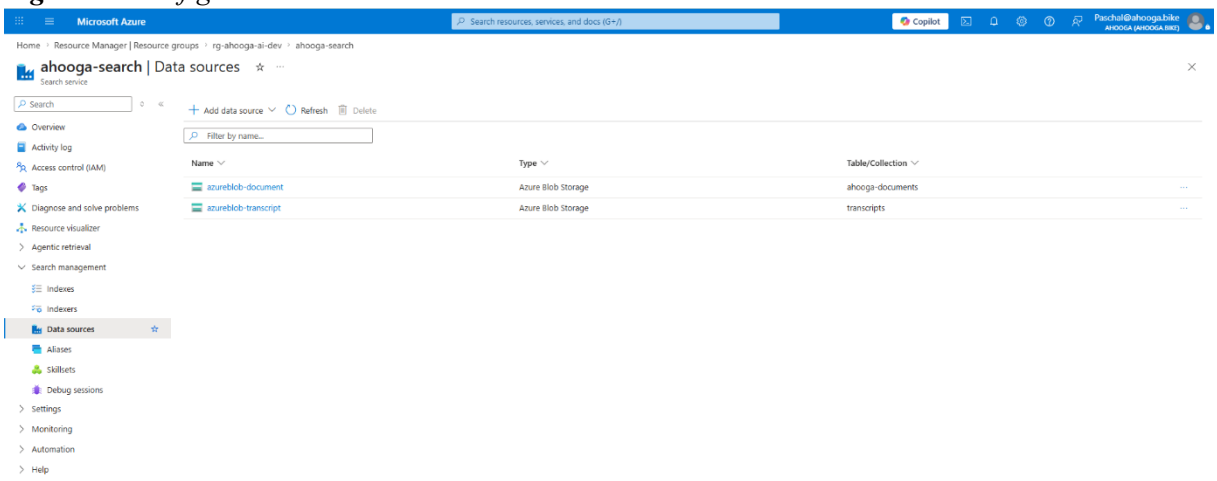
The original raw videos were kept separately for future use, while the processed summaries became part of the searchable knowledge base used by the assistant.

Azure AI Search Index Structure

After organizing the documents and video summaries, the next step was indexing the knowledge into Azure AI Search. Azure AI Search was used as the central retrieval layer of the system and stored both the searchable text chunks and their associated metadata.

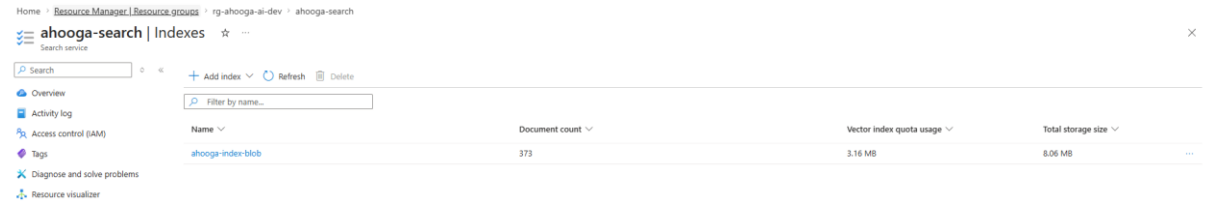
Separate data sources were configured for documents and transcript summaries. These data sources connected directly to the Azure Blob Storage containers.

***Figure 17**. configured Azure AI Search data sources.*



The indexed content was stored inside a unified search index that combined documentation and processed video summaries into a single searchable knowledge base.

***Figure 18**. configured Azure AI Search index.*



During ingestion, each document was split into smaller chunks before indexing. This improved retrieval precision because the system could retrieve only the most relevant sections instead of entire documents.

In addition to the text content itself, several metadata fields were indexed to support filtering and retrieval optimization.

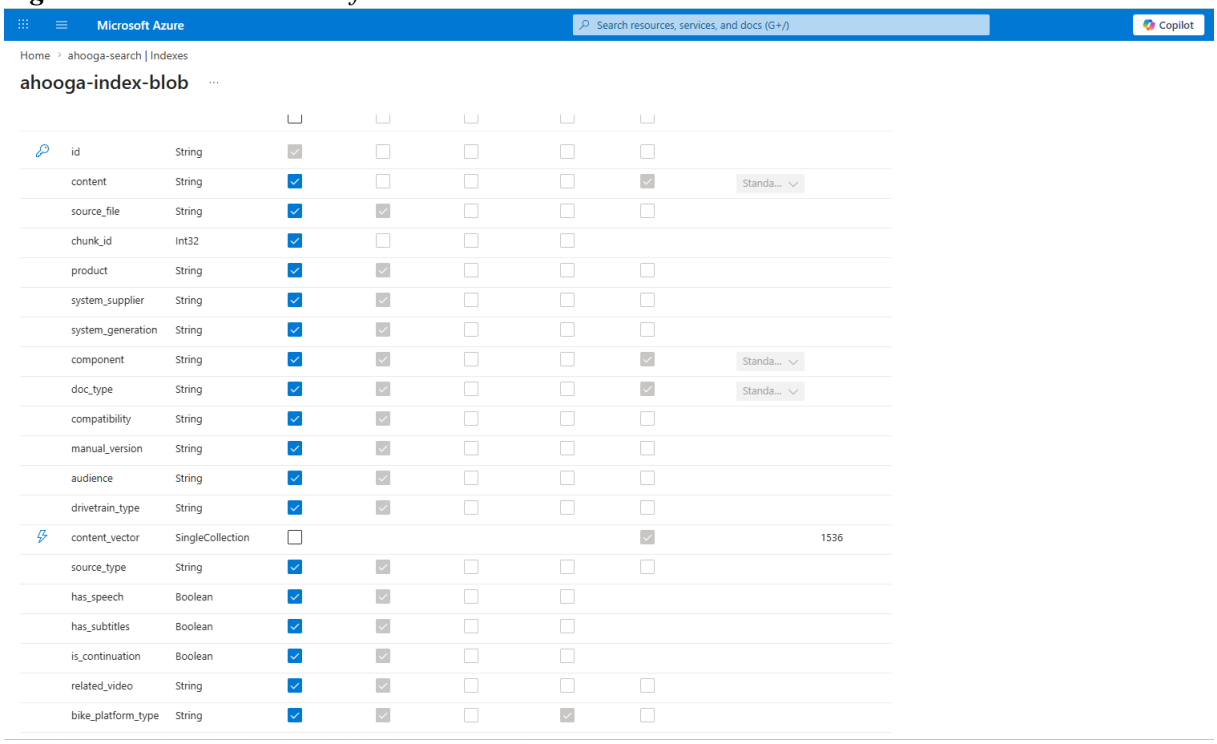
Field Name	Purpose
id	Unique identifier for each indexed chunk
content	Searchable text content extracted from documents or video summaries
source_file	Original file name used for traceability
chunk_id	Identifier used to separate document chunks
product	Bike platform associated with the content
system_supplier	Supplier system related to the content (Bafang or Mivice)
system_generation	Indicates old or new system generation

component	Related bike component or subsystem
doc_type	Type of document (manual, assembly, troubleshooting, video, etc.)
compatibility	Defines compatibility restrictions for retrieval
manual_version	Document version information
audience	Intended audience such as dealer or customer
drivetrain_type	Related drivetrain configuration
content_vector	Vector embedding used for semantic retrieval
source_type	Indicates whether the source originated from documents or video summaries
has_speech	Indicates whether speech was detected in processed video content
has_subtitles	Indicates whether subtitles were available
is_continuation	Indicates whether the chunk continues from a previous section
related_video	Reference to the associated source video
bike_platform_type	Platform category associated with the indexed content

***Table 3**. Explanation of the metadata fields used in the Azure AI Search index for retrieval and filtering.*

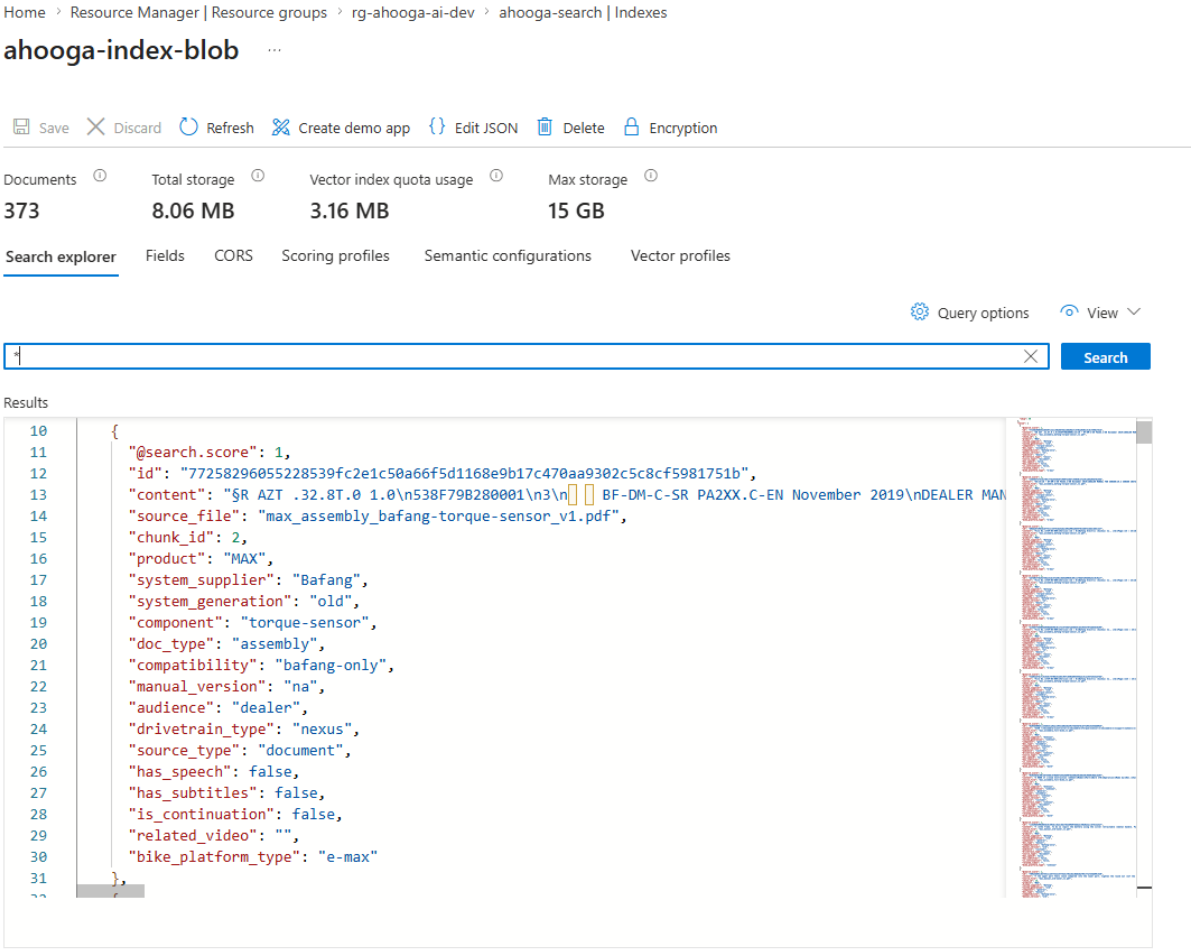
The metadata fields played an important role during retrieval because the backend could apply filters before generating answers. For example, the assistant could prioritize dealer-specific information or restrict results to documents related to a specific supplier system.

Figure 19. *indexed metadata fields inside Azure AI Search.*



The system also used vector embeddings for semantic retrieval. Each chunk of content was converted into a vector representation using the deployed embedding model. This allowed the assistant to retrieve semantically related information instead of relying only on keyword matching.

Figure 20. *an example of indexed chunks inside the search explorer.*



Data Ingestion and Configuration

To automate the ingestion process, multiple ingestion scripts were created for different types of knowledge sources. Separate scripts were used because documents and videos required different preprocessing steps.

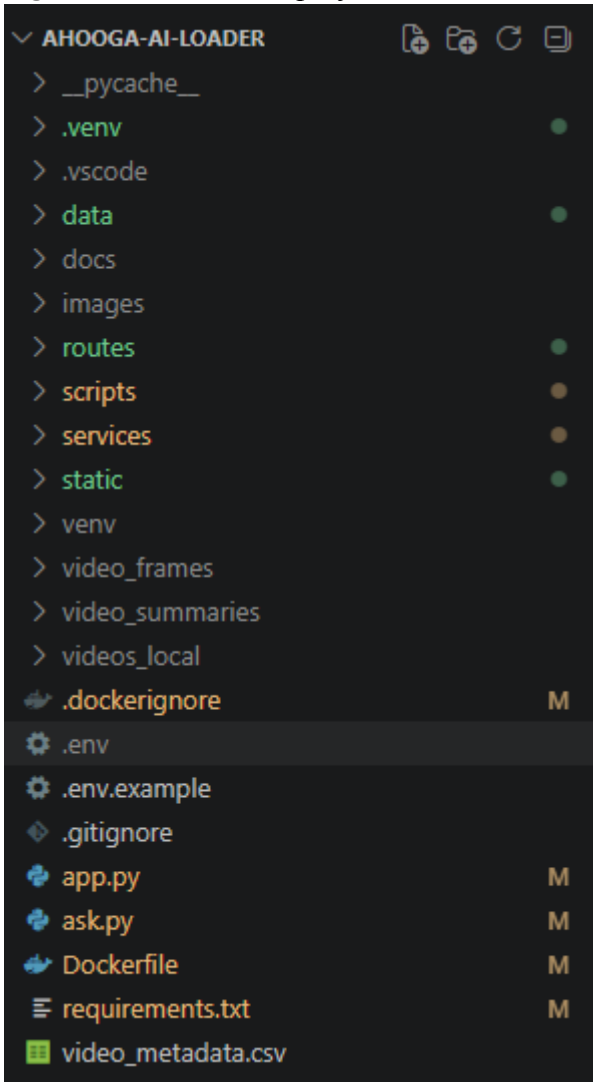
The main ingestion scripts included:

Script	Purpose
ingest.py	Document ingestion
ingest_videos.py	Video summary ingestion
extract_video_frames.py	Video frame extraction
process_video_frames.py	OCR and text processing

Table 4*. Overview of the main ingestion and preprocessing scripts used in the knowledge base pipeline.*

The backend structure was designed in a modular way to separate ingestion, routing, services, and frontend components. This improved maintainability and simplified future extensions of the system.

Figure 21. *the backend project structure used during development.*



The system configuration was managed using environment variables. This approach allowed the same codebase to be used both locally and in the deployed Azure environment while keeping sensitive information outside the source code.

An example of the configuration structure is shown below:

```
DOCS_BLOB_CONTAINER=ahooga-documents
DOCS_BLOB_PREFIX=max/
AZURE_SEARCH_INDEX=ahooga-index-blob
TRANSCRIPTS_BLOB_CONTAINER=transcripts
TRANSCRIPTS_BLOB_PREFIX=video-summary/max/
```

Sensitive values such as keys, endpoints, and connection strings were intentionally excluded from the project files for security reasons.

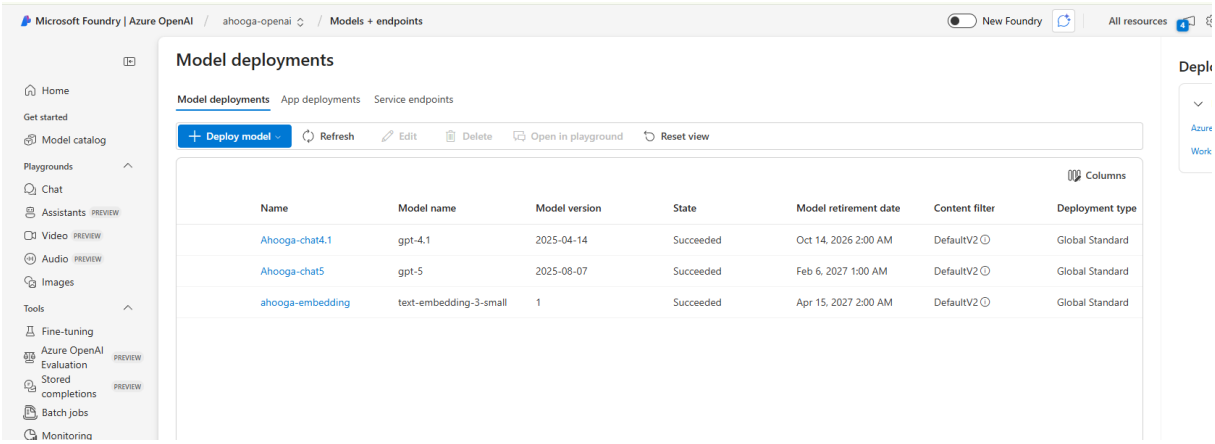
OCR processing used Tesseract during the local preprocessing stage. This was mainly required during video frame processing and was not needed in the deployed Azure environment because the summaries had already been generated before deployment.

Model Configuration and Retrieval Optimization

The retrieval and answer generation pipeline used Azure OpenAI services. Different model configurations were tested during development to balance response quality and response speed.

The final implementation mainly used GPT-5 for both classification and answer generation because it produced the most reliable and technically accurate responses during testing. The system also used the text-embedding-3-small embedding model for semantic retrieval.

Figure 22. *the deployed Azure OpenAI models used during the project.*



During optimization, GPT-4.1 was also tested as a lightweight classification model while GPT-5 handled answer generation. The goal was to improve response speed while maintaining answer quality. Although this approach slightly improved performance speed, the overall response quality and consistency were better when GPT-5 handled the complete reasoning pipeline. For this reason, the final implementation continued using GPT-5 as the primary model.

Even though the response quality became very strong, response speed remained one of the areas identified for future optimization. This was mainly due to the complexity of the retrieval pipeline and the use of a larger reasoning model. However, the final version of the assistant already achieved a strong balance between technical accuracy, retrieval quality, and practical usability for the after-sales team.

4.4 Retrieval and Response Workflow

After the knowledge base and indexing process were completed, the next important part of the system was the retrieval and response workflow. This workflow controls how the assistant processes a user question, retrieves the most relevant information from the knowledge base, and generates a final response.

The main objective of this workflow was not simply to generate answers, but to generate reliable and context-aware answers based on the correct bike configuration, supplier system, and technical context. To achieve this, the assistant combined structured retrieval, metadata filtering, prompt engineering, and lightweight backend logic.

Instead of relying heavily on hardcoded conditions, most of the intelligence of the system came from the quality of the prompts, the metadata structure, and the retrieval pipeline. The backend mainly acted as an orchestration layer that connected the different services together and controlled the overall flow of the system.

User Question Processing
The workflow starts when a dealer submits a question through the chat interface. Before the question is sent to the language model, the backend first performs several preprocessing and context detection steps.

The first step is input normalization. This includes cleaning the question, standardizing formatting, and preparing the text for classification and retrieval. The backend then attempts to identify important contextual information from the question itself.

One important part of the workflow is the identification of the bike configuration. Since different MAX bike configurations require different troubleshooting procedures, the system tries to determine the correct configuration before retrieval begins.

The assistant supports both:

- A-MAX (non-electric platform)
- E-MAX (electric platform)

The system also considers drivetrain configurations such as:

- Nexus
- Alfine
- Derailleur

These drivetrain configurations are important because they help identify the supplier platform used by the bike. For example:

- Nexus configurations are typically associated with the older Bafang system
- Alfine and Derailleur configurations are generally linked to the newer Mivice system

In some situations, the assistant can infer the likely bike type directly from the question itself. For example, if the user mentions components such as the motor, battery, or torque sensor, the system can already assume that the question relates to the E-MAX platform. This context detection improves retrieval quality and helps reduce incompatible troubleshooting responses.

*The complete request processing workflow was previously presented in **Figure 2** in the **system architecture** section. The following sections explain each stage of the workflow in more detail.*

The backend used lightweight logic together with prompt-based reasoning. Instead of building a large hardcoded rule system, the prompts and metadata filtering allowed the assistant to dynamically understand the technical context of the question.

Metadata-Based Retrieval

Once the technical context of the question is identified, the backend generates metadata filters that are used during retrieval in Azure AI Search.

The retrieval system uses the metadata fields created during the ingestion process to narrow down the search space before retrieving content. This improves retrieval precision and prevents unrelated information from being selected.

The filtering process considers:

- bike platform
- supplier system
- drivetrain type
- component category
- document type
- audience type

A simplified example of the filtering logic is shown below:

```
filters: List[str] = ["product eq 'MAX'"]

if platform in {"a-max", "e-max"}:
    filters.append(
        f"(bike_platform_type eq '{platform}' or bike_platform_type eq 'both' or bike_platform_type eq
'unknown')"
    )

if drivetrain in {"nexus", "alfine", "derailleur"}:
    filters.append(
        f"(drivetrain_type eq '{drivetrain}' or drivetrain_type eq 'both' or drivetrain_type eq 'unknown')"
    )

if system in {"Bafang", "Mivice"}:
    filters.append(
        f"(system_supplier eq '{system}' or system_supplier eq 'Both' or system_supplier eq 'Unknown')"
    )
```

The retrieval process was designed to remain flexible. In situations where some metadata could not be identified confidently, the retrieval system still allowed broader matching instead of fully blocking results. This reduced the risk of returning empty responses while still prioritizing the most relevant information.

The system also supported semantic vector retrieval using embeddings. Instead of relying only on keyword matching, Azure AI Search converted the indexed chunks into vector representations and retrieved semantically related content.

A simplified example of the vector retrieval process is shown below:

```
vector_query = VectorizedQuery(
    vector=vector,
    k_nearest_neighbors=top_k,
    fields="content_vector",
)

results = search_client.search(
    search_text="",
    vector_queries=[vector_query],
    filter=filter_expr,
    top=top_k,
)
```

This retrieval approach significantly improved the quality of troubleshooting responses because the assistant could retrieve conceptually related information even when the exact wording of the question did not match the original document text.

Context Ranking and Selection

After retrieval, the returned chunks were not immediately sent to the language model. The backend first evaluated the retrieved results and prioritized the most relevant content.

The ranking process considered:

- metadata compatibility
- supplier system matching
- drivetrain relevance
- document category
- semantic similarity

This additional ranking step helped reduce noisy retrieval results and improved answer consistency.

The assistant also prioritized troubleshooting-related documents when technical repair questions were detected. This was important because troubleshooting guides usually provided more direct operational instructions compared to general manuals.

Another important objective of the ranking process was reducing hallucinations. Since the assistant generated answers only from retrieved context, selecting high-quality chunks became critical for maintaining reliable responses.

Response Generation
Once the final retrieval context was selected, the information was passed to Azure OpenAI for answer generation.

The final implementation mainly used GPT-5 for both classification and answer generation because it produced the most consistent and technically reliable responses during testing.

The prompts used during generation were continuously adjusted throughout development to improve:

- technical consistency
- retrieval grounding
- response formatting
- troubleshooting accuracy
- dealer-focused behavior

The assistant was instructed to generate answers only from the retrieved context rather than relying on general knowledge. This reduced the risk of unsupported technical instructions.

The backend also maintained lightweight conversation state management. This allowed the assistant to remember previously identified information during the conversation, such as:

- detected bike type
- drivetrain configuration
- supplier system
- previously discussed components

This made follow-up interactions smoother and reduced the need for users to repeat technical details multiple times.

The connection to Azure services was managed using environment variables both locally and in the deployed Azure environment. This made deployment easier while keeping configuration values separated from the source code.

An example of the configuration structure is shown below:

```
AZURE_SEARCH_INDEX=ahooga-index-blob
AZURE_OPENAI_ENDPOINT=***
AZURE_OPENAI_API_KEY=***
DOCS_BLOB_CONTAINER=ahooga-documents
TRANSCRIPTS_BLOB_CONTAINER=transcripts
```

Sensitive values such as API keys and endpoints were intentionally excluded from the document for security reasons.

Model Testing and Optimization
Different model configurations were tested during development to improve both response quality and speed.

Initially, GPT-5 handled both the classification tasks and final answer generation. The response quality was very strong, especially for technical troubleshooting and dealer-related questions. However, response speed was slower than expected for real-time support usage.

To improve performance, another configuration was tested where GPT-4.1 handled classification tasks while GPT-5 focused only on the final answer generation. This slightly improved response speed, but the overall consistency and quality of the responses were still better when GPT-5 managed the complete reasoning pipeline.

As a result, GPT-5 remained the primary model used in the final implementation because the quality improvements were considered more important than the small speed gains achieved through the hybrid approach.

Even though the final system already performed well, response latency remained one of the identified areas for future optimization.

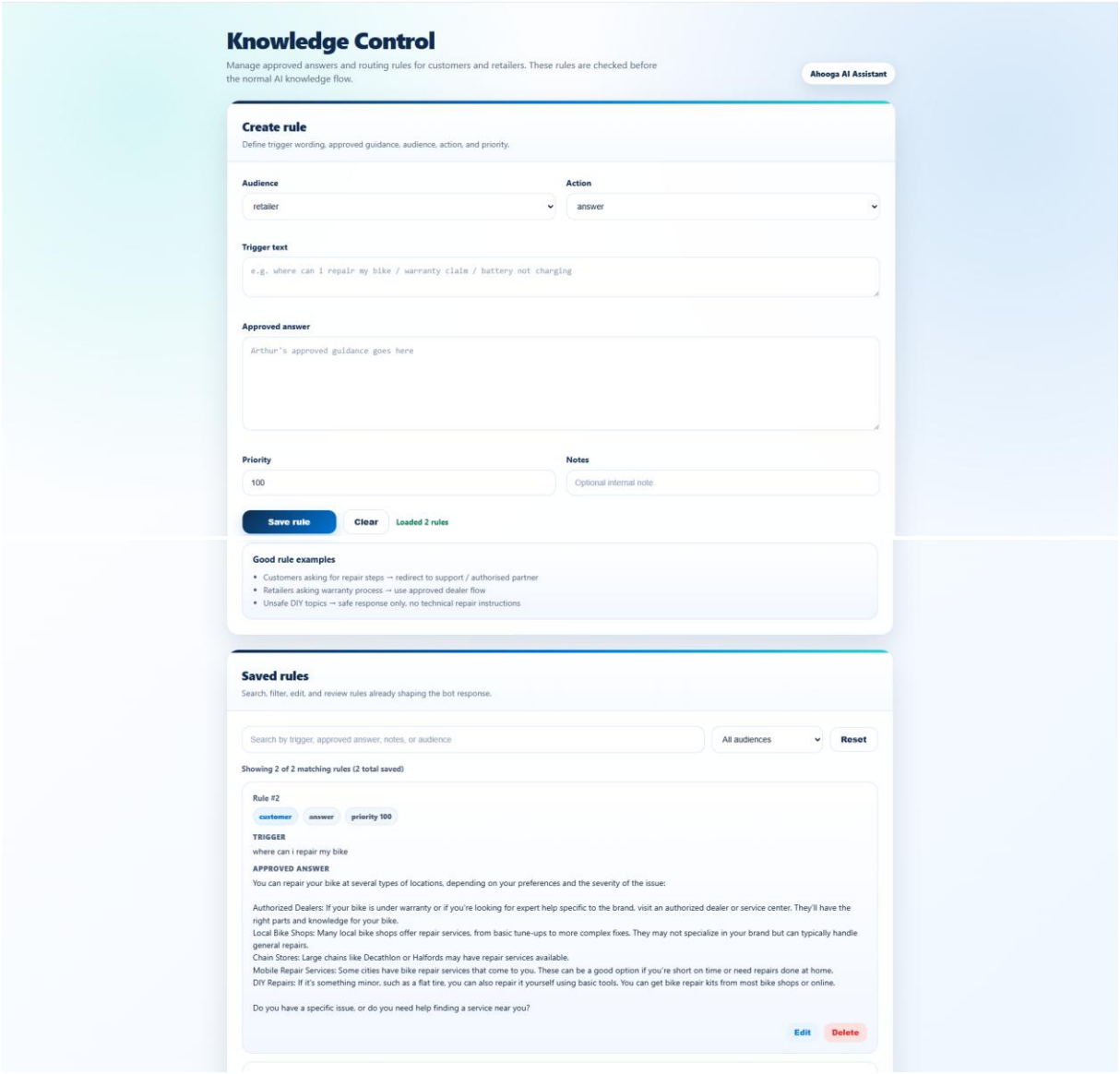
Arthur Rule and Feedback Layer
In addition to retrieval and generation, the system also included an internal control layer called the Arthur interface.

This layer allowed the after-sales team to:
- create approved answers
- define custom rules
- prioritize specific responses
- adjust assistant behavior without modifying code

When a matching rule existed, the backend could prioritize the predefined answer before relying entirely on retrieval generation. This was especially useful for:
- recurring support questions
- warranty procedures
- safety-related instructions
- operational guidelines

The Arthur layer improved consistency and gave the after-sales team more operational control over the assistant.

Figure 23. Internal after-sales control interface used for rule management and feedback adjustment.



This feedback-oriented approach also made the system easier to maintain after the internship because the company could continue improving responses without directly modifying the backend implementation.

Example Retrieval Workflow

The following example summarizes how the workflow operates during a typical interaction.

Example question:
“The MAX motor stops during acceleration.”
The workflow then proceeds as follows:

1. The backend identifies references related to an electric bike configuration
2. The assistant infers that the question relates to the E-MAX platform

3. Drivetrain and supplier-related context are evaluated
4. Metadata filters are generated dynamically
5. Azure AI Search retrieves relevant troubleshooting chunks
6. Retrieved chunks are ranked and filtered
7. GPT-5 generates the final response based on the selected context
8. The response is returned to the dealer interface

This workflow combines retrieval, metadata filtering, contextual reasoning, and prompt orchestration to produce more reliable and technically relevant after-sales support responses.

4.5 Odoo Ticket Processing and Validation

Odoo Ticket Processing and Validation
In addition to manuals, troubleshooting guides, and processed video content, Odoo support tickets were also explored as an important knowledge source for the AI assistant. These tickets contained real after-sales discussions between dealers and the support team, including technical problems, repair procedures, troubleshooting attempts, and final solutions.

Because many after-sales questions are repetitive, the ticket history represented valuable practical knowledge that could improve the assistant's ability to handle real support situations. However, unlike structured documentation, support tickets are often inconsistent and difficult to use directly. Some tickets contain incomplete information, duplicated replies, unrelated discussions, or personal customer information that should not be exposed through an AI system.

For this reason, a separate ticket processing and validation pipeline was developed before considering integration into the knowledge base.

The first step of the pipeline focused on filtering the Odoo tickets to retrieve only solved dealer-related cases. This reduced the amount of irrelevant or unfinished ticket data.

The following filtering logic was used during ticket retrieval:

```
domain = [
    ["stage_id.name", "in", ["Solved", "Closed"]],
    ["team_id.name", "=", "Dealers"],
]
```

The ticket pipeline followed the same retrieval-oriented philosophy used throughout the project. Instead of training the assistant directly on all ticket data, the goal was to retrieve and summarize only the most relevant validated cases when needed.

To keep the process manageable and easier to validate, the ticket integration started as a limited pilot dataset using a smaller set of technically relevant dealer tickets from recent years. The tickets were retrieved through the internal API layer connected to Odoo.

After retrieval, the ticket content passed through multiple cleaning and preprocessing steps. HTML formatting, signatures, duplicated replies, and unnecessary system-generated text were removed. The system then attempted to identify useful technical information such as:

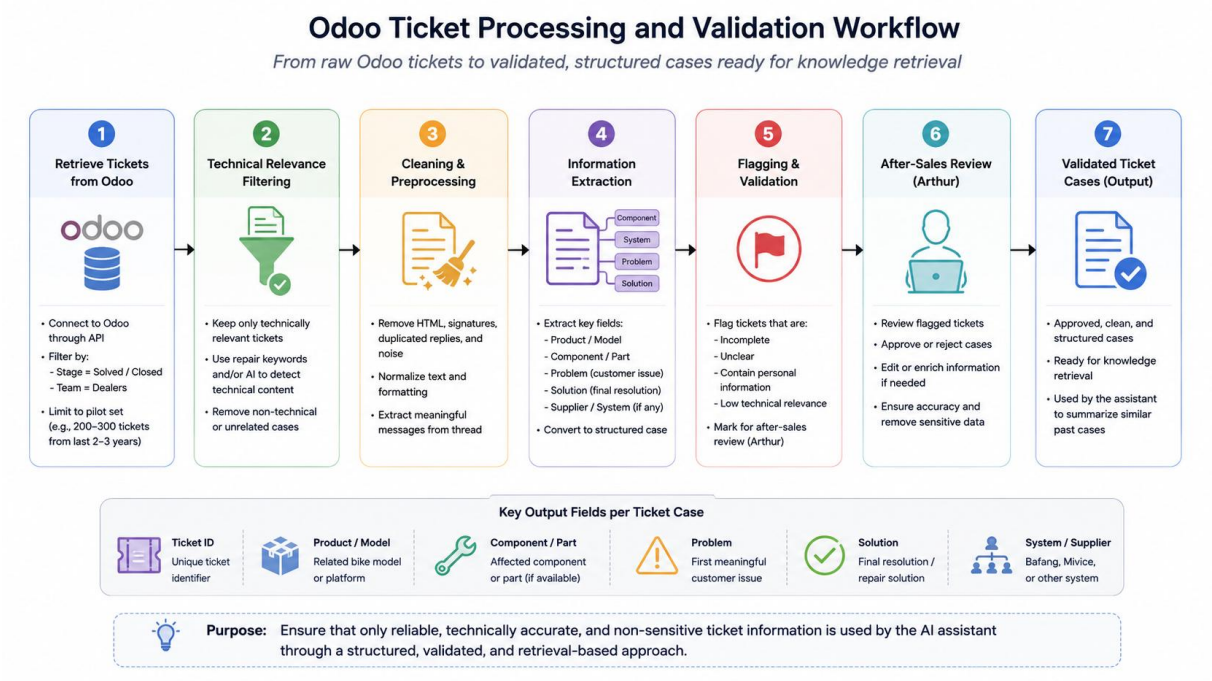
- affected component
- supplier platform
- technical problem
- final solution
- related bike configuration

The extraction pipeline converted the ticket conversations into cleaner structured cases.

A simplified example of the extracted structure is shown below:

```
return {
    "ticket_id": ticket["id"],
    "system": self.infer_system(combined_text),
    "component": self.infer_component(combined_text),
    "problem": first_meaningful or None,
    "solution": last_meaningful or None,
    "status": status,
}
```

***Figure 24**. Odoo ticket processing and validation workflow used during preprocessing.*



In addition to cleaning and structuring, a validation and flagging mechanism was also implemented. Even when tickets were technically cleaned correctly, some cases still required human review before being added into the production knowledge base.

Examples of tickets that could be flagged included:

- tickets containing personal customer information
- unclear troubleshooting discussions
- incomplete solutions
- inconsistent repair outcomes
- weak technical relevance

This validation step was considered important because support tickets can contain sensitive or ambiguous information that should not automatically become part of the AI retrieval system.

The pipeline therefore separated technical preprocessing from final business validation. The after-sales team could review flagged tickets and decide whether the case was reliable enough to be used as reusable support knowledge.

This approach improved both reliability and operational safety while reducing the risk of exposing incorrect or sensitive information through the assistant.

Although the final production integration of the tickets was not fully completed during the internship period, the ticket processing pipeline successfully demonstrated that Odoo support data could be transformed into structured and reusable technical knowledge. The

implemented cleaning, filtering, and validation workflow also created a strong foundation for future expansion of the assistant's knowledge base.

4.6 Arthur Interface and Feedback Management

In addition to the retrieval and validation systems, an internal Arthur interface was also developed during the project. The purpose of this interface was to allow the after-sales team to manage, improve, and control specific assistant responses without directly modifying the backend code or retrieval system.

Although the retrieval-based assistant was already capable of generating strong technical answers from the knowledge base, some recurring support situations required more controlled and standardized responses. In real after-sales operations, certain answers must remain consistent, especially when dealing with troubleshooting instructions, repair procedures, or approved company communication. Because of this, a rule-based feedback and override system was introduced.

The Arthur interface allowed authorized internal users to create predefined rules and approved responses that could be prioritized before the retrieval pipeline was executed. This ensured that validated company answers could be returned immediately when a matching situation was detected.

*The Arthur interface was previously presented in **Figure 7** (System Architecture) and **Figure 23** (Request Processing Workflow). The following section explains how the interface is used internally to manage validated responses, rules, and feedback within the assistant.*

The interface was connected to the backend through the centralized API layer and communicated directly with the chatbot services. The stored rules were managed separately from the document retrieval system, allowing the after-sales team to adjust responses without modifying the ingestion pipeline or Azure AI Search index.

Several management features were included in the interface, such as:

- creating new response rules
- editing existing rules
- deleting rules
- defining the intended audience
- managing approved response content

The audience logic was an important part of the system design. Even though the deployed version of the assistant mainly focused on dealer support, the interface still included audience options for both dealers and customers. This decision was made to support future scalability and allow the system to later expand into customer-facing use cases without requiring major architectural changes.

The rule system was also designed to work together with the contextual detection pipeline described earlier in the workflow section. When a user submitted a question, the backend first attempted to determine the product platform, supplier system, component context, and intended audience. The system then checked whether a matching approved rule already existed before continuing to the retrieval stage.

A simplified example of the rule-check logic is shown below:

```
if matching_rule:
    return approved_response
```

If a valid rule was found, the approved response was returned directly. Otherwise, the normal retrieval and generation workflow continued through Azure AI Search and Azure OpenAI.

This hybrid approach combined the flexibility of AI retrieval with the reliability of manually validated company knowledge. It also reduced the need for unnecessary retrieval operations in highly repetitive support situations.

Another important advantage of the Arthur interface was that it allowed the assistant to improve continuously over time. After-sales employees could monitor recurring questions, identify weak responses, and create improved validated answers directly through the interface. This created a practical feedback loop between real support usage and system improvement.

The Arthur interface therefore played an important role in making the assistant more maintainable, scalable, and operationally reliable for long-term company use.

4.7 Internal API Layer and System Integration

During the later stages of the internship, additional discussions with Hendrik led to the development of a centralized internal API layer alongside the chatbot system. While this was not part of the original project scope, the idea emerged from the need to create a more reusable and scalable integration structure that could support not only the assistant itself, but also future internal company use cases.

As the project evolved, multiple systems started interacting with the same data sources and backend services. Because of this, directly connecting every component to Odoo or individual services would have made the system harder to maintain and expand over time. The API layer was therefore introduced as a centralized communication layer between the different parts of the project.

The API layer was developed using FastAPI and followed a modular backend structure where routes, services, and processing logic were separated into different components. This helped keep the backend organized and made future maintenance easier.

Several systems were already connected to this API layer during the internship, including:

- the chatbot backend
- the Arthur interface
- the Odoo helpdesk ticket processing services
- Odoo field extraction services

One important part of this implementation was the extraction of structured Odoo fields. Dedicated endpoints were created to expose useful business and operational data from Odoo in a cleaner and more reusable format. Initially, this was mainly developed to simplify internal development and data access during the project. However, it also opened additional possibilities for future company use cases.

Because the API layer exposes structured data through reusable endpoints, future teams could potentially build additional tools or integrations on top of the same infrastructure.

For example, the extracted Odoo fields could later support:

- reporting tools
- analytics dashboards
- BI integrations
- operational monitoring systems
- future AI assistants connected to company business data

This means that future systems could retrieve structured information through the API layer instead of directly interacting with the raw Odoo environment.

Another advantage of the centralized API structure was that it reduced direct dependency between systems. Instead of tightly coupling every component to Odoo or individual backend services, the API layer acted as an intermediary communication layer that simplified integration management and improved scalability.

The backend structure used for the API layer was previously presented in the knowledge base and data processing section. The implementation mainly focused on creating a maintainable and reusable backend structure that the company could continue building on after the internship period.

Although the API layer was initially developed to support the AI assistant project, it also created a strong technical foundation for future internal integrations and system expansion within the company.

4.8 Testing, Evaluation and Improvements

Throughout the internship, the system went through multiple testing and improvement phases before reaching its final state. Since the project was based on AI retrieval and response generation, achieving reliable and consistent answers required continuous experimentation, prompt adjustments, retrieval improvements, and feedback from the after-sales team.

The first working versions of the assistant already demonstrated the potential of the retrieval-based approach, but several limitations quickly appeared during testing. Early responses were sometimes inconsistent, retrieval quality was not always reliable, and the assistant occasionally mixed incompatible bike configurations or generated responses that were too generic for technical after-sales usage.

Because of this, a large part of the internship focused on iterative improvement rather than simply adding more features.

One of the most important improvement areas was prompt engineering. The prompts were continuously refined and restructured to improve:

- technical accuracy
- response consistency
- dealer-focused behavior
- troubleshooting quality
- contextual understanding
- response formatting

Instead of relying heavily on hardcoded backend logic, the project mainly used prompt structure, metadata filtering, and retrieval context to guide the assistant behavior. This made prompt optimization an important part of the development process.

Several internal discussions and feedback sessions also played a major role during testing. Meetings with Arthur, Hendrik, and Steve were used to review the chatbot responses, compare answer quality, identify weak areas, and decide on further improvements. Realistic dealer-like questions and troubleshooting scenarios were repeatedly tested to evaluate how the assistant behaved in practical situations.

One major challenge during development was balancing answer quality with response

speed. The final implementation mainly used GPT-5 because it produced the best technical responses and handled contextual reasoning more reliably than the other tested models.

However, GPT-5 also introduced higher response latency during retrieval and answer generation. To improve speed, another configuration was tested where GPT-4.1 handled the classification tasks while GPT-5 generated the final answer. Although this slightly improved performance speed, the overall response quality and consistency were still better when GPT-5 managed the complete reasoning process.

For this reason, GPT-5 remained the primary model used in the final implementation, even though response speed was still identified as one of the areas that could be further optimized in the future.

Another important improvement area was retrieval quality. As the project evolved, the metadata structure, contextual filtering, and retrieval logic were gradually refined to reduce incorrect matches and improve troubleshooting relevance.

The separation between:

- A-MAX and E-MAX
- Bafang and Mivice systems
- Drivetrain/Gear system configurations

significantly improved retrieval precision and reduced the risk of incompatible troubleshooting responses.

The ranking and filtering logic also improved over time. Instead of retrieving only keyword-based matches, the system gradually evolved into a more context-aware retrieval workflow that considered:

- supplier compatibility
- component relevance
- drivetrain context
- audience type
- semantic similarity

This greatly improved the quality of the generated responses.

The knowledge base itself also went through several re-ingestion and restructuring phases during development. Documents, processed video summaries, and metadata structures were repeatedly reorganized and refined to improve retrieval consistency. The naming conventions and metadata organization became an important part of maintaining reliable retrieval behavior across the system.

The processed Odoo ticket pipeline was also tested successfully, although the final production integration of the tickets into the knowledge base was intentionally postponed.

Even though the cleaning and extraction process worked correctly, additional validation by the after-sales team was considered necessary before allowing ticket information to become part of the production retrieval pipeline.

By the end of the internship, the assistant had evolved into a stable dealer-focused after-sales support system for the MAX platform. The chatbot was capable of handling a wide range of troubleshooting and technical support questions using retrieval-based reasoning combined with structured company knowledge.

Although some future improvements were still identified, the final system already demonstrated strong practical value for the company and created a solid foundation for future expansion and continuation after the internship period.

Several limitations still remained at the end of the project. These included:

- response latency caused by large model usage
- incomplete ticket validation integration
- support currently focused mainly on the MAX platform
- dependency on knowledge base quality
- OCR limitations for certain video frames and visual materials

Despite these limitations, the overall project successfully demonstrated how retrieval-based AI systems can be applied to improve technical after-sales support workflows in a real company environment.

4.9 Future Work and Project Continuation

Although the assistant reached a functional and stable state during the internship, several opportunities for future development were identified.

One of the main next steps is the continuation of the Odoo ticket integration process. The ticket extraction, cleaning, and structuring workflow was completed during the internship, but additional validation by the after-sales team is still required before the ticket data can be safely integrated into the production knowledge base. The established workflow provides a solid foundation for continuing this process after the internship.

Another area for future development is the expansion of the assistant beyond the MAX platform. The current implementation focuses primarily on MAX dealer support, but the same architecture can be extended to support additional bike platforms, suppliers, and product lines used by the company.

Response performance also remains an area for improvement. During testing, GPT-5 produced the highest quality responses, but at the cost of increased response time. Future work could investigate alternative model combinations, caching strategies, or retrieval optimizations to improve responsiveness while maintaining answer quality.

The internal API layer developed during the project also creates opportunities beyond the chatbot itself. Since Odoo data and operational information are now exposed through reusable endpoints, the same infrastructure could be reused for reporting dashboards, business intelligence applications, automation workflows, or future AI-powered tools within the company.

Overall, the project delivered not only a working assistant but also a technical foundation that can continue to evolve and support future digital initiatives at Ahooga.

5. Conclusion

This internship project focused on the development of an AI-based after-sales assistant for Ahooga. The main objective of the project was to improve how technical support knowledge is accessed and used within the company, while reducing the repetitive workload faced by the after-sales team.

Throughout the internship, the project evolved from an initial research and analysis phase into a working retrieval-based AI assistant focused mainly on dealer support for the MAX bike platform. The final system combined Azure AI Search, Azure OpenAI, structured metadata filtering, processed documentation, video summaries, and contextual retrieval logic to generate more reliable technical support responses.

Several important functionalities were successfully implemented during the project. These included:

- a retrieval-based chatbot backend
- structured document and video ingestion
- metadata-driven retrieval filtering
- contextual response generation using GPT-5
- the Arthur rule and feedback layer
- Odoo ticket preprocessing and validation
- and a centralized internal API layer for system integration

One of the most important aspects of the project was the focus on reliability and operational control. Instead of creating a fully open-ended chatbot, the system was designed to generate

responses based on validated company knowledge and structured retrieval. This helped improve consistency and reduce the risk of incorrect technical guidance.

The project also demonstrated the importance of data organization and retrieval quality in AI systems. A large part of the work involved structuring documents, improving metadata, refining prompts, testing retrieval behavior, and continuously adjusting the workflow based on feedback and real troubleshooting scenarios.

Although some future improvements were still identified at the end of the internship, the final system already demonstrated strong practical value for the company. The chatbot was capable of handling technical dealer-focused questions in a much more structured and efficient way compared to manually searching through documentation.

The internship also highlighted several real-world engineering challenges. During development, multiple changes were made to the project scope and system design based on feedback, testing, and operational needs inside the company. This required adaptability, continuous planning, and the ability to work under pressure while managing different technical priorities within a limited internship period.

From a personal perspective, the internship provided valuable experience not only in AI development, but also in system design, problem solving, backend architecture, and working within a real company environment. It also improved skills related to time management, communication, iterative development, and handling evolving project requirements.

Overall, the project created a strong technical foundation that Ahooga can continue building on in the future. The implemented architecture, retrieval pipeline, ticket processing workflow, and centralized API layer provide opportunities for future expansion, additional integrations, and further development of AI-supported after-sales operations within the company.

Reference list

1. Microsoft. *Azure AI Search Documentation*. Available at: https://learn.microsoft.com/en-us/azure/search/ Accessed: 2026.
2. Microsoft. *Azure OpenAI Service Documentation*. Available at: https://learn.microsoft.com/en-us/azure/ai-services/openai/ Accessed: 2026.
3. Microsoft. *Azure Blob Storage Documentation*. Available at: https://learn.microsoft.com/en-us/azure/storage/blobs/ Accessed: 2026.
4. OpenAI. *OpenAI API Documentation*. Available at: https://platform.openai.com/docs/ Accessed: 2026.
5. FastAPI. *FastAPI Documentation*. Available at: https://fastapi.tiangolo.com/ Accessed: 2026.
6. Google. *Tesseract OCR Documentation*. Available at: https://tesseract-ocr.github.io/ Accessed: 2026.
7. Odoo. *Odoo Documentation*. Available at: https://www.odoo.com/documentation/ Accessed: 2026.
8. Microsoft. *Azure AI Foundry Documentation*. Available at: https://learn.microsoft.com/en-us/azure/ai-foundry/ Accessed: 2026.
9. Microsoft. *Azure AI Search Vector Search Documentation*. Available at: https://learn.microsoft.com/en-us/azure/search/vector-search-overview Accessed: 2026.

Use of Generative AI

Generative AI tools were used throughout different stages of this internship project to support development, research, debugging, documentation, and content structuring. The main AI tools used during the project included ChatGPT within the Ahooga company environment and Azure OpenAI models integrated within the implemented system itself.
AI assistance was mainly used for:

- brainstorming and exploring implementation approaches
- debugging and improving backend logic
- refining prompt structures
- improving documentation wording and structure
- assisting with technical explanations
- testing retrieval and response behavior
- generating and refining development ideas during experimentation phases

During the implementation phase, generative AI was also used as part of the actual project solution through the integration of GPT-based models for classification, retrieval reasoning, and response generation inside the after-sales assistant.
The use of AI during the report writing process mainly focused on language refinement, structuring support, and improving readability. All generated content, explanations, and outputs were manually reviewed, verified, adjusted, and adapted to reflect the actual implementation and internship work carried out during the project.
The technical implementation, architectural decisions, integrations, testing processes, and development work described in this document were based on the practical work completed during the internship at Ahooga.
Examples of prompts used during development and documentation included:

- “How can metadata filtering improve retrieval accuracy in a RAG system?”
- “Suggest a cleaner prompt structure for dealer-focused troubleshooting responses.”
- “Explain the difference between retrieval-based AI and fine-tuning.”
- “Improve the clarity and readability of this implementation explanation.”
- “Suggest ways to structure processed support ticket data for AI retrieval.”

The outputs generated through AI tools were never used without review or validation. Generated suggestions and explanations were critically evaluated and adjusted where necessary to ensure technical correctness, consistency, and alignment with the actual project implementation.

ATTACHMENTS

Appendix A – Example Prompt Structures

The following examples show simplified versions of prompts used during development for classification, retrieval guidance, and response generation inside the assistant.

Example Classification Prompt

Determine whether the question relates to:
- A-MAX or E-MAX
- Bafang or Mivice
- Nexus, Alfine, or Derailleur

Return only the detected configuration and technical context.

Example Retrieval Guidance Prompt

Use only the retrieved company knowledge to answer the question.
Do not generate unsupported troubleshooting steps.
Prioritize dealer-focused technical instructions.

The prompts were continuously adjusted during development to improve response consistency, retrieval grounding, and technical reliability.

Appendix B – Example Odoo Ticket Processing Output

The following simplified example shows how support ticket data was converted into a structured format during preprocessing.

***Figure A1**. Example structured Odoo ticket preprocessing output after cleaning and validation.*


```
{
  "ticket_id": 63200,
  "ticket_name": "Extra Battery MAX Ahooga bike",
  "flag": "low_confidence",
  "flag_reason": "Resolution unclear or very short",
  "product_model": null,
  "system": "Bafang",
  "component": "battery",
  "confidence": "low",
  "problem": "Inquiry about an extra battery for a MAX Ahooga bike.",
  "resolution": "Provided guidance to use the B2R portal and Bafang Go+ app for updates and offered further assistance if needed.",
  "original_content": "Extra Battery MAX Ahooga bike\n\n\n\nHello, you can access to the B2R portal and find the possible updates that you can do
},
{
  "ticket_id": 63202,
  "ticket_name": "MAX",
  "flag": "low_confidence",
  "flag_reason": "Resolution unclear or very short",
  "product_model": "MAX",
  "system": "Bafang",
  "component": "display",
  "confidence": "low",
  "problem": "A part on the MAX bike has broken off and needs replacement/repair.",
  "resolution": "Quotation S16948 (€59.40) was issued; customer provided contact details and paid. Parts will be shipped (to BR Roselaere) for a
  "original_content": "MAX\n\n\n\n\n[GREETING],\n Thank you for reaching out to our customer service department. To book an appointment, simply use
},
```


The preprocessing pipeline cleaned the raw ticket discussions and extracted reusable technical information before validation.

Appendix C – Example API Endpoint Structure

The following example shows a simplified backend endpoint used during ticket retrieval.

```
@router.get("/solved-dealers")
def get_solved_dealer_tickets(limit: int = 20):
    return ticket_service.get_solved_dealer_tickets(limit=limit)
```

The API layer centralized communication between the chatbot backend, Arthur interface, Odoo integrations, and ticket processing services.

***Figure A2**. Example API endpoint testing interface used during ticket retrieval validation.*