POLITECNICO DI TORINO

Master degree course in Digital Skills for Sustainable Societal Transitions

Master Degree Thesis

A Cloud-Based Smart Water Monitoring System with Integrated Conversational AI: Implementation and Case Study in the Piedmont Region

Relatori
Urgese Gianvito
Fanuli Giuseppe
Gallego Gomez Walter

Candidato
Shobeyri Sayed Emadedin

Supervisore Aziendale
Becchis Elio (Fondazione DIG421)

July 2025

Abstract

This thesis presents the design and implementation of an intelligent cloud-based infrastructure for smart water distribution monitoring, featuring a conversational AI interface that simplifies operator interaction with complex systems. It addresses limitations in current monitoring setups, where traditional interfaces demand technical expertise and reduce usability.

The solution leverages Google Cloud Platform services to build a monitoring ecosystem that integrates real-time sensor data processing, hydraulic simulation, machine learning-based leakage detection, and geospatial mapping. Cloud-native principles like microservices, serverless computing, and event-driven processing ensure scalability, reliability, and cost-efficiency. A key innovation is a conversational AI system built with Large Language Models and LangChain, enabling natural language interaction with monitoring dashboards. A Grafana-based graphical interface provides intuitive visualization of real-time and historical data, geospatial mapping, and interactive controls.

The system includes automated data ingestion via Cloud Functions, BigQuery for data warehousing, Grafana dashboards, EPANET for hydraulic simulation, and a React-based frontend. PostgreSQL with PostGIS manages geospatial data, enabling efficient spatial queries and integration with mapping features.

Development involved prototyping with QGIS Web Client and Google Looker Studio, with Grafana ultimately chosen for unified visualization. The AI assistant evolved from a CLI tool to a hybrid architecture where LLMs extract user intent and Python functions execute actions. Key milestones include minimizing LLM usage for efficiency, integrating the Grafana API with WebSocket support, and implementing advanced features like natural language queries and geospatial navigation.

The system was developed and validated in collaboration with AlpiAcque and Tesisquare, focusing on the Marene water network in Cuneo, Piedmont, Italy. Districts like Ponte, Via Torino, San Bernardo, Salza, and Concentrico are equipped with sensors providing 15-minute interval data for analysis. Historical sensor and leakage data were used to validate machine learning models. The IDEA SCADA system, managed by IDEA Srl, was integrated for data collection, and enhanced

with anomaly detection, predictive analysis, and district-level geospatial visualization.

This research contributes to infrastructure monitoring, cloud architecture, conversational AI, and HCI design. Key outcomes include integration models for AI-driven industrial interfaces, transparent query workflows, resilient cloud patterns, and evaluation methods for AI-enhanced monitoring. The work shows how conversational AI can improve performance and accessibility in technical environments.

Acknowledgments

I am deeply grateful to my thesis supervisor, Professor Gianvito Urgese, for his invaluable guidance, encouragement, and expertise throughout this journey. His insightful feedback and profound knowledge have been fundamental to the success of this work. His support has not only shaped my academic path but also inspired me to explore new horizons in cloud computing and smart infrastructure.

I would like to thank Politecnico di Torino for providing an inspiring academic environment and the resources that made this research possible. The technical infrastructure and learning opportunities offered by the university were essential to the development of this thesis.

My sincere appreciation goes to TesiSquare and DIG421 for the opportunity to conduct my internship and thesis work within the company, which enriched my academic experience with a practical industry perspective.

Special thanks to my colleagues, Walter Gallego Gomez and Giuseppe Fanuli, for their technical discussions, support, and friendship. Their collaborative spirit and invaluable insights helped refine many aspects of this work. I am also thankful to all my friends and dear housemates for their encouragement and companionship throughout this journey.

I owe my deepest gratitude to my family for their unwavering support and patience throughout my life. A special thanks to my mother, whose endless love, sacrifices, and strength have been the foundation of all my achievements.

I am especially grateful to my special someone and the love of my life, whose unwavering support, kindness, and love have been a constant source of strength. Her encouragement and belief in me have meant more than words can express.

Finally, I dedicate this thesis to the memory of my father. His wisdom, values, and belief in education continue to inspire me every day. I hope this achievement would have made him proud.

Contents

List of Figures

Chapter 1

Introduction

Smart water distribution monitoring systems face significant challenges in translating advanced sensor data and analytical capabilities into accessible, actionable insights for operational personnel. While technologies such as cloud computing, *IoT* sensors, and data analytics offer powerful capabilities for infrastructure monitoring, the complexity of existing interfaces often limits their practical effectiveness. This thesis addresses these challenges through the development of an integrated cloud-based infrastructure for smart water monitoring, featuring automated data processing pipelines, comprehensive error detection and recovery mechanisms, and an innovative conversational *AI* interface that enables natural language interaction with Grafana dashboards [1] and analytical tools.

The implemented system leverages Google Cloud Platform's [2] native services to create a robust, scalable architecture with automated monitoring, intelligent resource scaling, and comprehensive system observability. The cloud-native approach ensures reliable operation through automated error detection, recovery mechanisms, and sophisticated data validation processes that maintain system integrity while enabling seamless scalability based on operational demands.

1.1 Problem Statement

Current water monitoring systems typically suffer from fragmented architectures that require operators to interact with multiple disconnected tools and interfaces. *SCADA* systems provide real-time operational control but often lack sophisticated analytical capabilities. Geographic information systems (*GIS*) provide spatial context but may not integrate effectively with real-time sensor data. Grafana dashboards offer powerful visualization capabilities but require technical expertise to configure and modify effectively. This fragmentation results in inefficient workflows and reduced effectiveness of analytical capabilities.

The complexity of existing monitoring interfaces presents significant barriers to

effective utilization. Traditional dashboard platforms require extensive technical knowledge to add panels, modify visualizations, construct analytical queries, or access historical data. This complexity limits accessibility to technical specialists while preventing broader operational personnel from effectively utilizing available information resources or adapting visualizations to changing operational needs.

The static nature of many monitoring visualizations limits their effectiveness for dynamic operational scenarios. Dashboard configurations often cannot be easily modified during emergency response situations or analytical investigations, requiring manual intervention by technical personnel to reconfigure views for different operational contexts.

1.2 Research Objectives

This research aims to address the identified challenges through the development of an integrated, cloud-based infrastructure that combines advanced data processing capabilities with a novel conversational *AI* interface to create a comprehensive monitoring solution that makes sophisticated dashboard management accessible through natural language interaction.

The primary objective is to design, implement, and evaluate a complete conversational interface for Grafana dashboard management that enables operators to perform complex visualization tasks through natural language commands. This system provides comprehensive dashboard manipulation capabilities including panel creation and modification, time range adjustments, variable management, data querying with transparent *SQL* display, and geospatial operations with automatic district zoom functionality while maintaining operational safety through built-in confirmation workflows and undo capabilities that leverage Grafana's version history.

A critical component involves developing innovative natural language processing capabilities using Google's Gemini 2.0 Flash *LLM* [3] integrated with LangChain [4] and LangGraph [5] frameworks. The research investigates how conversational interfaces can effectively bridge the gap between complex dashboard management requirements and operational accessibility, including natural language command interpretation, automated dashboard configuration, intelligent parameter extraction, and transparent *SQL* query generation for data analysis.

Technical objectives include developing scalable, cloud-native architectures based on FastAPI [6] and WebSocket communication [7] that can process natural language inputs in real-time while maintaining secure integration with Grafana *HTTP API*, Google Cloud BigQuery [8] for data analysis, and geospatial services for district-based operations. The system implements comprehensive state management using LangGraph workflows that support complex multi-step operations with appropriate error handling and user confirmation requirements.

1.3 Research Approach

The research methodology follows a systematic engineering design approach combining theoretical investigation with practical implementation and comprehensive evaluation. The approach is based on design science research principles emphasizing the creation and evaluation of innovative artifacts addressing practical problems.

This research builds upon extensive preliminary investigation and prototyping work conducted during a focused internship period. The initial exploration examined various visualization platforms including *GIS* integration approaches and business intelligence tools, which provided valuable insights into user accessibility challenges and workflow inefficiencies. These early findings motivated the focus on conversational *AI* interfaces as a means to bridge the gap between complex technical capabilities and operational usability.

The evolution from traditional *GIS*-based interfaces to conversational *AI* represents a fundamental shift in approach. Development progressed through iterative phases starting with backend functionality validation and evolving toward comprehensive web-based systems with natural language capabilities. Critical architectural insights emerged regarding the optimal integration of *LLMs* for natural language understanding while utilizing dedicated functions for precise technical operations, ensuring both reliability and cost-effectiveness.

The final system architecture leverages Google's Gemini 2.0 Flash model through LangChain and LangGraph frameworks to create a sophisticated conversational interface that interprets natural language commands, manages complex state transitions, and coordinates operations across multiple system components including Grafana dashboards, BigQuery data analysis, and geospatial services.

While this thesis addresses machine learning integration within the monitoring system, the core machine learning algorithms for anomaly detection were developed by a research colleague. This thesis focuses primarily on the novel conversational *AI* interface implementation, Grafana *API* integration, cloud infrastructure development, and the comprehensive workflow orchestration system that makes complex dashboard operations accessible through natural language interaction.

1.4 Thesis Structure and Contributions

This thesis is organized to provide comprehensive coverage of the research problem, solution development process, implementation details, and evaluation results.

Chapter 2 provides examination of the technological foundations directly relevant to the conversational *AI* dashboard interface, including Google Cloud Platform services, LangChain and LangGraph frameworks for *AI* application development, Grafana *HTTP API* integration approaches, FastAPI and WebSocket technologies for real-time communication, React [9] based frontend development with Vite build

system, and BigQuery integration for data analysis capabilities.

Chapter 3 presents a comprehensive exploration of the development journey, architectural design, and technical implementation of the intelligent water monitoring system. The chapter begins with the methodological approach that guided the research evolution from preliminary QGIS-based implementations to an integrated Grafana-based solution, highlighting critical technology decisions and lessons learned. It then details the systematic architectural design principles, including the cloud-native infrastructure, modular component integration, and specialized interfaces. The core implementation section covers the complete technical stack including data acquisition pipelines, hydraulic simulation integration, machine learning algorithm incorporation, geospatial data management, advanced visualization systems, and most significantly, the innovative conversational AI interface.

Chapter 4 presents comprehensive evaluation results, focusing on the system's capabilities for real-time monitoring, geospatial visualization, and conversational AI-driven dashboard management. The chapter begins with a detailed case study of the Marene water distribution network, highlighting collaboration with Alpi-Acque and Tesisquare, and the integration of SCADA systems, GIS platforms, and historical leakage data. It then evaluates the system's ability to modify dashboard panels, manage time ranges, and execute natural language queries for data analysis, including districts with highest and lowest leakage probabilities. The chapter also demonstrates the operational utility of geospatial navigation and safety mechanisms like undo functionality, emphasizing practical applications in municipal water management.

Chapter 5 synthesizes the research contributions, discussing the methodological value of real-world validation and industry collaboration. It provides a critical analysis of the system's effectiveness in integrating cloud infrastructure, data processing, visualization, and conversational AI components. The chapter concludes with reflections on the limitations of the current implementation and outlines future directions for enhancing AI-driven monitoring systems, including scalability improvements, expanded geospatial capabilities, and broader industry adoption.

The primary contributions of this thesis include the development and validation of large language model technology for industrial Grafana dashboard management through natural language commands. Technical contributions include innovative LangGraph workflow patterns for complex multi-step operations, comprehensive parameter extraction and validation mechanisms, transparent *SQL* query generation with user visibility, and robust error handling with confirmation workflows. The system demonstrates significant improvements in dashboard management accessibility while maintaining operational safety through built-in confirmation mechanisms and undo functionality.

Chapter 2

Technological Foundations and Related Work

This chapter provides a comprehensive examination of the technological foundations and related work that underpin the development of the intelligent water distribution monitoring system presented in this thesis. The review focuses specifically on the technologies and methodologies that have been integrated to create the proposed solution, progressing from cloud-based data infrastructure through geospatial visualization technologies, conversational artificial intelligence frameworks, and specialized water distribution monitoring approaches. The analysis establishes the theoretical and technical foundations for the innovative integration of Google Cloud Platform services, advanced visualization tools, and natural language processing capabilities for infrastructure monitoring applications.

2.1 Google Cloud Platform and Data Infrastructure

The development of large-scale monitoring systems for critical infrastructure requires robust, scalable cloud computing platforms that can handle diverse data processing requirements, provide reliable storage solutions, and support sophisticated analytical workloads. Google Cloud Platform (*GCP*) has emerged as a leading cloud infrastructure provider, offering a comprehensive suite of services particularly well-suited for data-intensive applications in infrastructure monitoring [2]. Figure 2.1 illustrates the key GCP services utilized in this infrastructure monitoring implementation.

Figure 2.1: Overview of Google Cloud Platform Services Utilized in The System

2.1.1 Google Cloud BigQuery for Time Series Data Management

Google Cloud BigQuery represents a fundamental component of modern data infrastructure for monitoring applications, providing a fully-managed, serverless data warehouse optimized for analytical workloads [8]. BigQuery's architecture is particularly well-suited for time series data common in infrastructure monitoring applications, offering several key advantages for water distribution system monitoring.

The columnar storage format employed by BigQuery provides significant performance advantages for analytical queries typical in monitoring applications. Time series data, characterized by frequent insertions and analytical queries across temporal ranges, benefits from BigQuery's optimized storage and query execution capabilities [10]. The system's ability to automatically partition data based on ingestion time or custom partitioning schemes enables efficient querying of large temporal datasets while controlling storage costs through automated data lifecycle management.

BigQuery's integration with Google Cloud's broader ecosystem provides seamless connectivity with data processing services, enabling sophisticated data transformation and analysis workflows. The platform's *SQL*-based query interface facilitates accessibility for operational personnel while supporting advanced analytical capabilities through user-defined functions and machine learning integration [2].

The serverless nature of BigQuery eliminates infrastructure management overhead while providing automatic scaling capabilities that accommodate varying workloads typical in monitoring applications [11]. This characteristic proves particularly valuable for water distribution monitoring systems that experience variable data volumes and analytical demands based on operational requirements and system conditions.

2.1.2 Google Cloud Functions for Serverless Data Processing

Google Cloud Functions provides event-driven, serverless computing capabilities essential for implementing responsive data processing workflows in monitoring systems [2]. The functions-as-a-service model enables the development of specialized processing components that execute automatically in response to data availability or system events, eliminating the need for continuously running infrastructure.

Cloud Functions' integration with Google Cloud Storage enables automatic processing of data files as they arrive in designated storage buckets. This capability supports automated data ingestion workflows where sensor data files trigger processing functions upon upload, ensuring timely data transformation and loading into analytical systems [11].

The execution environment provided by Cloud Functions supports multiple programming languages, with Python being particularly well-suited for data processing applications due to its extensive ecosystem of data manipulation and analysis libraries. Functions can leverage libraries such as pandas [12] for data transformation, enabling sophisticated data quality checks, format standardization, and enrichment processes within the serverless execution context.

Event-driven scaling characteristics of Cloud Functions provide cost-effective processing for variable workloads while maintaining responsiveness to data arrival patterns. Functions scale automatically from zero to handle processing demands, with costs incurred only during actual execution periods, making this approach particularly suitable for monitoring applications with irregular data patterns.

2.1.3 Google Cloud Compute Engine for Specialized Workloads

While serverless computing provides efficient solutions for many data processing tasks, certain specialized workloads in infrastructure monitoring require dedicated computing resources. Google Cloud Compute Engine provides virtual machine instances that can be configured for specific application requirements, offering flexibility for deploying specialized software components.

Compute Engine's support for custom machine configurations enables optimization for specific workload characteristics. Applications requiring substantial memory for spatial data processing or specialized software installations benefit from the ability to configure virtual machines with appropriate resource allocations [13].

The integration of Compute Engine with Google Cloud Scheduler enables cost optimization through automated instance lifecycle management. Virtual machines can be programmed to start automatically for scheduled processing tasks and shut down upon completion, reducing operational costs while maintaining processing capabilities for resource-intensive operations.

Persistent disk storage options provide reliable data storage for applications requiring local file systems or specialized database installations. The separation of compute and storage resources enables flexible scaling approaches while maintaining data persistence across instance lifecycles.

2.1.4 Google Cloud Storage for Data Backup and Archival

Google Cloud Storage provides scalable object storage services essential for comprehensive data management strategies in monitoring systems. The platform's multiple storage classes enable cost-effective data lifecycle management while ensuring appropriate availability and access characteristics for different data types.

Standard storage classes provide high-performance access for frequently accessed data, supporting operational requirements for recent monitoring data and analytical results. Nearline and Coldline storage classes offer cost-effective solutions for archival data while maintaining reasonable access times for historical analysis requirements [14].

Cloud Storage's integration with data processing services enables efficient workflow implementations where raw data storage triggers processing functions while providing reliable backup capabilities for processed results. Object versioning capabilities support data integrity requirements while lifecycle management policies automate storage class transitions based on data age and access patterns.

The global distribution capabilities of Cloud Storage support disaster recovery requirements while enabling efficient data access from multiple geographic locations. This characteristic proves valuable for monitoring systems requiring high availability and geographic distribution of data processing capabilities.

2.1.5 Google Cloud Scheduler for Workflow Orchestration

Google Cloud Scheduler provides cron-based job scheduling capabilities essential for coordinating automated workflows in monitoring systems. The service enables reliable execution of periodic tasks including data processing jobs, system maintenance operations, and resource lifecycle management.

Scheduler's integration with other Google Cloud services enables comprehensive workflow orchestration through HTTP targets, Cloud Functions invocation, and Pub/Sub message publication. This flexibility supports complex workflow patterns while maintaining reliable execution scheduling [15].

The service's built-in retry mechanisms and error handling capabilities ensure robust execution of critical monitoring workflows. Configurable retry policies and dead letter queue support help maintain system reliability even when individual components experience temporary failures.

Integration with Google Cloud Monitoring provides comprehensive observability for scheduled jobs, enabling proactive identification of workflow issues and performance optimization opportunities. This capability supports operational requirements for maintaining reliable monitoring system operations.

2.2 Data Processing and Workflow Orchestration Technologies

Modern infrastructure monitoring systems require sophisticated data processing and workflow orchestration capabilities to handle the complexity and scale of real-time monitoring data. The integration of automated data processing pipelines with robust orchestration frameworks enables reliable, scalable monitoring solutions that can adapt to evolving operational requirements and data characteristics.

2.2.1 Apache Airflow for Workflow Management

Apache Airflow has emerged as a leading open-source platform for developing, scheduling, and monitoring data workflows, providing essential capabilities for managing complex data processing pipelines in infrastructure monitoring applications [16]. Airflow's directed acyclic graph (*DAG*) approach to workflow definition enables clear specification of task dependencies while supporting complex orchestration patterns.

The platform's Python-based workflow definition approach provides significant flexibility for developing custom processing logic while leveraging the extensive Python ecosystem for data manipulation and analysis. Workflows can incorporate data validation, transformation, machine learning, and system integration tasks within unified execution frameworks [16].

Airflow's scheduler component provides reliable execution of workflow tasks based on specified triggers including time-based schedules, external events, and dependency completion. The scheduler's support for parallel execution and resource management enables efficient utilization of available computational resources while maintaining appropriate execution ordering for dependent tasks.

The platform's extensive operator library provides pre-built components for common data processing tasks including database operations, file transfers, *API* interactions, and cloud service integrations. Custom operators can be developed for specialized requirements, enabling seamless integration with domain-specific tools and systems.

Airflow's web-based user interface provides comprehensive monitoring and management capabilities for operational workflows. The interface enables real-time monitoring of task execution, historical analysis of workflow performance, and manual intervention capabilities for troubleshooting and maintenance operations.

2.2.2 Python Ecosystem for Data Processing

Python has established itself as the dominant programming language for data processing applications, offering an extensive ecosystem of libraries and frameworks specifically designed for data manipulation, analysis, and integration tasks. The language's combination of readability, flexibility, and powerful library support makes it particularly well-suited for infrastructure monitoring applications.

The pandas library provides fundamental data structure and manipulation capabilities essential for processing time series data typical in monitoring applications. DataFrames and Series objects enable efficient representation and manipulation of structured data while providing comprehensive functionality for data cleaning, transformation, and aggregation operations [12].

NumPy's array processing capabilities provide the foundation for numerical computing operations required in monitoring data analysis. The library's optimized array operations enable efficient processing of large datasets while supporting mathematical operations essential for statistical analysis and signal processing applications [17].

The datetime module provides essential temporal data handling capabilities for time series processing. Infrastructure monitoring applications require sophisticated time zone handling, temporal arithmetic, and timestamp manipulation capabilities that Python's temporal libraries provide comprehensively.

The requests library enables robust *HTTP* client functionality essential for integrating with web-based *APIs* and services. Monitoring systems frequently require integration with external services, and the requests library provides reliable, feature-rich *HTTP* client capabilities with comprehensive error handling and authentication support.

2.2.3 Automated Data Validation and Quality Assurance

Data quality represents a critical concern in infrastructure monitoring systems, where decision-making depends on accurate, reliable sensor data. Automated data

validation and quality assurance processes ensure data integrity while identifying potential issues that could compromise monitoring effectiveness.

Schema validation techniques verify that incoming data conforms to expected structures and data types, preventing downstream processing errors caused by malformed data. Python's schema validation libraries provide comprehensive capabilities for defining and enforcing data structure requirements while generating appropriate error messages for validation failures.

Statistical outlier detection methods identify unusual data values that may indicate sensor malfunctions or unusual system conditions. Techniques such as interquartile range analysis, z-score calculations, and isolation forest algorithms can automatically flag suspicious data points for further investigation [18].

Temporal consistency checks verify that time series data maintains appropriate temporal ordering and identifies gaps or overlaps in data sequences. These checks ensure that analytical operations depending on temporal continuity can proceed reliably while highlighting potential data collection issues.

Cross-sensor validation techniques compare readings from related sensors to identify inconsistencies that may indicate equipment failures or calibration issues. For example, flow conservation principles can be applied to validate consistency across multiple flow sensors in a water distribution network.

2.2.4 Error Handling and Recovery Mechanisms

Robust error handling and recovery mechanisms are essential for maintaining reliable operation of automated data processing workflows. Infrastructure monitoring systems must continue operating effectively even when individual components experience failures or unexpected conditions.

Retry logic with exponential backoff provides resilience against temporary failures in external services or network connectivity issues. Configurable retry parameters enable appropriate balance between system responsiveness and resource utilization while avoiding overwhelming failing services with repeated requests.

Dead letter queue implementations enable systematic handling of processing failures that cannot be resolved through retry mechanisms. Failed processing tasks can be diverted to specialized queues for manual investigation while preventing workflow disruption for successfully processed data.

Circuit breaker patterns provide protection against cascading failures when external dependencies become unavailable. These patterns enable graceful degradation of system functionality while providing automatic recovery capabilities when dependencies return to normal operation.

Comprehensive logging and monitoring integration enables rapid identification and diagnosis of processing issues. Structured logging with appropriate severity levels and contextual information supports operational troubleshooting while providing data for system performance analysis and optimization.

2.2.5 Performance Optimization and Scalability

Performance optimization represents a continuous concern in data processing workflows, particularly as monitoring systems scale to handle increasing data volumes and complexity. Systematic approaches to performance optimization ensure that processing capabilities can grow with operational requirements.

Parallel processing techniques enable efficient utilization of multi-core processing environments while reducing overall processing times. Python's multiprocessing and concurrent.futures libraries provide accessible approaches for implementing parallel processing patterns in data workflows.

Memory optimization strategies become increasingly important as data volumes grow. Techniques such as chunked processing, memory-mapped files, and streaming data processing enable handling of datasets that exceed available memory while maintaining processing performance.

Database query optimization ensures efficient data retrieval and storage operations. Proper indexing strategies, query optimization techniques, and connection pooling practices minimize database load while maximizing data access performance.

Caching strategies reduce redundant processing and data access operations. In-memory caching, distributed caching, and application-level caching techniques can significantly improve system responsiveness while reducing computational resource utilization.

2.3 Geospatial Technologies and Visualization Systems

Infrastructure monitoring systems require sophisticated geospatial capabilities to effectively represent and analyze the spatial relationships inherent in distributed infrastructure networks. The integration of spatial databases, geographic information systems, and advanced visualization platforms enables comprehensive spatial analysis and intuitive presentation of complex infrastructure data.

2.3.1 PostgreSQL and PostGIS for Spatial Data Management

PostgreSQL represents a mature, feature-rich relational database management system that has gained widespread adoption for applications requiring robust data management capabilities [19]. The addition of the PostGIS extension transforms PostgreSQL into a powerful spatial database capable of storing, indexing, and querying geographic data with high performance and reliability [20].

PostGIS provides comprehensive support for geographic objects including points, lines, polygons, and complex geometric shapes essential for representing water distribution network components. The extension implements the OpenGIS Simple Features Specification, ensuring compatibility with standard geographic information system applications and data exchange formats [20].

Spatial indexing capabilities provided by PostGIS enable efficient querying of geographic data even with large datasets typical in infrastructure monitoring applications. R-tree indexing structures optimize spatial queries including point-in-polygon tests, proximity searches, and geometric intersection calculations that are fundamental to infrastructure analysis operations.

The integration of spatial and temporal data within unified database structures enables sophisticated spatio-temporal analysis capabilities. Water distribution networks require analysis of how spatial relationships change over time, and the combination of PostgreSQL's temporal data handling with PostGIS spatial capabilities provides comprehensive support for these requirements.

Advanced spatial analysis functions provided by PostGIS include geometric calculations, topology operations, and spatial relationship testing. These functions enable complex analysis operations such as network connectivity analysis, service area calculations, and proximity-based queries essential for infrastructure monitoring applications.

2.3.2 QGIS for Geographic Information System Capabilities

QGIS (Quantum Geographic Information System) provides comprehensive open-source *GIS* capabilities essential for spatial data management, analysis, and visualization in infrastructure monitoring applications [21]. The platform's extensive functionality and plugin ecosystem make it particularly suitable for complex geospatial analysis requirements.

Figure 2.2: Illustration of QGIS Web Client (QWC) displaying QGIS Projects published by QGIS Server via WMS.

QGIS Server extends desktop *GIS* capabilities to web-based environments, enabling centralized geospatial data services that can be accessed by multiple client applications. The server component implements Open Geospatial Consortium (*OGC*) web service standards including Web Map Service (*WMS*) and Web Feature Service (*WFS*), ensuring interoperability with diverse client applications [21]. Figure 2.2 illustrates the core concept of QWC displaying QGIS Projects published by QGIS Server via WMS.

The platform's support for diverse data formats enables integration with existing infrastructure data regardless of original format or source system. QGIS can read and write numerous vector and raster formats while providing data transformation capabilities for format conversion and coordinate system transformations.

Advanced cartographic capabilities provided by *QGIS* enable creation of professional-quality maps and visualizations suitable for operational and presentation purposes. The platform's symbol management, labeling capabilities, and layout tools support creation of comprehensive mapping products that effectively communicate spatial information.

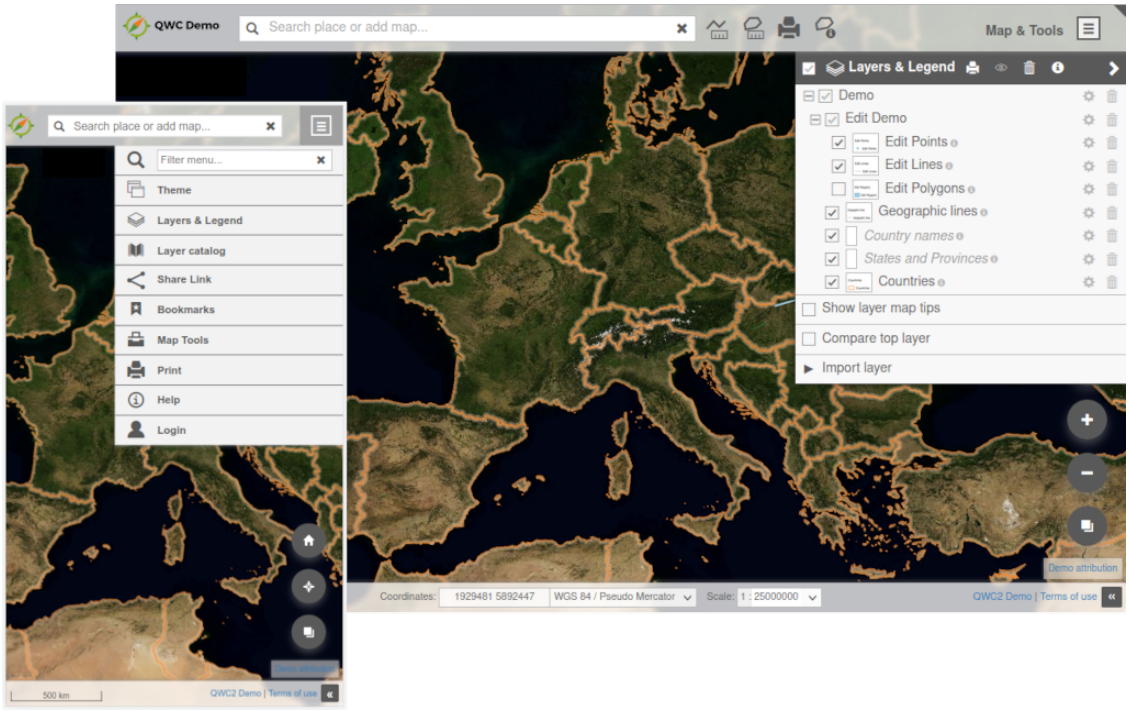

Figure 2.3: Overview of the QGIS Web Client user interface and architecture.

QGIS Web Client provides browser-based access to *GIS* capabilities, enabling users to interact with spatial data without requiring specialized desktop software installations. The web client supports common *GIS* operations including map navigation, layer management, and feature identification while maintaining responsive

performance characteristics. Figure 2.3 provides an overview of the QGIS Web Client architecture and user interface.

2.3.3 Grafana for Real-time Visualization

Grafana has emerged as a leading platform for creating interactive dashboards and visualizations, particularly for time series data common in monitoring applications [1]. The platform's extensible architecture and comprehensive visualization capabilities make it well-suited for infrastructure monitoring dashboards that require real-time data display and user interaction.

Grafana's data source abstraction layer enables integration with diverse data storage systems including relational databases, time series databases, and cloud-based analytics platforms. This flexibility supports monitoring applications that utilize multiple data sources while providing unified visualization interfaces [1].

The platform's extensive panel library provides specialized visualization types optimized for different data characteristics and user requirements. Time series panels excel at displaying temporal data trends, while stat panels provide clear presentation of current values and status indicators essential for monitoring applications.

Grafana's alerting capabilities enable proactive notification of system conditions requiring operator attention. Alert rules can be configured based on data thresholds, trend analysis, or complex queries, with notifications delivered through multiple channels including email, SMS, and integration with incident management systems.

Variable support within Grafana enables creation of dynamic dashboards that can be customized for different operational contexts. Template variables allow users to filter data by location, time range, or system component while maintaining consistent dashboard layouts and functionality.

2.3.4 GeoJSON for Geographic Data Exchange

GeoJSON represents a widely-adopted standard for encoding geographic data structures using JavaScript Object Notation (*JSON*) format. The format's simplicity and broad support across web-based applications make it particularly suitable for exchanging geographic data between different systems and platforms.

The *GeoJSON* specification defines standardized representations for common geographic features including points, lines, polygons, and feature collections. This standardization enables reliable data exchange between geographic information systems while maintaining compatibility with web-based visualization libraries [22].

GeoJSON's integration with web technologies enables efficient transmission and processing of geographic data in web-based monitoring applications. The format's JSON foundation ensures compatibility with modern web development frameworks

while maintaining human-readable data structures that facilitate debugging and data validation.

Feature properties within GeoJSON objects enable association of descriptive attributes with geographic features, supporting rich data models that combine spatial and non-spatial information. This capability proves essential for infrastructure monitoring applications where geographic features must be associated with operational data and metadata.

The format's support for coordinate reference systems enables proper handling of geographic data regardless of original projection or coordinate system. This flexibility supports integration of data from diverse sources while maintaining spatial accuracy and consistency.

2.3.5 Advanced Visualization Techniques for Infrastructure Data

Infrastructure monitoring applications require specialized visualization techniques that can effectively communicate complex spatial and temporal relationships while supporting operational decision-making. Advanced visualization approaches enhance user understanding while enabling efficient identification of system conditions requiring attention.

Heat map visualizations provide effective methods for displaying spatial distribution of scalar values across geographic areas. These techniques prove particularly valuable for displaying system performance metrics, identifying areas of concern, and communicating overall system status through intuitive color-coding schemes.

Network visualization techniques enable clear representation of connectivity relationships within infrastructure systems. Specialized layouts and styling approaches can highlight critical pathways, display flow directions, and indicate system capacity constraints while maintaining visual clarity even with complex network topologies.

Time-lapse visualization capabilities enable effective communication of temporal changes in system conditions. These techniques prove valuable for analyzing system evolution, identifying temporal patterns, and communicating system behavior to stakeholders with varying technical backgrounds.

Interactive visualization elements enhance user engagement while enabling detailed exploration of system data. Techniques such as drill-down capabilities, filtering controls, and dynamic data querying enable users to efficiently access relevant information while maintaining overview of overall system status.

2.4 Conversational AI and Natural Language Processing Frameworks

The development of natural language interfaces for technical systems represents a significant advancement in human-computer interaction, enabling intuitive access to complex system capabilities through conversational interactions. Modern conversational *AI* frameworks provide sophisticated capabilities for natural language understanding, dialog management, and system integration that enable effective deployment of conversational interfaces in infrastructure monitoring applications.

2.4.1 Large Language Models and Generative AI

Large Language Models (*LLMs*) have revolutionized natural language processing applications by providing sophisticated capabilities for understanding and generating human-like text [23]. These models, trained on extensive text corpora, demonstrate remarkable abilities in language understanding, reasoning, and generation that enable sophisticated conversational *AI* applications.

Google Gemini 2.0 Flash represents a state-of-the-art large language model that provides advanced natural language understanding and generation capabilities particularly suitable for technical applications [3]. The model's training on diverse text sources including technical documentation and code repositories enables effective handling of technical terminology and domain-specific concepts.

The model's multimodal capabilities enable processing of both text and structured data formats, supporting applications that require integration of natural language processing with data analysis and system operation tasks. This capability proves essential for infrastructure monitoring applications where conversational interfaces must interpret user requests and translate them into appropriate system operations.

Prompt engineering techniques enable optimization of large language model performance for specific application domains. Careful construction of system prompts, few-shot examples, and context management strategies can significantly improve model accuracy and reliability for domain-specific tasks while reducing hallucination risks.

The integration of large language models with external tools and systems through function calling capabilities enables conversational interfaces to perform complex operations beyond text generation. These capabilities support development of conversational agents that can query databases, manipulate system configurations, and coordinate multiple system operations in response to natural language requests.

2.4.2 LangChain Framework for AI Application Development

LangChain has emerged as a leading framework for developing applications that integrate large language models with external systems and data sources [4]. The framework's modular architecture and extensive integration capabilities make it particularly suitable for developing sophisticated conversational *AI* applications in technical domains.

The framework's chain abstraction enables composition of complex workflows that combine multiple processing steps including prompt generation, model invocation, output parsing, and system integration. This approach supports development of sophisticated conversational flows that can handle multi-step reasoning and complex system interactions [4].

LangChain's memory management capabilities enable maintenance of conversation context across multiple interaction turns, supporting natural conversational flows where previous exchanges inform subsequent responses. Memory implementations range from simple conversation history to sophisticated semantic memory systems that maintain relevant context while managing memory limitations.

Tool integration capabilities provided by LangChain enable conversational agents to interact with external systems including databases, *APIs*, and specialized software components. The framework's standardized tool interface supports development of custom integrations while providing pre-built tools for common integration requirements.

The framework's agent architecture enables development of conversational systems that can plan and execute complex multi-step operations in response to user requests. Agents can analyze user intentions, determine appropriate action sequences, and coordinate multiple tool invocations while maintaining appropriate user interaction throughout the process.

2.4.3 LangGraph for Workflow and State Management

LangGraph extends LangChain's capabilities by providing sophisticated workflow orchestration and state management specifically designed for complex conversational *AI* applications [5]. The framework's graph-based approach to workflow definition enables clear specification of conversation flows while supporting complex branching logic and error handling.

The framework's state management capabilities enable maintenance of complex application state throughout conversational interactions. State objects can include conversation history, system context, user preferences, and operational data, providing comprehensive context for conversational agent decision-making [5].

Graph-based workflow definition enables visual specification of conversation

flows including decision points, parallel processing paths, and error handling procedures. This approach supports development of sophisticated conversational experiences while maintaining clear documentation of system behavior.

The framework's conditional routing capabilities enable dynamic conversation flows that adapt based on user input, system conditions, and conversation context. Routing logic can implement complex decision trees while maintaining natural conversation flows that respond appropriately to user needs and system constraints.

Integration with LangChain's tool ecosystem enables LangGraph workflows to coordinate multiple system operations while maintaining appropriate conversation management. Workflows can include database queries, system modifications, and external service integrations within unified conversational experiences.

2.4.4 Natural Language Understanding for Technical Domains

Natural language understanding in technical domains presents unique challenges including specialized terminology, complex domain concepts, and precise operational requirements. Effective *NLU* systems for infrastructure monitoring applications must accurately interpret user intentions while maintaining appropriate precision for system operation tasks.

Intent classification techniques enable identification of user goals from natural language input, supporting routing of requests to appropriate system capabilities. Machine learning approaches including transformer-based models provide sophisticated classification capabilities while supporting adaptation to domain-specific terminology and usage patterns.

Entity extraction capabilities enable identification of specific parameters, values, and system components mentioned in user requests. Named entity recognition techniques adapted for technical domains can identify infrastructure components, temporal specifications, and operational parameters essential for system operation tasks.

Context management strategies ensure that conversational systems maintain appropriate understanding of conversation context while handling domain-specific references and implicit information. Techniques including coreference resolution and context window management support natural conversation flows in technical applications.

Query generation capabilities enable translation of natural language requests into appropriate database queries, API calls, and system operations. Template-based approaches combined with semantic understanding can generate accurate technical operations while maintaining user control and transparency.

2.4.5 Dialog Management and User Experience Design

Effective dialog management represents a critical component of conversational AI systems, ensuring natural interaction flows while maintaining appropriate user control and system reliability. Infrastructure monitoring applications require dialog management approaches that balance conversational naturalness with operational precision and safety.

Turn-taking management enables natural conversation flows while ensuring appropriate system response timing. Techniques including response acknowledgment, processing indicators, and appropriate wait handling support user expectations for conversational interactions with technical systems.

Confirmation and verification mechanisms ensure user control over potentially significant system operations while maintaining natural conversation flows. Multi-level confirmation strategies can provide appropriate safety checks for different operation types while avoiding excessive interaction overhead for routine tasks.

Error handling and recovery strategies enable graceful management of system failures, user input errors, and unexpected conditions. Recovery mechanisms should provide clear error communication while offering appropriate paths for users to achieve their intended goals despite initial failures.

Conversation repair capabilities enable correction of misunderstandings or errors without requiring conversation restart. Techniques including clarification requests, suggestion mechanisms, and undo capabilities support natural error recovery while maintaining conversation context and user confidence.

2.5 EPANET for Water Distribution Network Modeling

2.5.1 Overview of EPANET

EPANET represents the most widely adopted software platform for hydraulic modeling of water distribution networks, providing comprehensive simulation capabilities for pressure, flow, and water quality analysis [24]. Developed by the U.S. Environmental Protection Agency, EPANET has become the de facto standard for water distribution system analysis in both research and operational applications.

2.5.2 EPANET Simulation Workflow

EPANET functions as an advanced hydraulic simulation engine that processes real-world network data to calculate pressure and flow distributions throughout water distribution systems. Its core functionality centers on processing input data from flow sensors at key inlet points and boundary conditions to generate comprehensive hydraulic predictions across the entire network.

2.5.3 Sensor Data Integration

The simulation workflow begins with the integration of flow measurements from sensors positioned at network inlets, pumping stations, and other strategic locations. These measurements provide critical boundary conditions that anchor the hydraulic model to real-world operating conditions. EPANET can process both real-time sensor data streams and historical flow records, enabling both operational monitoring and retrospective analysis capabilities [25].

2.5.4 Hydraulic Simulation Engine

Once sensor data is integrated, EPANET's hydraulic engine implements fundamental fluid mechanics principles including conservation of mass and energy equations to calculate system-wide pressure and flow distributions. The simulation process applies these principles across the entire network topology, accounting for:

- Pipe characteristics including diameter, length, and roughness coefficients
- Network connectivity and topology relationships
- Elevation differences between network nodes
- Operational status of valves, pumps, and other control elements

2.5.5 Node-Level Output Generation

The primary output of the EPANET simulation process includes comprehensive pressure and flow predictions for every node and link in the water distribution network. These outputs provide a complete hydraulic profile of the system that extends well beyond the limited visibility provided by physical sensors alone. The node-level outputs include:

- Pressure values at each junction node
- Flow rates and velocities in each pipe segment
- Direction of flow throughout the network
- Hydraulic grade lines across the system

The simulation results enable network operators to visualize conditions throughout the entire distribution system, including areas without physical monitoring equipment, providing complete visibility into network behavior based on the available sensor inputs.

2.5.6 Applications in Network Monitoring

EPANET's simulation capabilities enable numerous practical applications in water distribution network monitoring and management. The software supports both real-time operational decisions and long-term planning processes through its ability to transform limited sensor data into comprehensive network visibility.

Real-time Operational Monitoring

When integrated with SCADA systems, EPANET can provide continuous network state estimation by processing real-time flow measurements from inlet sensors. This integration enables operators to maintain constant awareness of pressure conditions throughout the network, including areas without physical sensors. The approach significantly enhances operational visibility while minimizing physical sensor deployment requirements.

Hydraulic Anomaly Detection

By comparing actual sensor readings with expected values from the hydraulic model, EPANET enables detection of discrepancies that may indicate leaks, valve malfunctions, or other operational issues. This model-based anomaly detection approach extends detection capabilities beyond physically monitored locations to the entire simulated network.

Operational Scenario Analysis

EPANET supports evaluation of hypothetical operational scenarios including demand variations, pump schedule modifications, or valve configuration changes. Operators can simulate these scenarios to predict system-wide impacts before implementing changes in the physical network, reducing operational risks and optimizing decision-making processes.

2.5.7 EPANET Integration with Monitoring Systems

Modern water distribution monitoring systems typically implement EPANET as a core component within a broader monitoring architecture. Integration strategies enable automated simulation processes that continuously generate network-wide pressure and flow estimates based on available sensor inputs.

Data Integration Architecture

Effective EPANET integration requires robust data interfaces that connect sensor measurement systems with the hydraulic simulation engine. These interfaces typically include:

- Data acquisition modules that collect and pre-process sensor readings
- Validation routines that identify and filter problematic sensor data
- Calibration mechanisms that adjust model parameters to match observed conditions
- Output processing components that translate simulation results into actionable information

Model Calibration Processes

To maintain simulation accuracy, EPANET implementations require regular calibration processes that align model parameters with physical network characteristics. Automated calibration approaches use historical sensor data to optimize parameters including pipe roughness coefficients, minor loss factors, and demand distribution patterns. Well-calibrated models significantly enhance the reliability of pressure and flow predictions throughout the network.

2.6 Web Technologies and Real-time Communication

Modern monitoring systems require sophisticated web technologies and real-time communication capabilities to support responsive user interfaces, efficient data exchange, and seamless integration between diverse system components. The selection and implementation of appropriate web technologies significantly influence system usability, performance, and maintainability.

2.6.1 FastAPI for High-Performance Web Services

FastAPI has emerged as a leading Python web framework for developing high-performance *API* services, combining the simplicity of modern Python development with the performance characteristics required for production applications [6]. The framework's design principles emphasize developer productivity while maintaining excellent runtime performance and comprehensive feature support.

The framework's automatic *API* documentation generation provides significant advantages for system integration and maintenance. FastAPI automatically generates interactive *API* documentation using OpenAPI and *JSON* Schema standards, enabling developers and system integrators to understand and test *API* endpoints without additional documentation overhead [26].

Type hint integration represents a core feature that enables both improved code quality and automatic validation. Python type hints provide static analysis

benefits while enabling FastAPI to automatically validate request and response data, reducing development time and improving system reliability through early error detection.

Asynchronous request handling capabilities enable efficient processing of concurrent requests while maintaining responsive performance characteristics. The framework's async/await support enables non-blocking operations for database queries, external *API* calls, and file operations that are common in monitoring applications.

WebSocket support provided by FastAPI enables real-time bidirectional communication between client applications and server components. This capability proves essential for monitoring applications that require real-time data updates and interactive user experiences that respond immediately to system changes.

The implementation establishes persistent WebSocket connections with comprehensive state management for each client session:

2.6.2 WebSocket Protocol for Real-time Communication

WebSocket protocol provides full-duplex communication capabilities over single TCP connections, enabling efficient real-time data exchange between web applications and server components. The protocol's design addresses limitations of traditional HTTP request-response patterns while maintaining compatibility with existing web infrastructure.

The persistent connection model eliminates the overhead associated with establishing new connections for each data exchange, enabling efficient transmission of frequent updates typical in monitoring applications. This characteristic proves particularly valuable for dashboards and visualization interfaces that require continuous data updates [7].

Low-latency communication capabilities enable responsive user interfaces that can react immediately to system changes or user interactions. The protocol's message-based communication model supports both text and binary data transmission while maintaining minimal protocol overhead.

Bidirectional communication enables both server-initiated data pushes and client-initiated requests within unified communication sessions. This capability supports monitoring applications where both automatic data updates and user-initiated queries must be handled efficiently within single connections.

Browser compatibility across modern web browsers ensures broad accessibility while native browser *APIs* provide comprehensive programming interfaces for WebSocket integration. This compatibility enables development of monitoring interfaces that work reliably across diverse client environments without requiring specialized plugins or applications.

2.6.3 React for Interactive User Interfaces

React 19 has established itself as a leading JavaScript library for building interactive user interfaces, providing sophisticated component-based development approaches that enable creation of complex, maintainable web applications [9]. The library's design principles emphasize reusability, predictability, and performance optimization essential for sophisticated monitoring interfaces.

Component-based architecture enables development of reusable interface elements that encapsulate both visual presentation and interactive behavior. This approach supports creation of comprehensive component libraries that can be consistently applied across different monitoring interfaces while maintaining design coherence and reducing development overhead [9].

Virtual *DOM* implementation provides performance optimization for dynamic interfaces that frequently update in response to changing data. React's reconciliation algorithm minimizes actual *DOM* manipulations while ensuring interface responsiveness even with complex data visualizations and frequent updates common in monitoring applications.

State management capabilities enable sophisticated handling of application data and user interface state while maintaining predictable component behavior. React's unidirectional data flow pattern supports development of complex interfaces while enabling effective debugging and testing practices.

Ecosystem integration with extensive third-party libraries provides access to specialized components for data visualization, form handling, and user interface enhancement. This ecosystem enables rapid development of sophisticated monitoring interfaces while leveraging proven solutions for common interface requirements.

2.6.4 Vite for Modern Development Tooling

Vite represents a modern build tool and development server that addresses performance and developer experience limitations of traditional bundling approaches [27]. The tool's design leverages modern browser capabilities while providing efficient development workflows essential for complex web application development.

Hot module replacement capabilities enable real-time code updates during development without losing application state, significantly improving development velocity for complex interfaces. This capability proves particularly valuable when developing monitoring interfaces that require extensive testing with real-time data flows [27].

Native ES module support eliminates bundling overhead during development while leveraging browser-native module loading capabilities. This approach provides immediate feedback for code changes while supporting modern JavaScript development practices including dynamic imports and code splitting.

Optimized production builds utilize advanced bundling and optimization techniques including tree shaking, code splitting, and asset optimization. These optimizations ensure efficient loading performance for monitoring applications while maintaining comprehensive feature support.

Plugin architecture enables integration with diverse development tools and frameworks while supporting customization for specific project requirements. Vite's plugin ecosystem provides solutions for common development needs including CSS preprocessing, testing integration, and deployment optimization.

2.6.5 REST API Design and Integration Patterns

Representational State Transfer (REST) architectural principles provide a foundation for designing scalable, maintainable web APIs that support integration between diverse system components. Effective REST API design enables reliable system integration while supporting evolution and maintenance of complex monitoring systems.

Resource-oriented design approaches enable intuitive API structures that map naturally to system entities and operations. Well-designed resource hierarchies support both simple operations and complex queries while maintaining consistent interaction patterns across different API endpoints [28].

HTTP method semantics provide standardized approaches for different operation types including data retrieval, creation, modification, and deletion. Proper use of HTTP methods enables appropriate caching behavior, error handling, and tool integration while maintaining predictable API behavior.

Status code utilization enables precise communication of operation results while supporting appropriate client-side error handling and retry logic. Comprehensive status code usage provides clear feedback for both successful operations and various error conditions that may occur during system integration.

Authentication and authorization mechanisms ensure appropriate security controls while supporting diverse integration requirements. Token-based authentication approaches including JWT (JSON Web Tokens) provide secure, scalable solutions for API access control while maintaining compatibility with modern web development practices.

Content negotiation capabilities enable flexible data format support while accommodating diverse client requirements. Support for multiple content types including JSON, XML, and specialized formats enables broad compatibility while optimizing data exchange efficiency for different use cases.

2.7 Chapter Summary and Research Direction

This comprehensive literature review establishes the theoretical and technological foundation for the intelligent water distribution monitoring system presented in this thesis. The examination of each technology domain reveals that while significant advances have been made in individual areas including cloud computing platforms, data processing frameworks, conversational *AI*, and visualization technologies, substantial opportunities exist for innovative integration of these capabilities to address persistent challenges in infrastructure monitoring.

The Google Cloud Platform provides the foundational infrastructure for scalable data storage, processing, and machine learning capabilities that enable the comprehensive monitoring system developed in this thesis. The integration of BigQuery for large-scale data analytics, Cloud Functions for serverless processing, and Cloud Storage for reliable data persistence creates a robust foundation for infrastructure monitoring applications.

Apache Airflow's [16] workflow orchestration capabilities, combined with Python's extensive data processing libraries [12], provide the framework for automated data collection, processing, and quality assurance that ensures reliable system operation. The combination of these technologies enables sophisticated data pipeline management while maintaining operational simplicity and reliability.

The integration of PostgreSQL [19] with PostGIS extensions [20] and QGIS Web Client [29], and modern web mapping technologies creates comprehensive geospatial capabilities that are essential for infrastructure monitoring applications. The ability to visualize and analyze spatially distributed monitoring data provides critical insights for operational decision-making and system optimization.

LangChain and LangGraph frameworks, combined with Google's Gemini 2.0 Flash language model, provide the foundation for conversational *AI* capabilities that make complex technical systems accessible through natural language interaction. The integration of these technologies with function calling capabilities enables sophisticated system interaction while maintaining appropriate safety and validation mechanisms.

The specialized water distribution monitoring technologies, including hydraulic modeling software [24], provide domain-specific capabilities that are essential for effective infrastructure monitoring. The integration of these specialized tools with modern cloud platforms and *AI* technologies creates new possibilities for comprehensive monitoring and management.

Modern web technologies, including React 19, FastAPI, and WebSocket communication, enable the development of responsive, real-time user interfaces that can effectively present complex monitoring data while supporting sophisticated user interaction patterns. The combination of these technologies with cloud-native deployment approaches ensures scalable and reliable system operation.

The identified research gaps, particularly in the areas of conversational *AI* for

technical applications, intelligent dashboard management, and cloud-native monitoring architectures, provide clear justification for the research direction and innovations presented in subsequent chapters. The integration challenges and opportunities identified through this review inform the architectural decisions and implementation approaches used throughout the system development process.

The foundation established through this literature review demonstrates that the convergence of cloud computing, artificial intelligence, and modern web technologies creates unprecedented opportunities for developing intelligent infrastructure monitoring systems. The specific combination of technologies examined in this chapter provides the technical foundation necessary to address the complex requirements of modern water distribution monitoring while enabling new paradigms for human-computer interaction in technical domains.

The research direction established through this analysis emphasizes the importance of comprehensive technology integration, user-centered design principles, and practical operational considerations in developing effective monitoring systems. The subsequent chapters build upon this foundation to demonstrate how these diverse technologies can be integrated to create innovative solutions that address real-world infrastructure monitoring challenges while advancing the state of the art in intelligent system design.

Chapter 3

Materials and Methods

This chapter presents the comprehensive approach taken for designing and developing the intelligent water distribution monitoring system. The chapter is structured to first provide insight into the development methodology and evolutionary process that guided the research, followed by the systematic architectural design principles and system design decisions, and finally the detailed technical implementation of the developed system components. This organization enables readers to understand both the methodological foundation and the practical realization of the intelligent monitoring system.

3.1 Development Journey and Methodology

The development of the intelligent water distribution monitoring system represented a comprehensive journey through multiple technological domains, integration challenges, and iterative refinement processes. This section provides detailed insight into the methodological approach, decision-making processes, and evolutionary development that led to the final system architecture and implementation. The methodology demonstrates how systematic investigation and iterative development enabled the creation of innovative solutions for complex infrastructure monitoring challenges, with particular emphasis on system integration, cloud infrastructure development, and conversational AI interface innovation.

Collaborative Research Context: This thesis represents part of a collaborative research effort where machine learning algorithm development was conducted by a research colleague, while this work focused on comprehensive system integration, cloud infrastructure implementation, visualization development, and the innovative conversational AI interface that makes complex analytical results accessible through natural language interaction.

3.1.1 Research and Development Methodology

The research approach employed for this work follows a systematic design science research framework that combines theoretical investigation with practical implementation and comprehensive evaluation. This methodology ensures that research outcomes are both technically sound and practically applicable to real-world operational environments.

The design science approach includes several key phases: problem identification and motivation based on analysis of existing system limitations and operational requirements, solution objective definition that establishes clear criteria for successful outcomes, design and development activities that create innovative technical artifacts, demonstration of solution capabilities through prototype implementation, and comprehensive evaluation that assesses solution effectiveness across multiple dimensions.

The iterative development process enabled continuous refinement of solution approaches based on evaluation results and emerging insights. This approach was particularly important for the conversational AI interface development, where user feedback and practical testing revealed important requirements that were not apparent during initial conceptualization.

Systems engineering principles guided the technical development activities, emphasizing systematic analysis of requirements, structured design processes, and comprehensive testing and validation procedures. This approach was essential for managing the complexity of integrated systems that must satisfy multiple stakeholder requirements while maintaining reliability and performance characteristics.

3.1.2 Initial Exploration and Technology Foundation

The project commenced during an internship period that provided the foundation for the comprehensive system described in this thesis. The initial project scope centered on integrating water distribution system data into a GIS platform with user visualization capabilities. Through systematic evaluation of available technologies and practical implementation experience, the project evolved to incorporate advanced conversational AI capabilities that became the central innovation of this thesis.

Preliminary QGIS-Based Implementation

The development journey began with a comprehensive exploration of existing geospatial and business intelligence tools during the preliminary implementation phase. This initial phase established the technical foundation while revealing critical limitations that informed subsequent development decisions toward the final integrated system architecture.

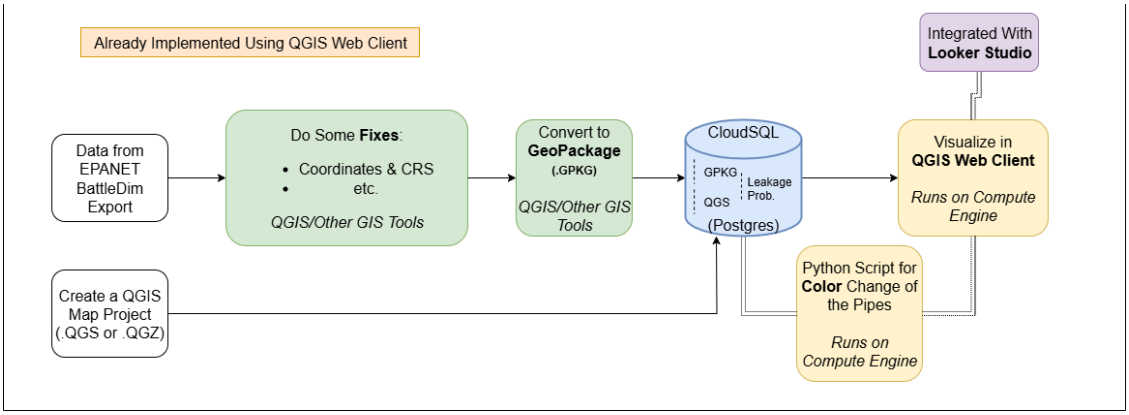

Figure 3.1: Schema for Geodata Pipeline Designed for the Preliminary Implementation

The preliminary implementation utilized QGIS Web Client and QGIS Server deployed on Google Compute Engine, providing comprehensive GIS capabilities for water distribution network visualization and basic editing functions. The pipeline for geodata, originating from Battledim and Epanet, involved fixes such as correcting CRS and coordinates manually in QGIS software, exporting the data as a GeoPackage, storing it in the PostgreSQL database, and visualizing it on QGIS Web Client hosted on a Compute Engine with integration of Google Looker Studio for time-series visualization, as shown in Figure 3.1. This initial approach leveraged the mature GIS ecosystem while exploring integration possibilities with modern cloud-based analytics platforms.

Core System Components:

The initial architecture integrated several specialized components:

- **PostgreSQL with PostGIS extensions** for spatial data storage, providing robust geospatial data management capabilities with support for complex geometric operations and spatial indexing
- **QGIS Server** for web mapping services, enabling publication of GIS projects as web services with support for standard OGC protocols including WMS and WFS
- **QGIS Web Client** for browser-based GIS interaction, providing full-featured GIS functionality accessible through web browsers without requiring desktop software installation
- **Google Looker Studio** for time-series data visualization, offering sophisticated business intelligence capabilities with integration to BigQuery data sources

- **Manual Python script execution** for data processing and analysis, providing flexibility for custom analytical workflows while requiring direct technical intervention

Database Schema Architecture:

The preliminary database architecture, illustrated in Figure 3.2, established design patterns that influenced the final system implementation:

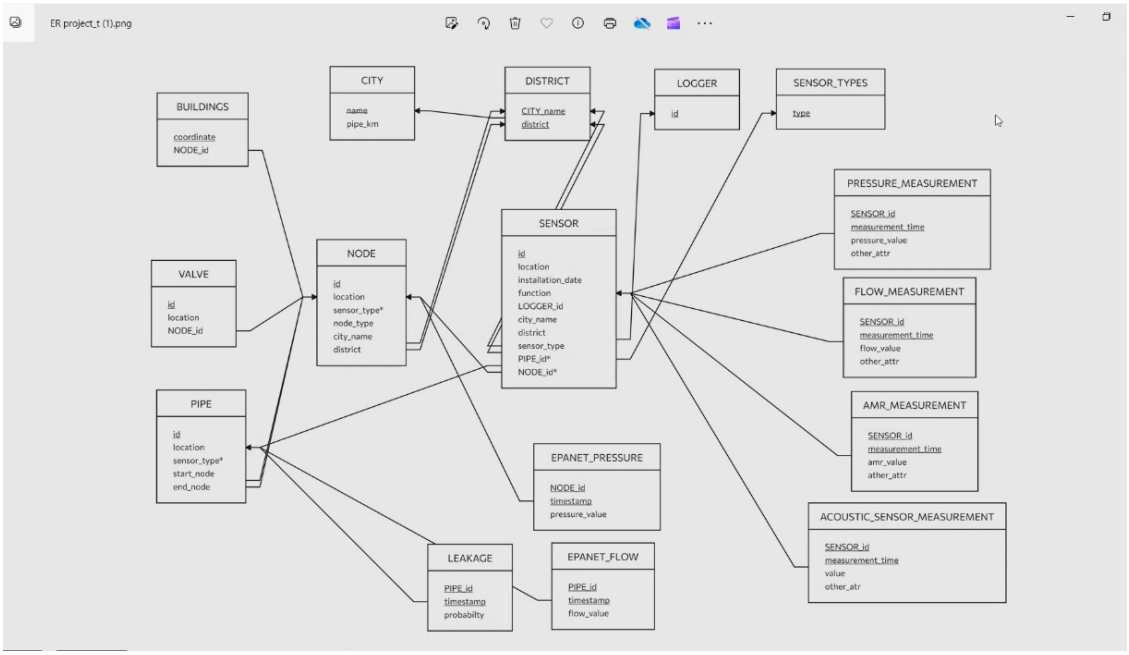

Figure 3.2: Entity-Relationship Diagram Designed as the Database Schema

GIS Data Schema (PostgreSQL/PostGIS):

- Network topology tables storing pipe segments, junctions, and system components with full geometric representation
- Sensor location tables with precise geographic coordinates and metadata
- District boundary definitions with polygon geometries for spatial analysis
- User authentication and authorization tables supporting role-based access control

Time-Series Data (BigQuery):

- Sensor measurement tables with timestamp partitioning for query performance optimization
- Calculated metrics tables storing derived values and analytical results
- Event logging tables capturing system activities and user interactions

Integration Challenges and Limitations Discovery

The preliminary implementation phase revealed several critical challenges that necessitated the evolution toward the final integrated system architecture:

URL Parameter Coordination: Initial attempts to coordinate between QGIS Web Client and Looker Studio through URL parameters proved complex and limited in functionality. The graphical representation of the water network in QGIS Web Client, as shown in Figure 3.3, includes components such as pipes, nodes, and sensors, along with the feature info panel and layers and legend panel. The approach required manual construction of complex URL strings and provided fragile integration that frequently broke due to parameter encoding issues or service updates.

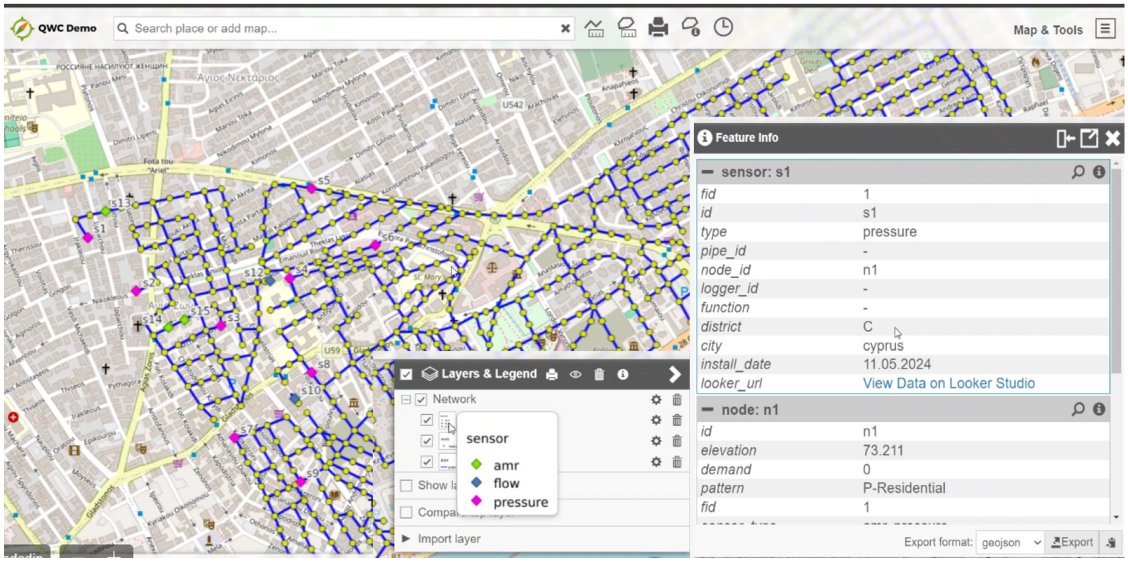

Figure 3.3: Graphical Representation of the Network in QGIS Web Client

Manual Script Execution: The preliminary system required manual execution of Python scripts for data processing and visualization updates. One such script was responsible for dynamically changing the pipe colors based on predicted leakage values, as illustrated in Figure 3.5. This approach created workflow interruptions, required technical expertise from users, and prevented automated processing workflows that could respond to data updates in real-time.

Multi-Interface Navigation: Users needed to navigate between multiple specialized interfaces to complete operational tasks, creating cognitive overhead and reducing operational efficiency. The switching between GIS tools, business intelligence dashboards, and command-line interfaces created barriers to effective system utilization.

Limited Interactivity: Traditional GIS and BI tools provided limited support for dynamic interactions that could adapt to user queries or operational contexts.

Users were constrained to predefined views and analysis approaches that could not accommodate the diverse operational requirements encountered in real-world monitoring scenarios.

These limitations provided valuable insights that informed the design requirements for the final integrated system, leading to the identification of needs for unified visualization platforms and intelligent conversational interfaces that could address these fundamental operational challenges.

3.1.3 Technology Selection and Strategic Evolution

The preliminary investigation phase involved comprehensive analysis of cloud computing platforms, database technologies, and visualization approaches. This phase established the technical foundation while revealing important limitations in user accessibility and system integration that motivated the development of intelligent conversational interfaces.

Technology selection decisions significantly influenced both development velocity and long-term system characteristics. The evaluation process required careful consideration of factors including technical capabilities, cost implications, learning curve requirements, and long-term maintenance considerations.

Critical Technology Decisions

Key technology decisions that shaped the system evolution included:

- **Google Cloud Platform (GCP):** Chosen as the primary infrastructure foundation for its comprehensive capabilities in data processing, machine learning integration, and scalable deployment.

- **PostgreSQL with PostGIS:** Selected for geospatial data management, reflecting the importance of spatial analysis for water distribution monitoring.

- **BigQuery:** Adopted for large-scale analytics, enabling sophisticated time-series analysis and historical trend evaluation.

- **Transition from QGIS to Grafana:** The transition from QGIS Web Client to Grafana represented a significant architectural decision. The initial implementation integrated QGIS Web Client for geospatial visualization and Looker Studio for analytics, which led to deployment complexity and an inconsistent user experience. After evaluating alternatives, including Google Looker which had cost and complexity limitations, Grafana was chosen for its ability to integrate mapping and analytics within a unified platform.

Visualization Platform Evolution

The evolution of visualization platforms reflects the systematic progression from fragmented tools toward integrated solutions:

Initial Multi-Tool Approach: The preliminary system utilized separate platforms for different visualization needs, creating operational complexity and user experience challenges. QGIS Web Client provided sophisticated geospatial capabilities but required specialized GIS expertise, while Looker Studio (Figure 3.4) offered powerful analytics but limited geospatial integration.

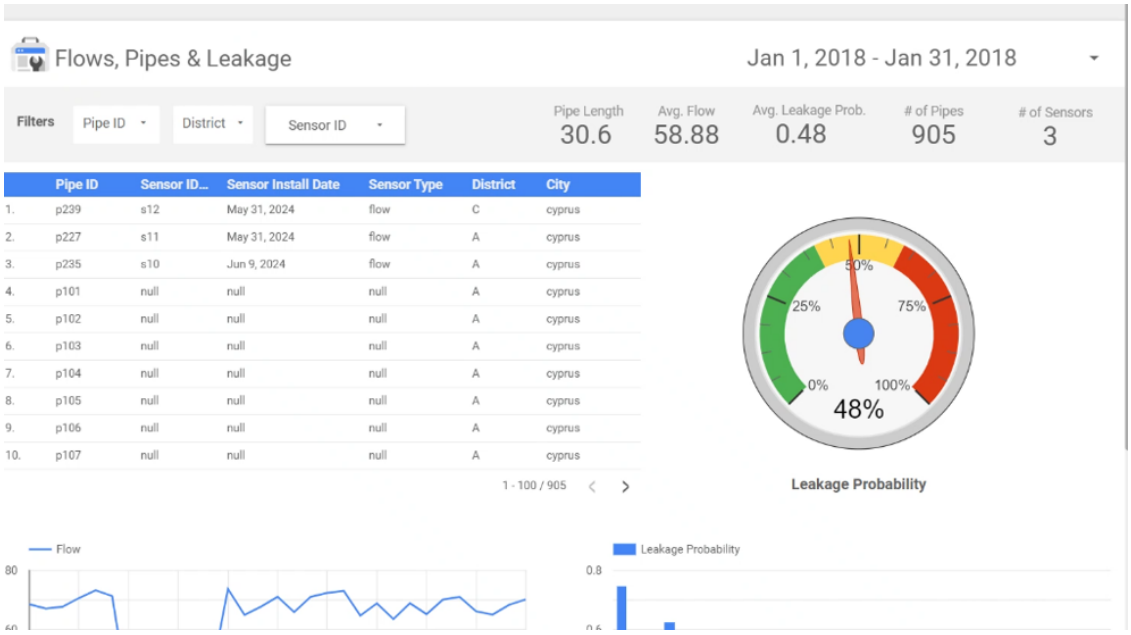

Figure 3.4: Looker Studio Dashboard

Integration Challenges: The multi-platform approach revealed several fundamental limitations including fragmented user experience requiring navigation between multiple specialized interfaces, inconsistent data refresh behaviors requiring manual intervention, limited customization capabilities preventing workflow adaptation, and absence of unified authentication mechanisms.

Strategic Evolution: The identification of these limitations motivated the strategic evolution toward Grafana as a unified visualization platform that could address both time-series analytics and geospatial visualization requirements through a single, integrated interface.

3.1.4 Iterative Development and Continuous Learning

The development process emphasized iterative refinement based on practical implementation experience and continuous learning from system behavior under realistic

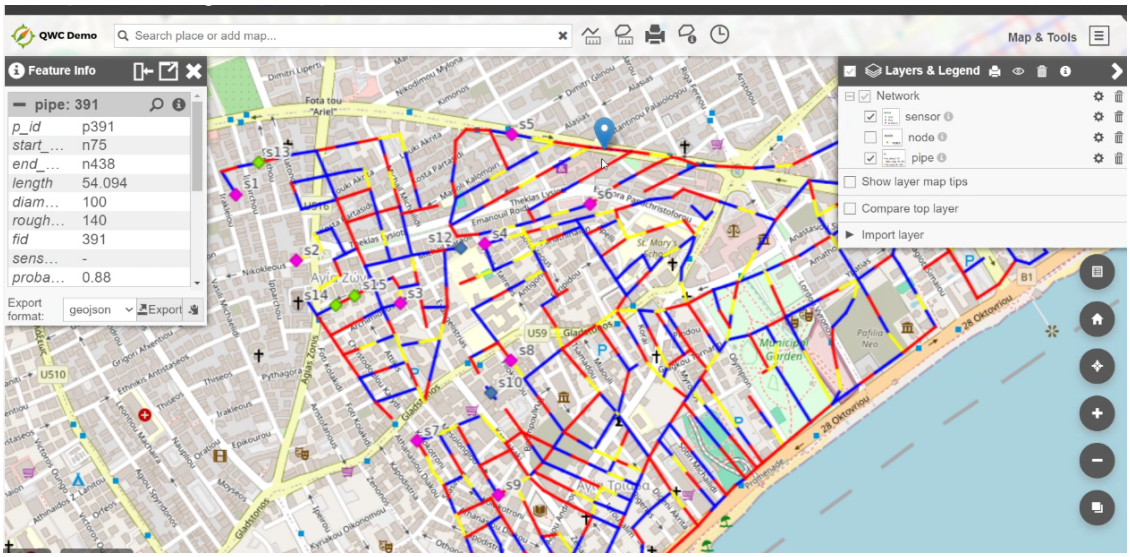

Figure 3.5: Different Coloring of the Pipes Representing the Level of Predicted Water Leakage Probability

operational conditions. This approach enabled the identification and resolution of issues that would not have been apparent through theoretical analysis alone.

Early prototyping work demonstrated the technical feasibility of integrating diverse data sources within cloud-native architectures while highlighting critical challenges in user experience design. Traditional monitoring interfaces required extensive domain expertise and complex navigation workflows that created barriers to effective system utilization.

User Experience Insights

User-centered design principles emerged as fundamental to creating systems that provide genuine operational value rather than merely technical sophistication. The emphasis on understanding user needs, operational workflows, and practical constraints proved essential for creating systems that achieve meaningful adoption and operational impact.

The preliminary system evaluation revealed that sophisticated technical capabilities become ineffective when they cannot be readily utilized by operational personnel in realistic working environments. This insight drove the development toward more intuitive, accessible interfaces that could accommodate diverse user backgrounds and expertise levels.

Evolution Toward Conversational AI

The evolution from basic integration to intelligent conversational AI represents a strategic advancement focused on creating cutting-edge user interaction paradigms using the latest developments in Large Language Models and conversational AI technologies. With the foundation of understanding established through the preliminary implementation, the development shifted toward creating an innovative conversational assistant that could serve as a reliable, intelligent interface for end users.

This evolution demonstrates the opportunity to transform user experience through natural language interaction, making sophisticated monitoring capabilities accessible through intuitive conversational commands while providing comprehensive functionality including dashboard management, data analysis, geospatial operations, and system control.

3.1.5 Conversational AI Development Journey

The development of conversational AI capabilities represented the most innovative aspect of the system evolution. Beginning with fundamental feasibility questions about LLM integration with Grafana systems, the methodology progressed through iterative phases that established the technical foundation for the final intelligent interface.

Initial Feasibility Exploration

The conversational AI development began with command-line interface experimentation that established the programmatic dashboard manipulation feasibility. This initial phase revealed both opportunities and limitations of direct LLM-based approaches while providing insights into optimal integration strategies.

Early testing revealed reliability issues including hallucinations and token consumption concerns with direct JSON manipulation approaches. These discoveries led to the development of a hybrid architecture where LLMs focus on natural language understanding while dedicated Python functions handle precise operations.

Architectural Insights and Refinements

Critical architectural insights emerged regarding optimal LLM utilization patterns. The development process revealed the importance of separating natural language understanding from precise technical operations, leading to more reliable and maintainable system architectures.

The progression to web-based systems required WebSocket implementation for real-time communication and conversation state persistence. This evolution enabled the integration of comprehensive capabilities including natural language data

querying, dashboard management, and geospatial operations through a unified conversational interface.

3.1.6 Lessons Learned and Development Insights

The comprehensive development journey provided numerous insights into effective approaches for creating sophisticated infrastructure monitoring systems that integrate artificial intelligence capabilities with traditional monitoring technologies. These lessons informed both the final system implementation and provide guidance for future development efforts.

Technical Architecture Insights

Incremental development approaches proved essential for managing the complexity inherent in sophisticated infrastructure monitoring systems. The ability to validate architectural decisions through practical implementation experience enabled course corrections and optimizations that significantly improved final system effectiveness.

Technology integration complexity required sophisticated architectural approaches that balanced flexibility with maintainability. The development experience demonstrated the importance of well-defined interfaces, comprehensive testing, and continuous monitoring for creating reliable integrated systems.

User Interface Evolution Insights

The conversational AI interface development revealed the transformative potential of natural language interaction for technical systems while highlighting the importance of appropriate user education, transparent operation, and robust error handling for successful adoption.

Performance optimization represented a continuous concern throughout the development process rather than a final implementation phase. The development experience demonstrated the importance of performance considerations in architectural design decisions and the value of continuous monitoring and optimization.

3.1.7 Methodology Contributions and Future Applications

This comprehensive development methodology illustrates the iterative, evolutionary nature of creating sophisticated infrastructure monitoring systems while highlighting the importance of user-centered design, robust architectural foundations, and continuous refinement based on practical implementation experience.

The methodological insights gained through this development process provide valuable guidance for future efforts to create intelligent, accessible infrastructure monitoring systems. The combination of systematic technology evaluation, iterative

development, and user-centered design principles creates a framework that can be applied to similar infrastructure monitoring challenges in diverse domains.

The development journey demonstrates that sophisticated technical capabilities can be made accessible to diverse operational personnel through appropriate interface design and conversational AI integration, providing a foundation for broader adoption of intelligent monitoring systems in infrastructure management applications.

3.2 System Architecture and Design Methodology

This section presents the comprehensive architectural design of the intelligent water distribution monitoring system, emphasizing the integration of cloud-native infrastructure, advanced data processing capabilities, and innovative conversational AI interfaces. The design methodology addresses the complex requirements of modern infrastructure monitoring while establishing a foundation for scalable, maintainable, and user-accessible monitoring solutions. The architectural approach balances technical sophistication with operational practicality, ensuring that advanced capabilities remain accessible to diverse operational personnel through intuitive interaction paradigms.

3.2.1 Architectural Design Philosophy and Principles

The system architecture embodies a comprehensive design philosophy that prioritizes user accessibility, technical robustness, and operational effectiveness. The architectural approach recognizes that sophisticated monitoring capabilities are only valuable when they can be effectively utilized by operational personnel with diverse technical backgrounds and varying levels of system expertise.

The modular design principle ensures that individual system components can be developed, tested, and maintained independently while contributing to coherent overall system functionality. This modularity enables selective enhancement of specific capabilities without requiring comprehensive system modifications, supporting both incremental improvement and strategic evolution of system capabilities.

Cloud-native architecture principles guide the infrastructure design, emphasizing scalability, reliability, and cost-effectiveness through intelligent utilization of managed cloud services [2]. The cloud-native approach enables automatic scaling based on demand patterns while reducing operational overhead through managed service utilization.

User-centered design principles inform interface development, ensuring that system complexity remains hidden behind intuitive interaction mechanisms. The emphasis on conversational *AI* interfaces reflects recognition that natural language

interaction can significantly reduce barriers to effective system utilization while enabling access to sophisticated analytical capabilities.

Data-driven decision making principles ensure that architectural choices are validated through practical implementation experience and performance measurement. The iterative refinement approach enables continuous optimization based on operational feedback and performance analysis.

3.2.2 High-Level System Architecture

The system architecture implements a distributed, event-driven design that coordinates multiple specialized subsystems through well-defined interfaces and communication protocols. The architecture addresses the complex requirements of real-time data processing, sophisticated analytical computation, and interactive visualization while maintaining appropriate separation of concerns and system reliability.

The data acquisition and processing subsystem provides the foundation for collecting, validating, and transforming sensor measurements from diverse monitoring networks. This subsystem implements comprehensive quality assurance mechanisms that ensure data integrity while supporting diverse sensor types and communication protocols.

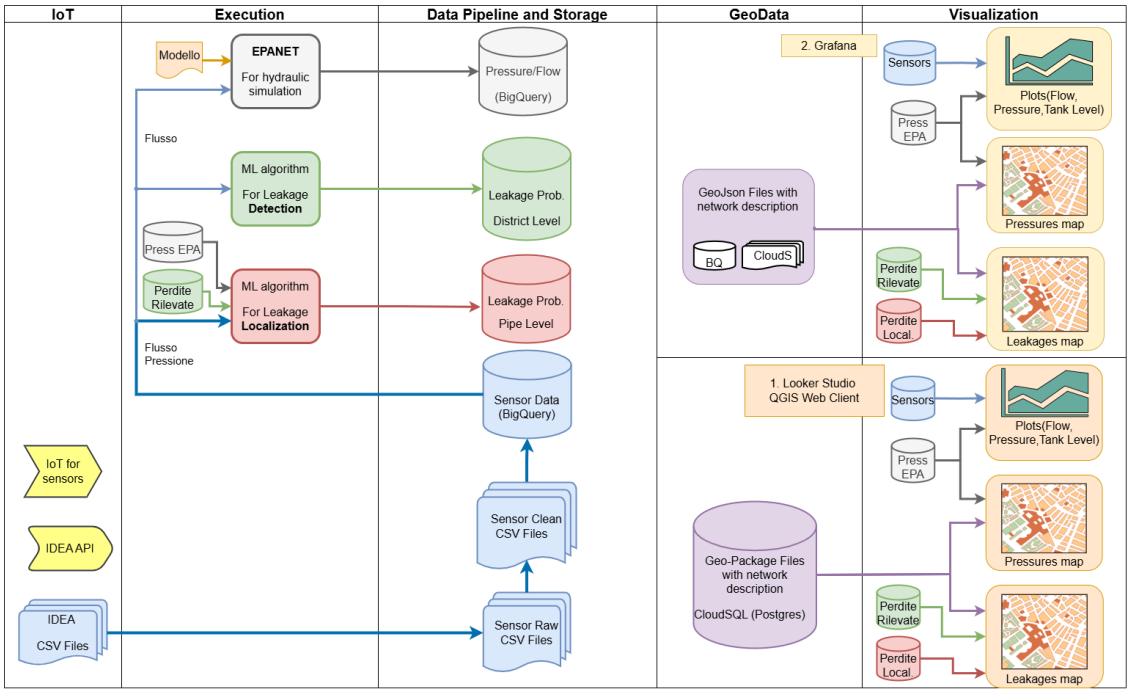

Figure 3.6: Comprehensive System Architecture Illustrating Integration of Cloud Infrastructure, Data Processing, Analytical Computation, and Intelligent User Interface Components

The computational analysis subsystem coordinates hydraulic simulation and machine learning algorithm integration workflows that transform sensor measurements into operational insights. This subsystem leverages distributed computing resources to execute complex simulations and integrate analytical algorithms developed by research colleagues while maintaining appropriate performance characteristics for operational monitoring requirements. Figure 3.6 illustrates the comprehensive system architecture and the integration of all major components including cloud infrastructure, data processing, analytical computation, and intelligent user interface elements.

The data management subsystem provides scalable, high-performance storage and retrieval capabilities for diverse data types including time-series sensor measurements, simulation results, geospatial network topology, and analytical outputs. The storage architecture utilizes specialized database technologies optimized for different data characteristics while maintaining unified access interfaces.

The visualization and interaction subsystem presents complex monitoring information through intuitive, interactive interfaces that support both routine operational tasks and sophisticated analytical workflows. This subsystem integrates traditional dashboard visualizations with innovative conversational *AI* capabilities that enable natural language interaction with complex technical systems.

The orchestration and automation subsystem coordinates workflow execution across distributed system components while providing comprehensive monitoring, error handling, and recovery capabilities. This subsystem ensures reliable operation under diverse conditions while enabling efficient resource utilization and cost optimization.

The architectural integration emphasizes loose coupling between subsystems while maintaining appropriate coordination and communication mechanisms. This approach enables independent evolution of individual components while preserving overall system coherence and reliability.

3.2.3 Intelligent Interface Architecture and Design

The conversational *AI* interface represents a fundamental architectural innovation that transforms traditional monitoring system interaction paradigms through natural language processing capabilities. The interface architecture addresses the critical challenge of making sophisticated technical systems accessible to operational personnel with diverse backgrounds and expertise levels.

The natural language processing layer implements sophisticated understanding capabilities that can interpret user intentions, extract relevant parameters, and translate conversational requests into appropriate system operations. This layer utilizes advanced large language models [23] combined with domain-specific knowledge to achieve reliable understanding of technical queries and operational commands.

The workflow orchestration layer coordinates complex multi-step operations involving multiple system components while maintaining appropriate user control and transparency. This layer implements sophisticated state management capabilities that track conversation context, user preferences, and ongoing operational tasks while providing appropriate confirmation and feedback mechanisms.

The system integration layer provides secure, efficient connectivity to all underlying system components including databases, visualization platforms, simulation tools, and analytical services. This layer abstracts the complexity of diverse system interfaces while providing unified access through conversational interactions.

3.2.4 Data Architecture and Information Flow Design

The data architecture implements a comprehensive approach to managing diverse information types while ensuring data quality, accessibility, and performance across various analytical and operational requirements. The architecture addresses the complex challenges of integrating time-series sensor data, geospatial network information, simulation results, and user interaction history within unified data management frameworks.

The ingestion architecture provides robust mechanisms for collecting sensor measurements from diverse sources while implementing comprehensive validation and quality assurance procedures. The processing architecture implements sophisticated transformation and enrichment capabilities that prepare raw sensor data for analytical and visualization requirements.

The storage architecture utilizes specialized database technologies optimized for different data characteristics and access patterns. Time-series data utilizes columnar storage optimized for analytical queries, geospatial data leverages spatial indexing for efficient geographic operations, and metadata utilizes normalized schemas for consistency and integrity.

The access architecture provides unified interfaces for data retrieval while implementing appropriate security, caching, and optimization mechanisms. The archival architecture implements comprehensive data lifecycle management that balances accessibility requirements with storage cost optimization.

3.2.5 Cloud Infrastructure Design and Scalability Architecture

The cloud infrastructure design implements comprehensive utilization of Google Cloud Platform services [2] while optimizing for performance, reliability, and cost-effectiveness. The infrastructure architecture addresses the variable workload characteristics inherent in infrastructure monitoring applications while providing appropriate scalability and fault tolerance capabilities.

The compute architecture utilizes a hybrid approach combining serverless functions for event-driven processing with managed container services for complex analytical workloads. The storage architecture leverages managed database services optimized for different data characteristics and access patterns.

The networking architecture implements secure, high-performance connectivity between distributed system components while providing appropriate access control and monitoring capabilities. The security architecture implements comprehensive protection mechanisms including identity and access management, data encryption, network security, and audit logging.

3.2.6 Integration Architecture and Interoperability Design

The integration architecture addresses the complex requirements of connecting diverse system components while maintaining appropriate security, performance, and reliability characteristics. The architecture emphasizes standardized interfaces and communication protocols that enable flexible system evolution while preserving operational stability.

The *API* architecture implements comprehensive service interfaces that provide secure, efficient access to system capabilities while abstracting implementation complexity. The messaging architecture provides reliable, scalable communication mechanisms for coordinating distributed system operations.

The security integration architecture ensures that authentication, authorization, and audit capabilities are consistently applied across all system components. The external system integration architecture provides flexible mechanisms for connecting with existing monitoring infrastructure while supporting various data formats, communication protocols, and operational procedures.

3.2.7 Design Goals and Quality Attributes

The architectural design addresses comprehensive quality attributes that ensure system effectiveness across diverse operational scenarios and usage patterns. These quality attributes guide design decisions throughout system development while providing measurable criteria for evaluating system success.

Scalability requirements ensure that the system can accommodate growing data volumes, increasing user populations, and expanding analytical capabilities without requiring fundamental architectural modifications. Reliability requirements ensure consistent system operation under diverse conditions including component failures, network disruptions, and unusual load patterns.

Usability requirements ensure that sophisticated system capabilities remain accessible to operational personnel with diverse technical backgrounds and expertise levels. Maintainability requirements ensure that system evolution and enhancement can proceed efficiently while preserving operational stability.

Security requirements ensure appropriate protection of sensitive operational data while maintaining necessary accessibility for authorized personnel. Performance requirements ensure that system responsiveness meets operational expectations across diverse usage patterns and system loads.

Cost-effectiveness requirements ensure that system operation remains economically sustainable while providing necessary capabilities and performance characteristics. The architectural design provides a robust foundation for intelligent water distribution monitoring while establishing clear pathways for continued enhancement and evolution.

3.3 Technical Implementation

This section describes the technical implementation of the intelligent water distribution monitoring system, representing the culmination of the development journey detailed in Section 3.1. The implementation encompasses the complete end-to-end data pipeline, visualization framework, hydraulic simulation integration, machine learning algorithm integration, and the innovative conversational AI interface that transforms how operators interact with complex monitoring infrastructure. Each subsystem is presented with sufficient technical detail to enable replication while highlighting the innovative approaches developed for this research.

The implementation builds upon the lessons learned from the preliminary QGIS-based system (Section 3.1.2) and addresses the limitations identified during the development evolution (Section 3.1.2). The final system architecture represents a unified, cloud-native solution that provides sophisticated monitoring capabilities through an intuitive conversational interface. Figure 3.9 shows an example of an Apache Airflow DAG named `sensor_workflow`, which orchestrates the tasks for sensor data automation and integrating machine learning algorithms developed by a research colleague for leakage detection.

Note on Machine Learning Contributions: The core machine learning algorithms for anomaly detection and localization were developed by a research colleague. This thesis focuses on the novel integration of these algorithms within the comprehensive monitoring system, including data pipeline development, cloud infrastructure implementation, visualization integration, and the innovative conversational AI interface that makes the analytical results accessible through natural language interaction.

3.3.1 Data Acquisition and Processing Infrastructure

The data acquisition subsystem forms the foundation of the intelligent monitoring system, implementing robust mechanisms for collecting, validating, and preprocessing sensor data from distributed monitoring networks. The implementation

addresses the complex challenges of working with heterogeneous sensor networks while maintaining reliability and data quality essential for effective infrastructure monitoring.

The sensor data processing pipeline utilizes CSV file formats provided by an external company as the primary data interchange mechanism, providing standardized sensor data transmission that balances simplicity with comprehensive information content. These CSV files contain sensor measurements along with metadata about measurement conditions, sensor status, and data quality indicators that enable sophisticated validation and analysis processes using Python's pandas library.

The current implementation utilizes manual upload procedures for transferring sensor data files to Google Cloud Storage buckets, providing reliable data ingestion with appropriate quality control and validation procedures. The manual approach enables operators to review data quality and resolve issues before data enters the processing pipeline, ensuring downstream analysis is based on validated measurements.

Figure 3.7 illustrates the complete data acquisition pipeline for incoming CSV files, including transformation and loading processes to BigQuery.

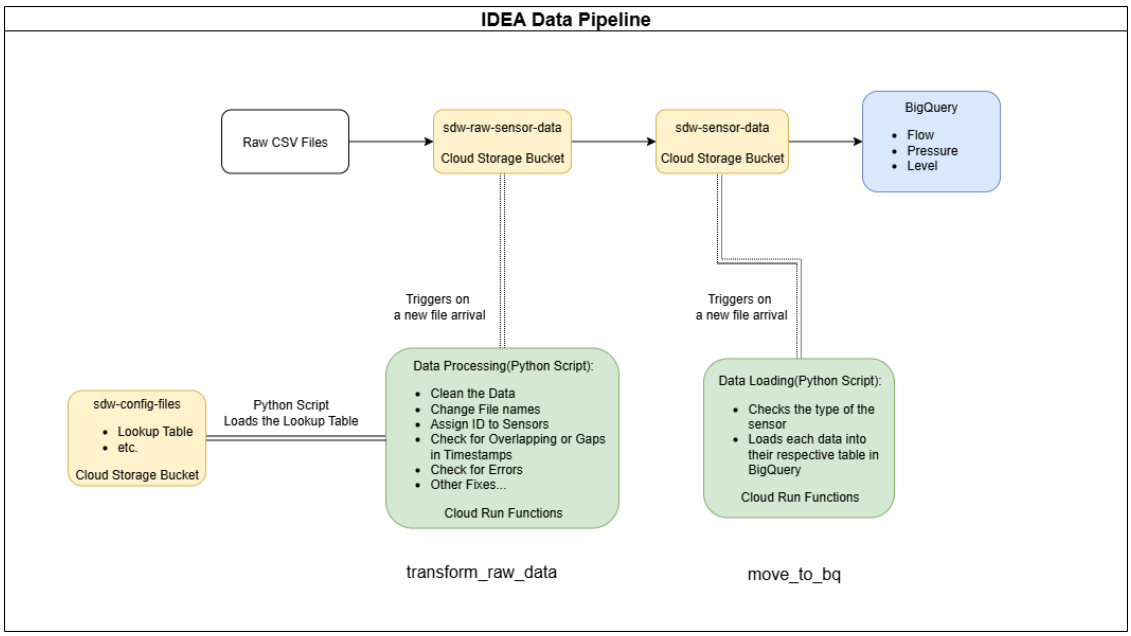

Figure 3.7: Data Acquisition Pipeline, Including Cloud Functions for Data Transformation and Loading

The automated ingestion pipeline implements two specialized Cloud Functions that coordinate data transformation and loading while maintaining appropriate error handling and monitoring capabilities. The `transform_raw_data` function performs comprehensive data validation and transformation operations including file

structure verification against expected schemas, timestamp detection and correction procedures, column header normalization, data type validation and conversion operations, and quality score calculation for downstream filtering.

The `move_to_bq` function implements sophisticated monitoring and loading procedures ensuring reliable data ingestion into BigQuery. This function continuously monitors designated storage locations for new datasets, performs schema validation, executes batch loading operations with appropriate error handling, maintains loading statistics and audit trails, and coordinates with other system components to signal data availability.

3.3.2 Hydraulic Simulation Implementation

The hydraulic simulation subsystem integrates EPANET software with the cloud-based data processing infrastructure to provide comprehensive hydraulic modeling capabilities. The implementation addresses the complex requirements of preparing sensor data for hydraulic simulation while ensuring reliable execution and result processing.

Python scripts running on Google Compute Engine instances perform comprehensive data preparation for EPANET input including correction and extrapolation of missing or anomalous sensor values, sampling interval adjustments to achieve consistent 15-minute intervals, district-level aggregate computations for areas such as Salza and Concentrico districts, and automated simulation execution with result upload to BigQuery. Figure 3.8 illustrates the complete data processing pipeline for hydraulic simulation using EPANET, managed by Apache Airflow.

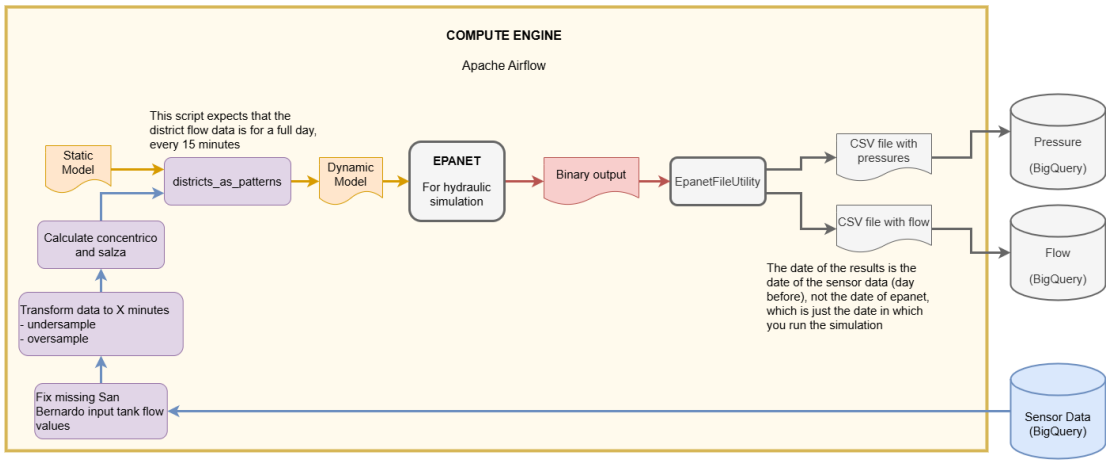

Figure 3.8: Data processing pipeline for hydraulic simulation using EPANET, managed by Apache Airflow

The hydraulic simulation pipeline implements a systematic five-step process. Data preparation includes fixing missing input tank flow values, transforming data

to specified intervals, and calculating district aggregates. Static and dynamic models provide initial district data and generate district patterns respectively. EPANET simulation processes the dynamic model data to produce binary output files. Data extraction uses EpanetFileUtility to convert binary output into CSV files containing pressure and flow data. Finally, data storage saves resulting CSV files to BigQuery with timestamps corresponding to sensor data dates rather than simulation run dates.

3.3.3 Machine Learning-Based Anomaly Detection

The machine learning subsystem integrates anomaly detection algorithms developed by a research colleague within the cloud-based monitoring infrastructure. The contribution of this thesis lies in the comprehensive system integration, including data preprocessing pipelines, automated workflow orchestration, result storage systems, and visualization interfaces that make algorithm outputs accessible to operators.

Integration Infrastructure Implementation: The integration addresses the complex challenges of operationalizing research algorithms within production systems:

- **Data Preprocessing Pipeline**: Automated data preparation workflows that transform raw sensor measurements into formats required by the algorithms, including data cleaning, validation, and feature preparation processes.
- **Cloud Execution Environment**: Scalable compute infrastructure using Google Cloud Platform that ensures reliable algorithm execution under varying operational conditions and data loads.
- **Result Storage and Management**: Comprehensive storage systems using BigQuery that capture algorithm outputs with appropriate schema design for analytical queries and historical analysis.
- **Workflow Orchestration**: Apache Airflow DAGs that coordinate algorithm execution with data processing pipelines, ensuring timely analysis and result availability.

The integration infrastructure includes comprehensive data processing workflows and system integration components:

- **BigQuery Integration:** Scalable storage and analysis of algorithm results using BigQuery, including automated result storage with appropriate schema design, historical trend analysis capabilities, and integration with visualization platforms.

- **PostGIS Spatial Integration:** Integration with PostGIS for spatial analysis workflows that combine algorithm results with network topology information, enabling geographic visualization of detection results and spatial correlation analysis.
- **Workflow Automation:** Apache Airflow orchestration ensures reliable algorithm execution through automated scheduling, comprehensive error handling, and systematic result validation processes.

Future system enhancements include improved cloud infrastructure scaling, enhanced data preprocessing capabilities, and deeper integration with the conversational *AI* interface for natural language querying of algorithm results.

Workflow Orchestration and Automation

To automate the execution of the pipeline, I used Apache Airflow, an open-source platform for programmatically authoring, scheduling, and monitoring workflows. Basically, every task in the Compute Engine instance is orchestrated by Airflow, which ensures that the tasks are executed in the correct order and with the correct dependencies.

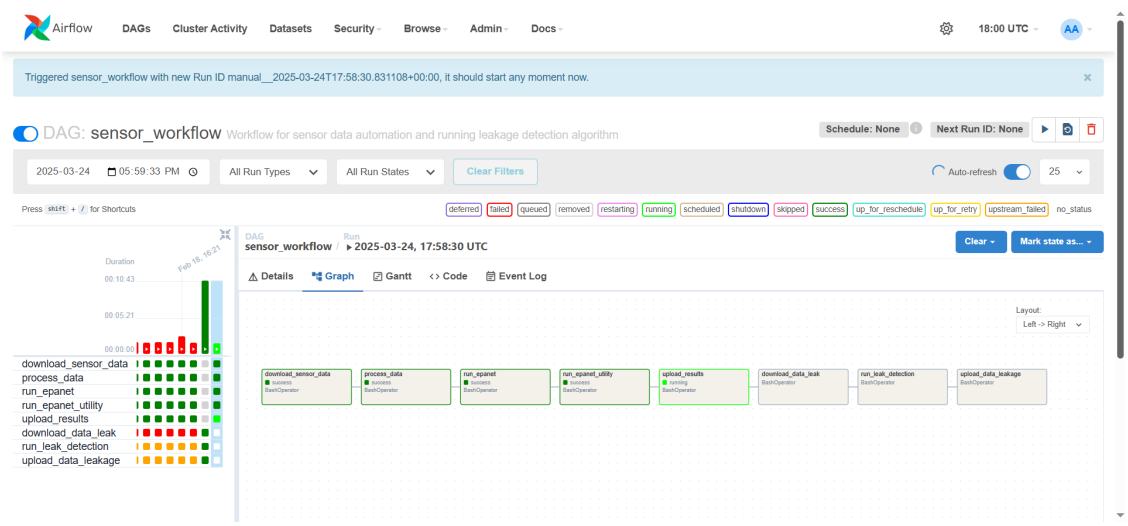

Figure 3.9: Apache Airflow *DAG* for Pipeline Automation

Core tasks in the Directed Acyclic Graph (*DAG*) include:

1. Data ingestion from the database.
2. Data cleaning & transformation (CSV reformat, interpolation, etc.).

3. Generating hydraulic input files and running EPANET.

4. Storing simulation outputs in BigQuery with timestamps.

5. Running *ML* detection/localization algorithms (developed by research colleague), plus updating BigQuery or Postgres with results.

Airflow's user-friendly UI and logging make it simpler to control these interdependent steps. Figure 3.9 shows an Apache Airflow DAG named `sensor_workflow`, which orchestrates the tasks for sensor data automation and integrating machine learning algorithms for leakage detection.

This *DAG* orchestrates a series of tasks for sensor data automation and integrating machine learning algorithms developed by a research colleague for leakage detection. The tasks are represented as nodes in the graph, and their successful execution is indicated by the green status. The tasks include:

- **download_sensor_data**: Downloads sensor data.
- **process_data**: Processes the downloaded data.
- **run_epanet**: Runs the EPANET hydraulic simulation.
- **run_epanet_utility**: Processes the EPANET output files.
- **upload_results**: Uploads the processed results to the database.
- **download_data_leak**: Downloads additional data required for leakage detection.
- **run_leak_detection**: Executes the leakage detection algorithm (developed by research colleague).
- **upload_data_leakage**: Uploads the leakage detection results to the database.

This workflow ensures that sensor data is processed, simulated, and analyzed efficiently, enabling near real-time hydraulic assessments and leak detection in water distribution networks.

Since the cost of running the Compute Engine instance can be high, the instance is automatically started and stopped based on a schedule to optimize costs.
This is achieved by using the Google Cloud Scheduler to trigger the start and stop of the Compute Engine instance at specific times. The codes are written in Python and are executed as Cloud Functions with and *HTTP* trigger, which is called by the Google Cloud Scheduler.

Figure 3.10 shows the Cloud Scheduler configuration for automatically starting the Compute Engine VM with an HTTP trigger every day at 7:00 AM.

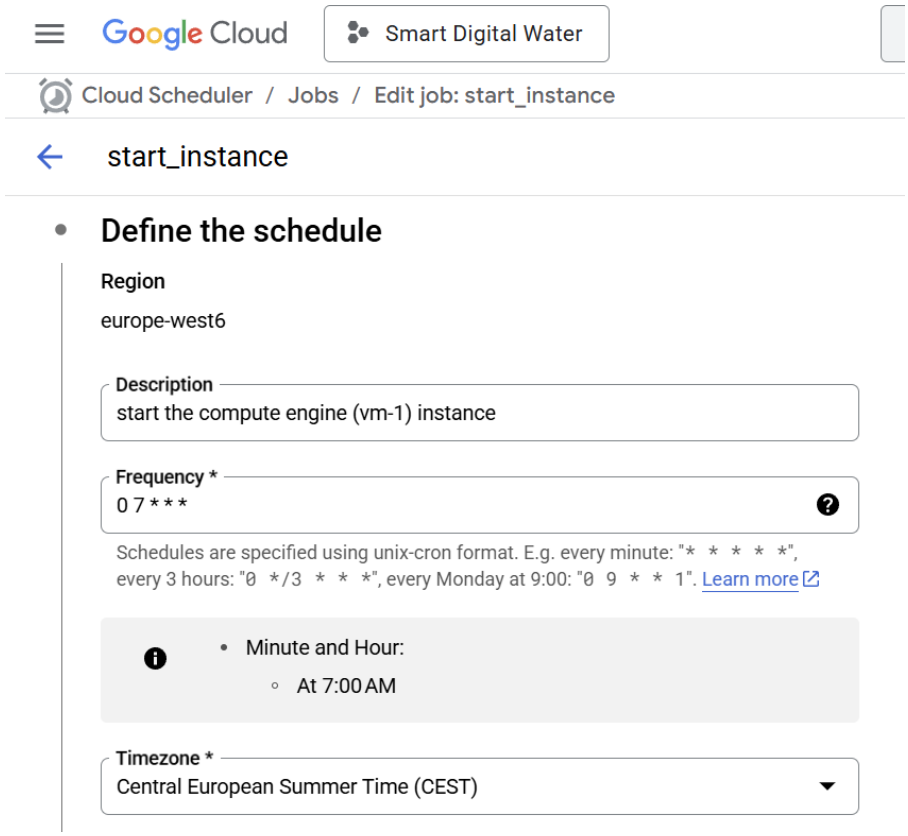

Figure 3.10: Cloud Scheduler Job for automatically starting the Compute Engine VM

3.3.4 Geospatial Data Management and *API* Infrastructure

The geospatial data management subsystem provides comprehensive storage, processing, and delivery capabilities for complex geographic information. The implementation addresses sophisticated challenges of managing multi-layered geospatial datasets while providing high-performance access to diverse client applications.

The system is validated through deployment in the Marene municipal water distribution network, managing comprehensive sensor networks distributed across operational districts. The deployment demonstrates practical effectiveness with flow, pressure, and tank level sensors strategically positioned throughout the network.

The storage infrastructure leverages PostgreSQL with the PostGIS extension to provide advanced geospatial data management capabilities that support sophisticated spatial queries, indexing, and analysis operations. PostGIS provides comprehensive support for geographic data types, spatial indexing algorithms, geometric

operations, coordinate system transformations, and advanced analytical functions that enable complex spatial analysis workflows.

The database design implements normalized schemas that separate geometric data from attribute information while maintaining referential integrity and optimizing query performance. The implementation includes specialized tables for network components with associated geometric representations, sensor installations with precise location data and deployment metadata, district boundaries with hierarchical organizational structures, and historical data with temporal and spatial indexing for efficient time-series analysis.

Spatial indexing strategies optimize query performance for the diverse access patterns required by the monitoring system. The implementation utilizes R-tree indexes for geometric data, compound indexes for temporal-spatial queries, partial indexes for frequently accessed subsets, and clustering strategies that co-locate related geographic features to minimize disk access overhead during complex analytical operations.

The Express.js *API* server provides a high-performance, scalable interface for accessing geospatial data while abstracting the complexity of database interactions from client applications. The server implements comprehensive *REST* endpoints that support diverse query patterns including spatial range queries for map viewport updates, attribute-based filtering for specific network components, temporal queries for historical analysis, and complex analytical operations that combine multiple data sources.

The *API* architecture implements sophisticated data processing pipelines that merge geospatial network topology information with real-time analytical results from BigQuery datasources. This integration enables the dynamic presentation of current system status overlaid on geographic network representations, providing operators with intuitive visual interfaces for understanding complex operational conditions.

The data merging process implements several critical operations including spatial joins that associate analytical results with geographic network components, temporal alignment that ensures consistency between sensor measurements and network topology, attribute enrichment that combines static infrastructure data with dynamic operational metrics, and formatting operations that transform complex datasets into standardized *GeoJSON* representations suitable for web-based visualization clients.

The *GeoJSON* generation process addresses the specific requirements of Grafana's GeoMap panel, which requires precisely formatted geographic data for effective visualization. The implementation ensures compliance with *GeoJSON* specifications while optimizing data structure for performance and visualization effectiveness. This includes coordinate precision optimization for map display scales, attribute organization for effective tooltip and popup displays, geometric simplification for appropriate zoom levels, and metadata inclusion for interactive functionality.

Performance optimization strategies address the challenging requirements of real-time geospatial data delivery while managing the computational overhead of complex spatial operations. The implementation includes comprehensive caching mechanisms for frequently accessed geographic data, query result caching with appropriate invalidation strategies, precomputed aggregations for common analytical operations, and optimized data transfer protocols that minimize bandwidth requirements.

```
async function getNodes() {
  const client = new pg.Client(connectionParams);
  await client.connect();

  try {
    const res = await client.query(`
        SELECT jsonb_build_object(
          'type', 'FeatureCollection',
          'features', jsonb_agg(
            jsonb_build_object(
              'type', 'Feature',
              'geometry', ST_AsGeoJSON(geom)::jsonb,
              'properties', to_jsonb(row) - 'geom'
            )
          )
        ) AS geojson
        FROM (SELECT * FROM public.model_junctions) row;
      `);

    const geoJSON = res.rows[0].geojson;
    return geoJSON;
  } catch (err) {
    console.error('Error executing query', err.stack);
  } finally {
    await client.end();
  }}
```

Listing 3.1: A Function in the Express.js Server to Retrieve the Nodes as GeoJson

The caching system implements multiple layers including in-memory caches for extremely high-frequency access patterns, distributed caches for shared data across multiple server instances, persistent caches for expensive analytical results, and intelligent cache invalidation based on data freshness requirements and update frequencies.

Error handling and reliability mechanisms ensure robust operation under diverse conditions including database connectivity issues, data inconsistency detection and resolution, query timeout handling with appropriate fallback strategies, and graceful degradation capabilities that maintain essential functionality during partial system outages.

The *API* server deployment strategy evaluates multiple hosting options to optimize performance, scalability, and cost considerations. The analysis compares deployment on dedicated Compute Engine instances with serverless platforms like

Cloud Run, considering factors including latency requirements for real-time visualization, scaling characteristics for variable load patterns, cost implications of different usage patterns, and integration capabilities with existing cloud infrastructure.

The serverless deployment approach offers advantages including automatic scaling based on demand, reduced infrastructure management overhead, built-in load balancing and fault tolerance, and cost optimization for variable usage patterns. However, this approach also presents challenges including cold start latency for infrequent requests, resource limitations for computationally intensive operations, and potential vendor lock-in considerations.

The dedicated instance approach provides advantages including predictable performance characteristics, complete control over the runtime environment, optimization capabilities for specific workload patterns, and independence from serverless platform limitations. The trade-offs include increased infrastructure management requirements, manual scaling and load balancing implementation, and fixed cost structures regardless of usage patterns.

The final implementation adopts a hybrid approach that utilizes serverless deployment for development and testing environments while maintaining dedicated instances for production workloads. This strategy optimizes development velocity while ensuring predictable performance for operational requirements.

The real-time visualization integration represents one of the most significant advantages of the geospatial *API* architecture. The system provides dynamic updates to map visualizations that reflect current network conditions, enabling operators to monitor leak probabilities, pressure distributions, flow patterns, and sensor status through intuitive geographic interfaces. This capability addresses previous limitations in static visualization approaches while providing the foundation for advanced analytical workflows and automated alerting systems.

3.3.5 Advanced Visualization System Implementation

The visualization subsystem represents a fundamental component of the intelligent water monitoring infrastructure, providing operators with intuitive, interactive interfaces for exploring complex datasets, monitoring system performance, and identifying potential anomalies or operational issues. The implementation addresses the critical challenges of presenting multi-dimensional temporal and geospatial data in accessible formats while maintaining the analytical depth required for effective infrastructure management.

Building upon the preliminary implementation experiences described in Section 3.1.2, the final visualization system reflects the systematic evolution from the multi-tool approach to unified platform integration. The transition from QGIS Web Client Services and Looker Studio to the current Grafana-based architecture addressed the fundamental limitations identified during the preliminary development phase,

as detailed in Section 3.1.2.

The comprehensive evaluation of limitations identified during the development journey necessitated architectural evolution toward integrated platforms. The transition to Grafana as the primary visualization platform addressed the fragmented user experience, inconsistent data refresh behaviors, limited customization capabilities, and authentication challenges that characterized the preliminary implementation phase. Figure 3.11 shows an example of the Grafana Geomap panel, where network components are colored based on leakage and flow data, providing an intuitive visualization of the system's status.

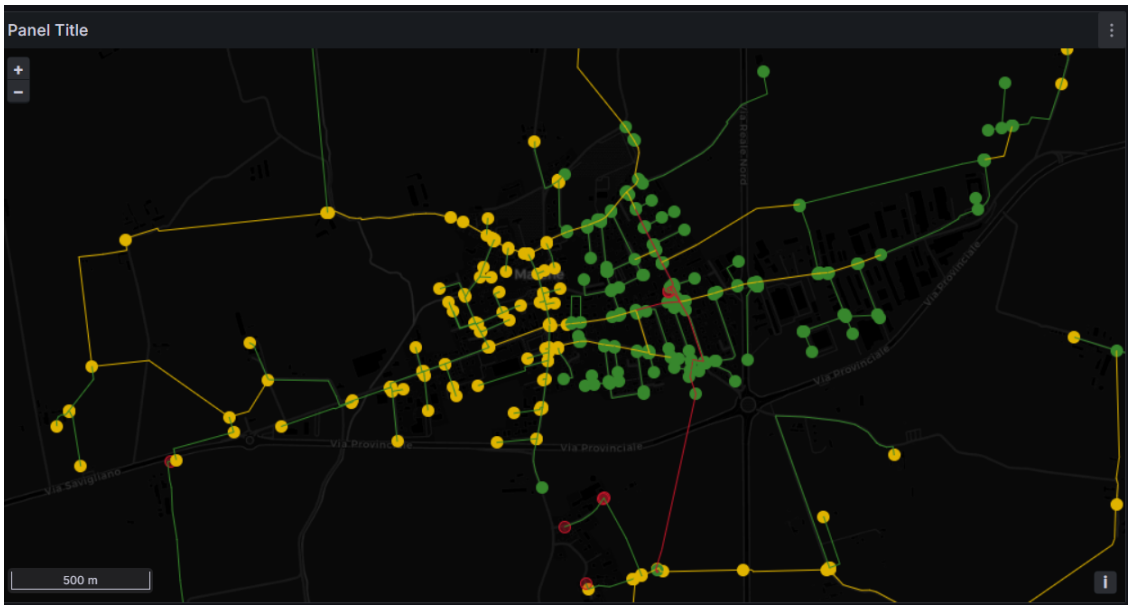

Figure 3.11: Grafana Geomap Panel Showing the Network Components with Rule-Based Coloring Based on Leakage and Flow

To address these limitations systematically, the implementation transitioned to Grafana as the primary visualization platform, leveraging its comprehensive capabilities for both time-series and geospatial data presentation. Grafana's plugin architecture and extensive *API* support enabled the development of sophisticated visualization solutions that could integrate seamlessly with the existing data infrastructure while providing the flexibility required for diverse operational requirements.

The Grafana deployment strategy required careful evaluation of hosting options to balance performance, scalability, and cost considerations. The analysis compared self-hosted deployment on Google Compute Engine instances with managed Grafana Cloud services, evaluating factors including computational resource requirements, maintenance overhead, data transfer costs, security and compliance considerations, and integration capabilities with existing Google Cloud Platform

services.

The self-hosted Compute Engine approach offered advantages in terms of complete control over the deployment environment, customization capabilities for specialized plugins and configurations, direct integration with internal networks and services, and predictable cost structures for high-usage scenarios. However, this approach also required significant administrative overhead, ongoing maintenance responsibilities, and expertise in system administration and security management.

The Grafana Cloud alternative provided advantages including managed infrastructure with automatic updates and security patches, built-in scalability and performance optimization, integrated backup and disaster recovery capabilities, and professional support services. The trade-offs included reduced customization flexibility, potential data transfer costs for large datasets, and dependency on external service availability.

The final implementation adopted a hybrid approach that leverages Grafana Cloud for production deployments while maintaining self-hosted development environments for customization and testing activities. This strategy optimizes the balance between operational efficiency and development flexibility while ensuring appropriate disaster recovery and business continuity capabilities.

The visualization system implements two primary categories of analytical interfaces: time-series visualization panels for temporal data analysis and geospatial visualization dashboards for network topology and spatial analysis. The time-series panels provide comprehensive capabilities for monitoring sensor measurements, simulation results, and anomaly detection outputs across configurable time ranges and aggregation levels.

The time-series implementation includes specialized panels for flow rate monitoring with support for multiple measurement points and comparative analysis, pressure monitoring with configurable alert thresholds and trend analysis capabilities, tank level visualization with capacity indicators and refill scheduling information, and leak detection results presentation with confidence intervals and historical comparison features. These panels incorporate advanced features including dynamic time range selection, automated refresh capabilities, interactive zooming and panning, statistical overlay functions, and export capabilities for reporting and documentation purposes.

The geospatial visualization system leverages Grafana's GeoMap plugin to provide interactive mapping capabilities that integrate network topology data with real-time operational information. The implementation addresses the complex challenges of visualizing multi-layered geospatial information while maintaining performance and usability across diverse datasets and user scenarios.

The GeoMap implementation includes comprehensive support for network infrastructure visualization through node representation with dynamic coloring based on operational parameters, pipe visualization with flow direction indicators and

capacity information, sensor location mapping with status indicators and measurement overlays, district boundary delineation with aggregated statistics and comparative analysis, and leak probability visualization with confidence intervals and historical trend information. The color coding system implements standardized schemes that enable rapid visual assessment of system status while providing detailed information through interactive tooltips and popup displays.

The integration with the Express.js API server enables dynamic data loading and real-time updates for geospatial visualizations. This integration addresses the critical challenge of maintaining current information display while managing the performance implications of frequent data updates. The solution implements intelligent caching mechanisms, differential update procedures, and optimized data transfer protocols that minimize bandwidth requirements while ensuring timely information presentation.

Figure 3.12 demonstrates the final integrated dashboard that combines all visualization components including time-series monitoring panels, geospatial mapping interfaces, and statistical analysis displays in a unified operational interface.

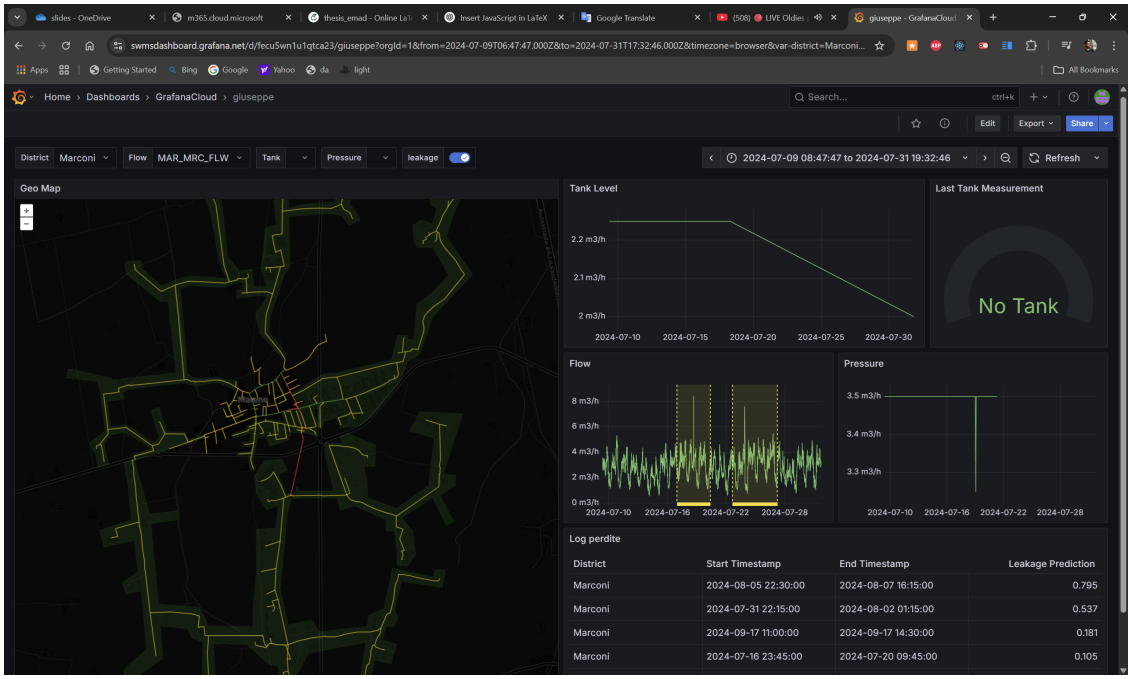

Figure 3.12: Final Grafana Dashboard Including Various Panels for Time Series and Geospatial Data

Future enhancements to the visualization system include integration of advanced alerting capabilities that can automatically notify operators through email, text

messaging, or mobile application notifications based on configurable threshold conditions and anomaly detection results. These alert systems will incorporate sophisticated logic for preventing false positives while ensuring rapid response to genuine operational issues.

Possible advanced visualization features include user-controlled temporal navigation through integrated date picker widgets that enable historical data exploration, JavaScript-based plugins that provide enhanced interactivity and customization capabilities, automated alert integration that provides visual notifications for anomaly detection events, and export functionality that supports report generation and documentation requirements. These enhancements aim to further improve the operational efficiency and user experience of the visualization system, ensuring that operators can access critical information quickly and intuitively while maintaining the analytical depth required for effective infrastructure management.

3.3.6 Future Improvements and Cost Optimizations

Building on the lessons learned from the intelligent interface implementation, several areas for continued development and optimization have been identified:

- **Advanced AI Integration**: Incorporate predictive maintenance models and anomaly detection algorithms that can be accessed through natural language queries, enabling operators to ask questions about future system behavior and potential issues.
- **Mobile Integration**: Develop field-ready applications for maintenance crews that leverage the conversational interface for accessing system data and updating maintenance records while on-site.
- **Expanded Connectivity**: Integrate with IoT sensors and real-time data streams to enable more comprehensive monitoring capabilities accessible through natural language commands.
- **Multi-Network Support**: Scale the architecture to support multiple water distribution networks simultaneously, with the chatbot providing intelligent routing and context management for multi-network operations.
- **Tightening Permissions**: Reduce service account privileges and unify access controls in GCP while maintaining the seamless user experience provided by the intelligent interface.
- **Compute Engine Auto-scaling**: Enhance the existing scheduling with intelligent workload prediction based on chatbot usage patterns and data query complexity.

- **Continuous Deployment for Cloud Functions**: Link GitHub repos to rapidly deploy function updates without manual overhead, including automated testing of chatbot functionality.
- **Machine Learning Integration Enhancements**: Improve system integration capabilities for incorporating enhanced leak-detection algorithms through more sophisticated data preprocessing pipelines, with results accessible through natural language queries about system anomalies and localization results.
- **Enhanced Alert Systems**: Implement intelligent alerting that can be configured and managed through natural language commands, allowing operators to set custom alert conditions using conversational interfaces.

3.3.7 Summary of Contributions and Reflections

By integrating data acquisition, processing, orchestration, simulation, and visualization, our pipeline addresses near real-time monitoring of hydraulic conditions in water networks. Practical tasks accomplished include:

- **Automated Ingestion**: Validating, cleaning, and loading raw CSVs into BigQuery.
- **EPANET Simulation Pipelines**: Generating `.inp` files on the fly and storing simulation results in BigQuery.
- **ML-Driven Leak Detection**: Capturing anomalies or suspected leaks for further investigation.
- **Geospatial Integration**: Storing nodes and pipes in PostGIS, serving them over an Express.js API for dashboards.
- **Enhanced Visualization**: Leveraging Grafana's Geomap plugin or custom map clients.

Combined, these developments translate real-world sensor data into actionable insights for operators, while ongoing improvements—such as Leaflet.js integration, expanded alerting, or refined compute workflows—promise an even more robust solution in the coming iterations.

3.3.8 Intelligent Dashboard Interface Development

With robust Grafana-based visualization successfully established, development focused on creating an innovative conversational assistant using cutting-edge *AI* technologies. The iterative development approach progressed through multiple phases, evolving from basic feasibility demonstration to comprehensive web-based intelligent interface capabilities.

Problem Identification and Requirements Analysis

The initial implementation of the Grafana-based visualization system, while functional, revealed several operational bottlenecks that warranted investigation. Through systematic observation of user interactions and analysis of system logs, I identified recurring patterns of inefficiency. Operators frequently struggled with the manual process of navigating through Grafana's interface hierarchy to access specific dashboards or modify visualization parameters. The GeoJSON layer updates, which were central to displaying real-time network status, exhibited inconsistent refresh behavior, often requiring manual browser refreshes to display updated data. Additionally, accessing historical data for comparative analysis required operators to manually construct time ranges and filter parameters, a process that was both time-consuming and error-prone.

These observations led me to formulate a comprehensive set of requirements for an enhanced interface system. The system needed to provide natural language processing capabilities to interpret operator commands in conversational form, maintain direct integration with the existing Grafana API to ensure compatibility with the established visualization infrastructure, offer transparent data access to the underlying BigQuery datasources while presenting results in an accessible format, provide real-time feedback and confirmation mechanisms to ensure data integrity, and support extensible functionality to accommodate future operational needs.

The intelligent chatbot interface was designed to address these requirements through conversational interaction capabilities, as demonstrated in Figure 3.13 which shows the system's ability to explain its current functionalities when queried by operators.

To validate these requirements, I conducted a preliminary feasibility study examining the technical constraints and opportunities within the existing architecture. This investigation revealed that Grafana's REST API provided comprehensive programmatic access to dashboard management functions, BigQuery's SQL interface could be abstracted through natural language processing techniques, and modern large language models offered sufficient context understanding for domain-specific applications.

Technology Selection and Architecture Design

Technology selection decisions required careful evaluation of capabilities, integration complexity, and long-term maintenance considerations. The command-line interface provided initial validation while revealing critical *LLM* limitations for precise operations. This informed the hybrid architecture where *LLMs* handle natural language understanding while Python functions manage precise dashboard operations.

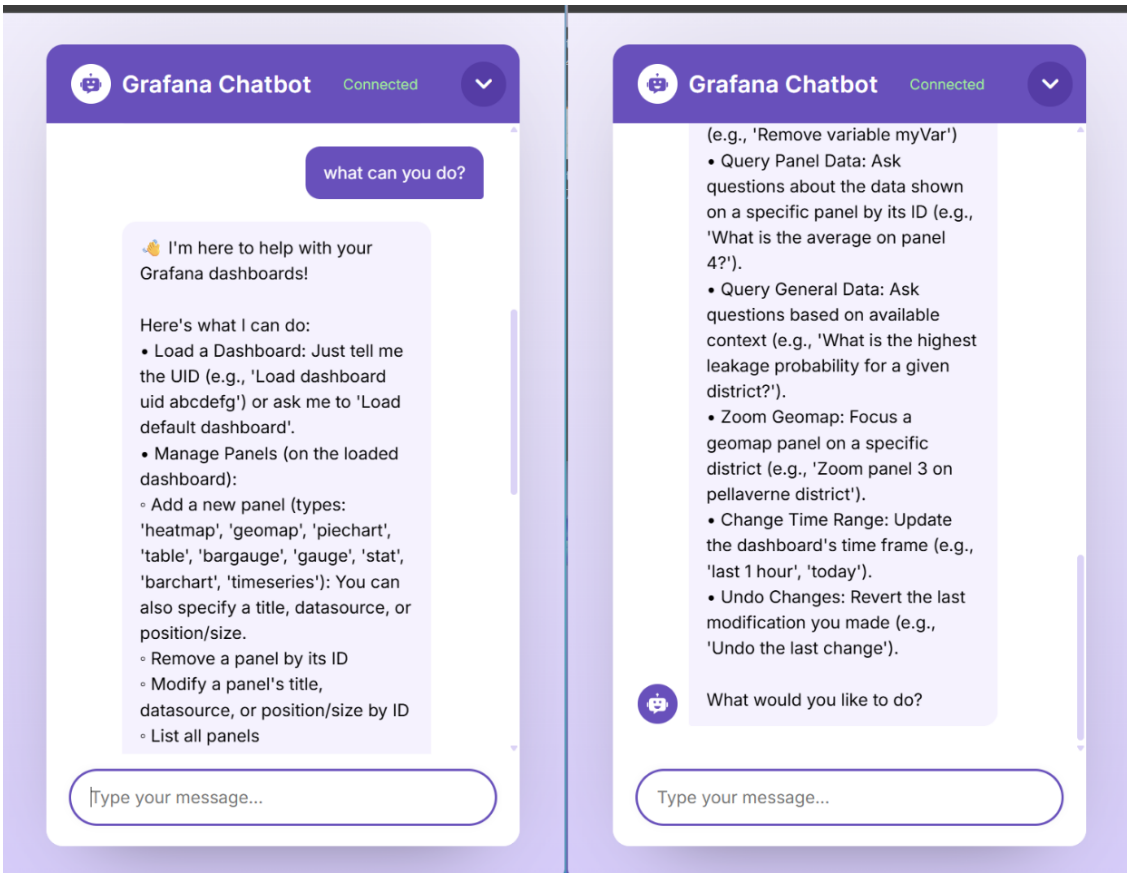

Figure 3.13: Asking the Chatbot About Its Current Functionalities

For the backend service architecture, I chose FastAPI as the primary web framework due to its native support for asynchronous operations, automatic API documentation generation, and robust type validation capabilities. The transition from command-line to web-based system required the implementation of WebSocket connectivity, chosen over traditional *HTTP* connections to enable real-time bidirectional communication and maintain persistent conversation state essential for complex multi-turn conversations.

The natural language processing capabilities were implemented using LangChain, a framework specifically designed for building applications with large language models. LangChain's modular architecture allowed for the creation of specialized chains for different types of operations, including dashboard management, data querying, and system administration tasks. The framework's built-in support for prompt templating and response parsing significantly accelerated the development of domain-specific language understanding capabilities.

For workflow orchestration and state management, I integrated LangGraph, which extends LangChain's capabilities to support complex, stateful conversations

with multiple decision points and confirmation requirements. This was essential for implementing the confirmation workflows that ensure operators explicitly approve dashboard modifications before they are applied to the production Grafana instance.

The choice of Google's Gemini 2.0 Flash as the primary large language model was based on several technical and practical considerations. Gemini 2.0 Flash demonstrated superior performance in understanding technical terminology related to water distribution systems and infrastructure monitoring. Its function calling capabilities enabled precise parameter extraction from natural language commands, while its context window size was sufficient to maintain conversation history and system state information across extended interaction sessions.

For the frontend implementation, I selected React as the primary framework due to its component-based architecture and extensive ecosystem of supporting libraries. The real-time communication requirements were addressed through WebSocket integration, enabling instant feedback and streaming responses for long-running operations such as complex data queries or dashboard modifications.

System Architecture and Component Integration

The intelligent dashboard interface follows a distributed architecture pattern that maintains clear separation of concerns while enabling seamless integration with the existing infrastructure components. The system consists of four primary architectural layers: the presentation layer implemented as a React based web application built with Vite, the *API* gateway layer built on FastAPI providing *REST* and WebSocket endpoints, the intelligence layer incorporating LangChain and LangGraph for natural language processing and workflow management, and the integration layer providing secure connections to external services including Grafana, BigQuery, and geospatial data services.

The presentation layer was designed with a focus on conversational user experience, featuring a chat-like interface that presents system responses in a structured format while maintaining the familiar paradigms of messaging applications. The interface incorporates specialized components for displaying tabular data from BigQuery queries, rendering dashboard thumbnails and previews, presenting confirmation dialogs for destructive operations, and providing visual indicators for system status and operation progress.

The API gateway layer serves as the primary orchestration point for all system operations, implemented in `api_server.py` using FastAPI with WebSocket support for real-time bidirectional communication. The core chatbot logic is centralized in `chatbot_core.py` which contains the main `Chatbot` class responsible for LangGraph workflow compilation and natural language processing orchestration. The system maintains persistent WebSocket connections for each user session, managing

individual `DashboardState` instances per connection to enable concurrent multi-user access. The gateway implements comprehensive session management including automatic connection state initialization with default dashboard loading, real-time message processing and response delivery, connection health monitoring with automatic cleanup, and error propagation with graceful degradation. The FastAPI framework's dependency injection system enabled the implementation of modular service components that could be easily tested and maintained.

The intelligence layer represents the core innovation of the system, implementing sophisticated natural language understanding capabilities specifically tailored to the domain of infrastructure monitoring and dashboard management. This layer incorporates multiple specialized components including intent recognition modules that classify user commands into operational categories, parameter extraction services that identify specific values and constraints from natural language input, validation engines that verify the feasibility and safety of requested operations, and execution orchestrators that coordinate the sequence of API calls required to fulfill complex requests.

The integration layer provides secure and efficient connectivity to external services while implementing appropriate retry logic, error handling, and circuit breaker patterns to ensure system resilience. Each external service integration is encapsulated in specialized adapter classes that abstract the complexity of API interactions and provide consistent interfaces for the higher-level system components.

The system's dashboard modification capabilities demonstrate the practical effectiveness of the natural language interface, as shown in Figure 3.14 which illustrates the successful addition of a panel to the Grafana dashboard through conversational commands.

Natural Language Processing Implementation

The development of natural language processing capabilities required extensive investigation into the specific linguistic patterns and terminology used by operators in the water distribution monitoring domain. I began this process by analyzing historical support requests, operator feedback, and common dashboard usage patterns to identify the most frequent types of commands and queries that the system would need to handle.

The initial implementation focused on a rule-based approach using regular expressions and keyword matching to identify basic command patterns. However, this approach quickly proved insufficient for handling the natural variation in human language and the complex parameter relationships inherent in dashboard management operations.

To address these limitations, I transitioned to a large language model-based approach that could understand context, resolve ambiguities, and extract multiple parameters from complex natural language inputs. The implementation leveraged

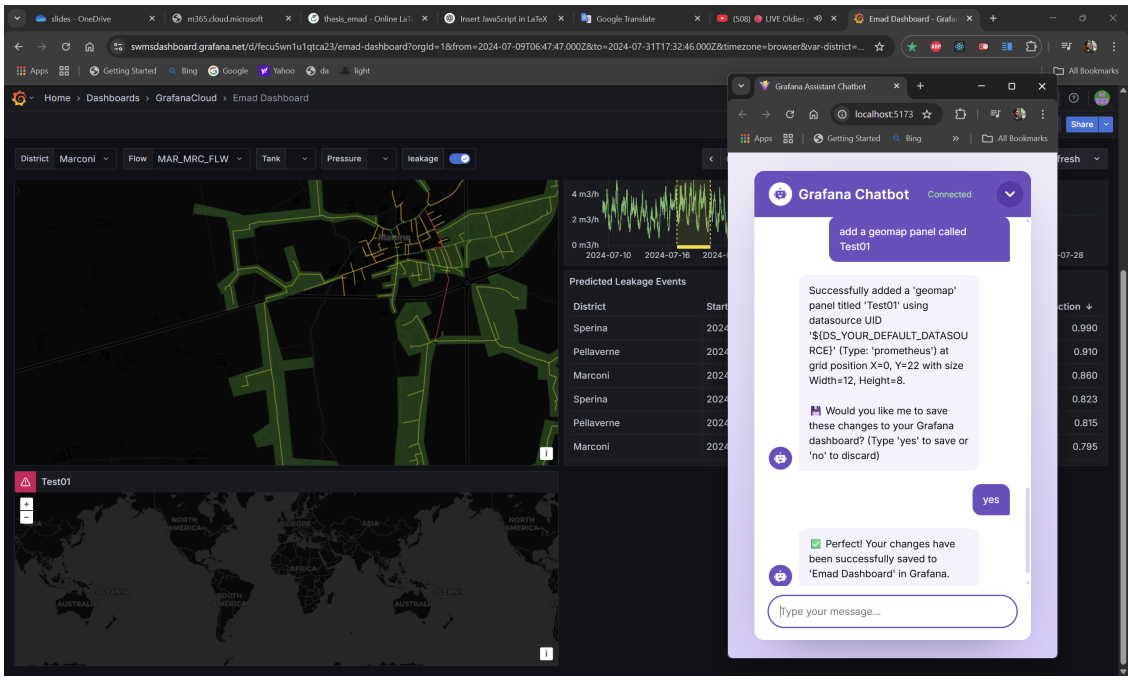

Figure 3.14: Successfully Adding a Panel to the Grafana Dashboard by using the Chatbot

LangChain's prompt engineering capabilities to create specialized templates for different types of operations.

For dashboard management operations, I developed prompt templates that could interpret commands related to loading existing dashboards, creating new dashboard configurations, modifying panel arrangements, adjusting time ranges, and managing dashboard variables. The prompts were carefully crafted to include sufficient context about the available dashboards, panel types, and configuration options while maintaining clarity about the expected response format.

The system's contextual understanding capabilities enable operators to ask sophisticated questions about sensor information and measurement data available on the dashboard, as demonstrated in Figure 3.15 which shows an example of querying the chatbot about water-related sensor measurements including sensor locations, measurement types, and associated metadata.

Data querying capabilities required a different approach, as the system needed to translate natural language questions about sensor data, simulation results, and system performance into valid *SQL* queries against the BigQuery datasources. I implemented a multi-stage process that first identifies the data entities and relationships mentioned in the user's query, then maps these entities to the appropriate database tables and columns, constructs a preliminary *SQL* query based on the

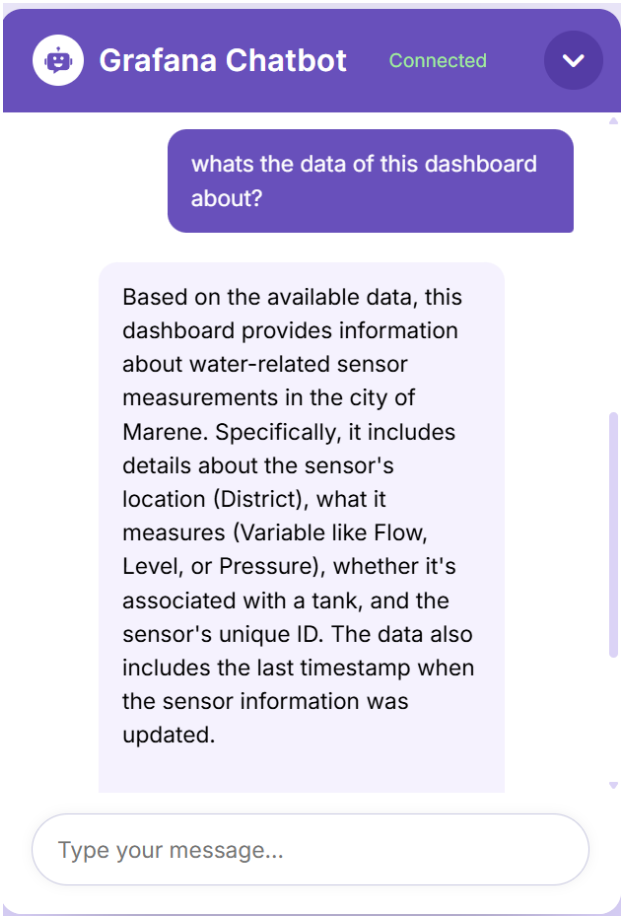

Figure 3.15: Example of Asking the Chatbot about Sensor Information and Measurements Available on the Dashboard

identified relationships, validates the query structure and applies necessary security constraints, and finally executes the query while providing transparency about the generated *SQL* code.

The parameter extraction process was implemented using LangChain's function calling capabilities, which allow the large language model to identify structured data within natural language input and format it according to predefined schemas. This approach enabled the system to handle complex commands with multiple parameters while maintaining type safety and validation.

Workflow Management and State Handling

One of the most critical aspects of the intelligent interface implementation was the development of robust workflow management capabilities that could handle complex, multi-step operations while maintaining system consistency and providing

appropriate safeguards against unintended modifications. The LangGraph framework provided the foundation for implementing stateful conversation flows with explicit decision points and confirmation requirements.

The workflow management system implements a specialized node-based architecture where each node represents a specific type of operation or decision point in the conversation flow. The system organizes workflow nodes into functionally distinct modules: `dashboard_nodes.py` handles all dashboard loading and panel management operations, `data_query_nodes.py` manages the complete data querying workflow including context loading and *SQL* generation, `interpretation_nodes.py` processes natural language input and extracts user intent and parameters, `variable nodes.py` provides dashboard variable management capabilities, `geomap_nodes.py` handles geospatial operations including district zooming with precise centroid calculations, and `utility_nodes.py` contains support functions for error handling, help messages, save confirmation workflows, and undo functionality. Each node receives and modifies a shared `DashboardState` TypedDict that maintains conversation context, current dashboard state, user parameters, and operational metadata throughout the workflow execution.

The state management implementation maintains conversation context across multiple interaction rounds, enabling the system to handle follow-up questions, clarification requests, and incremental modifications to complex operations. The state includes the current conversation history, extracted parameters and their validation status, pending operations awaiting confirmation, system status and availability information, and user preferences and authorization levels.

Error handling and recovery mechanisms were implemented at multiple levels within the workflow system. At the individual node level, each operation includes comprehensive error detection and reporting capabilities. At the workflow level, the system can gracefully handle partial failures and provide users with clear information about what operations succeeded and what requires attention. At the system level, circuit breaker patterns prevent cascading failures and ensure that temporary external service outages do not compromise the overall system functionality. The system also implements undo functionality that leverages Grafana's version history to restore dashboards to their state before the most recent chatbot-applied changes, providing an additional safety mechanism for operators.

Integration with Existing Infrastructure

The integration of the intelligent interface with the existing monitoring infrastructure required careful consideration of API compatibility, security requirements, and performance implications. Each external service integration was implemented as a separate module with comprehensive error handling and retry logic.

The Grafana integration leverages the platform's comprehensive *REST API* to

provide programmatic access to all dashboard management functions. I implemented specialized adapter classes for different types of Grafana operations including dashboard *CRUD* operations with proper *JSON* schema validation, panel management with support for all visualization types used in the monitoring system, user and permission management to ensure appropriate access controls, datasource configuration management, and variable and template management for dynamic dashboard behavior.

Integration with the external geospatial data service enables the chatbot to access district boundary information and coordinate mapping operations. This integration supports district boundary queries for map zoom operations through a configurable GeoJSON webserver, sensor location data for spatial analysis, network topology information for hydraulic calculations, and real-time status updates for system monitoring.

User Interface Development and Experience Design

The development of the conversational user interface required extensive research into best practices for chat-based applications while addressing the specific needs of technical operators working with complex monitoring systems. The interface design process began with the creation of user personas and usage scenarios based on observed operator behavior and feedback from the existing system.

The React based frontend implementation incorporates several specialized components designed to handle the unique requirements of technical conversations. The message display system supports rich content including formatted tables for query results, embedded dashboard previews, syntax-highlighted *SQL* code, interactive confirmation dialogs, and progress indicators for long-running operations.

The system's analytical capabilities enable sophisticated queries about operational data and anomaly detection results, as exemplified in Figure 3.16 which demonstrates querying the chatbot about the highest registered leakage probability in the Marconi district.

The real-time communication implementation uses WebSocket connections to provide immediate feedback and streaming responses. This enables the system to show typing indicators during *LLM* processing, stream partial results for long-running queries, provide real-time status updates during dashboard modifications, and maintain connection health monitoring with automatic reconnection.

Iterative Development and Feature Enhancement

The development of the intelligent interface followed a systematic iterative approach that progressed through distinct phases, each building upon the previous implementation while adding increasingly sophisticated capabilities. The development journey began with a command-line interface proof of concept and evolved

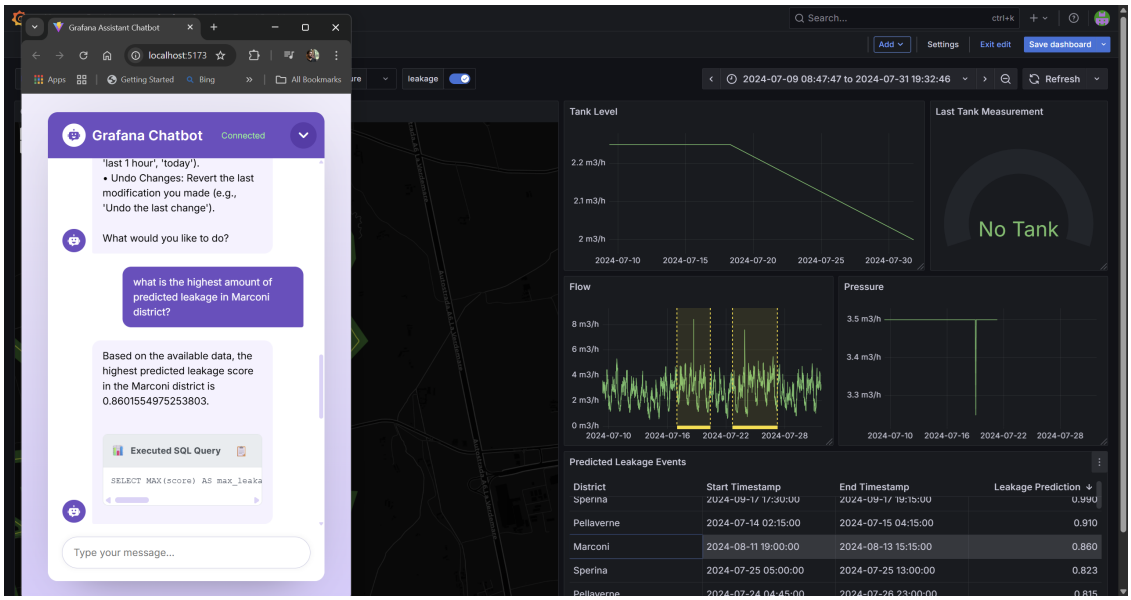

Figure 3.16: Asking the Chatbot about the Highest Registered Leakage Probability on Marconi District

into a comprehensive web-based conversational system.

The initial implementation started with a CLI version (`cli-version` branch) that provided basic functionality for adding and removing panels on a local Grafana dashboard JSON file. This foundational phase established the core concept of programmatic dashboard manipulation through natural language commands and validated the feasibility of using large language models for dashboard management tasks. The CLI implementation served as a testbed for developing prompt templates and understanding the complexity of translating natural language instructions into precise dashboard operations.

The first major enhancement (`cli-version` continued development) expanded the CLI capabilities to include panel modification and time frame changes. This phase required developing more sophisticated understanding of Grafana's panel configuration schemas and implementing logic to handle temporal parameter extraction from natural language inputs. The enhanced CLI version demonstrated the potential for complex dashboard operations while revealing the limitations of file-based dashboard management.

The second major enhancement (`feat/api-server` branch) marked the transition from CLI to web-based architecture with the implementation of the API server and comprehensive backend development. This phase involved integrating LangChain for natural language processing, implementing FastAPI for WebSocket communication, and establishing connections to the Grafana API for real-time dashboard management. The backend architecture provided the foundation for

stateful conversations and real-time dashboard operations while maintaining robust error handling and session management.

The third major enhancement (`feat/front-end` branch) focused on developing the React-based frontend and integrating it with the backend system. This phase involved creating the conversational user interface, implementing WebSocket communication for real-time interactions, and developing specialized components for displaying technical data in chat contexts. The frontend integration transformed the system from a command-line tool into an accessible web application suitable for operational use.

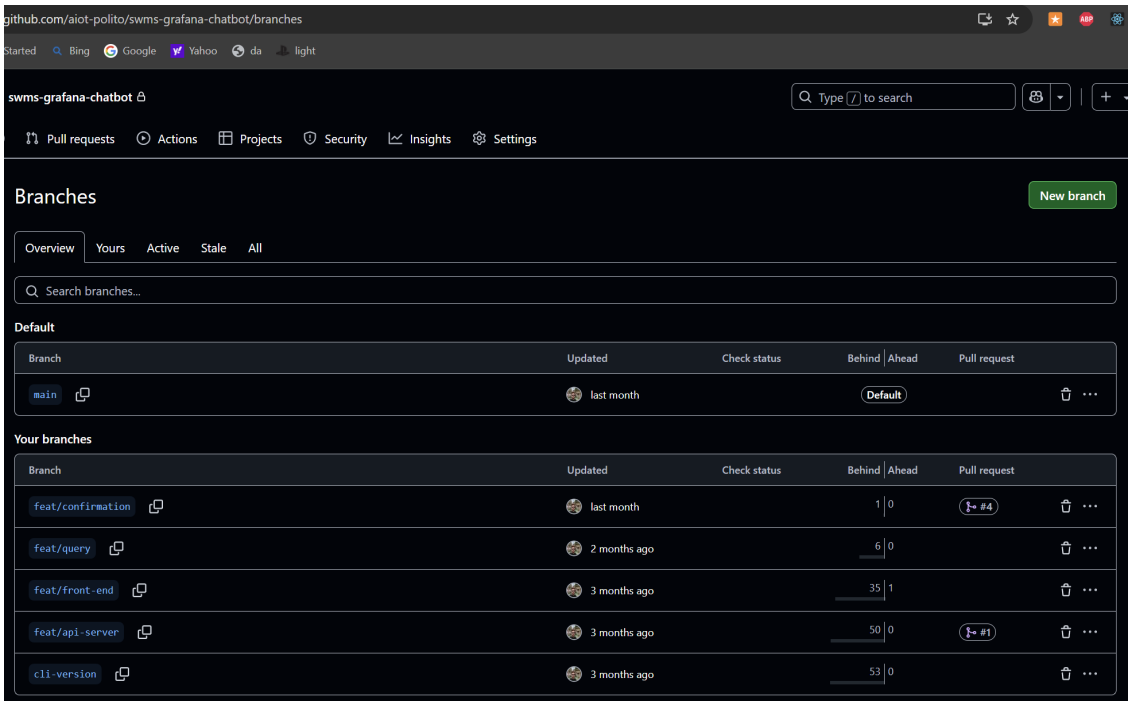

Figure 3.17: Github Branches Representing the Iterative Development of the Chatbot

The fourth major enhancement (`feat/query` branch) introduced sophisticated data querying capabilities that could translate natural language questions into valid SQL queries against BigQuery datasources. This phase required developing multi-stage query processing, implementing parameter extraction for complex analytical requirements, and providing transparency through syntax-highlighted SQL display. The data querying functionality significantly expanded the system's analytical capabilities beyond dashboard management.

The most recent and advanced enhancements (`feat/confirmation` branch) implemented critical safety and user experience features including confirmation workflows for dashboard modifications and undo functionality leveraging Grafana's versioning system. These features emerged from user feedback indicating the need

for safeguards against accidental changes to production dashboards. The confirmation system requires explicit user approval for all destructive operations, while the undo functionality provides an additional safety mechanism by utilizing Grafana's version history to restore dashboards to their previous state.

The systematic approach to feature development is reflected in the project's version control history, as shown in Figure 3.17 which illustrates the GitHub branches representing the iterative development phases.

The iterative development approach has proven essential for managing the complexity of the system while ensuring that new features integrate seamlessly with existing functionality. The modular architecture enabled rapid prototyping and testing of new capabilities while maintaining system stability and reliability.

This comprehensive development process resulted in a sophisticated intelligent interface that significantly enhances the usability and accessibility of the water distribution monitoring system. The system demonstrates the potential for conversational AI to bridge the gap between complex technical infrastructure and operational efficiency, providing a foundation for future enhancements and similar applications in other infrastructure monitoring domains.

Chapter 4

Implementation Results and System Validation

This chapter presents the implementation and validation results of the intelligent cloud-based water distribution monitoring system with conversational AI capabilities, evaluated through the Marene water distribution network case study.

4.1 Case Study: Marene Water Distribution Network

The Marene water distribution network case study provides a comprehensive validation environment for the intelligent monitoring system, demonstrating practical applicability in real-world municipal water infrastructure management. The case study involves multiple stakeholders and existing systems that required careful integration and validation.

4.1.1 Network Management and Operational Context

The Marene water distribution network is managed by AlpiAcque [30], a water utility company responsible for water supply and distribution across multiple municipalities in the region. AlpiAcque's operational requirements include efficient network monitoring, leak detection, pressure management, and ensuring reliable water supply to all districts within their service area. The collaboration with AlpiAcque provided essential domain expertise and realistic operational constraints that shaped the system development and validation process.

The conversational AI interface was developed as an enhancement to the core monitoring system requirements, providing an innovative approach to improve operational efficiency and user interaction with the monitoring platform. While not

part of the original specifications, this component demonstrated significant potential for enhancing operational workflows and reducing the technical barriers for system interaction.

4.1.2 Geographic and Infrastructure Organization

The Marene municipal water distribution network is strategically organized into distinct operational districts that facilitate management and monitoring activities. The primary districts include:

- **Ponte District**
- **Via Torino District**
- **San Bernardo District**
- **Salza District**
- **Concentrico District**
- **Rame District**
- **Marconi District**
- **Sperina District**
- **Pellaverne District**

This district-based organization enables targeted monitoring, maintenance planning, and operational decision-making while providing a natural framework for the conversational AI interface to understand and respond to location-specific queries.

4.1.3 Existing Information Systems Integration

The system validation required integration with multiple existing platforms that provide essential operational data and functionality for the Marene network.

GIS Platform Integration

The network topology and geographic information is managed through the GISMaster platform, developed by Technical Design, which serves as the authoritative source for network component locations, pipe specifications, and infrastructure metadata. The GISMaster platform provides comprehensive mapping capabilities and maintains detailed records of network topology, pipe materials and dimensions,

valve and fitting locations, service connection details, and historical infrastructure modifications.

For the intelligent monitoring system implementation, the GISMaster platform data was exported and transformed into standardized GeoJSON formats to enable integration with the web-based visualization components. This export process preserved essential topology information while optimizing data structure for real-time visualization and conversational interface operations.

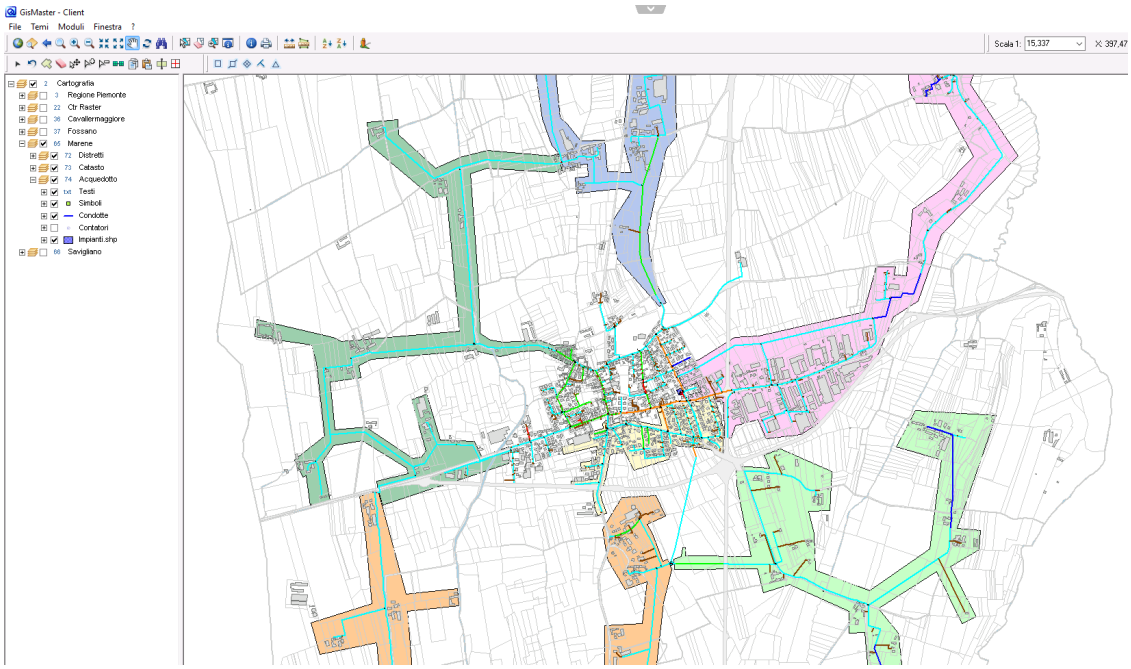

Figure 4.1: GIS Platform GISMaster provided by Technical Design [31].

Figure 4.1 shows the GISMaster platform interface displaying the Marene network topology and infrastructure details. The platform provides a topographical view of the Marene Water Distribution System, organizing the data into several groups and layers including the elements of the WDS itself (pipes, valves, pumps, tanks, etc.), information about the town such as buildings and streets, as well as the WDS districts. The comprehensive GIS visualization provides a detailed topographical representation of the water distribution system, with organized data layers that include network components, municipal infrastructure, and district boundaries. This multi-layered approach enables comprehensive spatial analysis and supports the intelligent monitoring system's geospatial capabilities.

SCADA System Integration

Operational sensor data for the Marene network is collected and managed through a SCADA (Supervisory Control and Data Acquisition) system operated by IDEA[32].

This SCADA system provides essential real-time and historical data collection capabilities including automated daily data collection from distributed sensors, basic visualization and trending capabilities, alarm management for out-of-range conditions, and data archival for regulatory compliance and operational analysis.

The SCADA system collects data from flow sensors, pressure monitoring points, tank level indicators, and valve position sensors distributed throughout the network. Data collection occurs at 15-minute intervals, providing sufficient temporal resolution for both operational monitoring and analytical applications.

While the SCADA system provides basic data visualization capabilities, it lacks advanced analytical features such as automated anomaly detection, predictive analysis capabilities, integration with GIS mapping systems, and sophisticated data querying interfaces. These limitations provided the motivation for developing the intelligent monitoring system that could leverage the existing SCADA data while providing enhanced analytical and visualization capabilities.

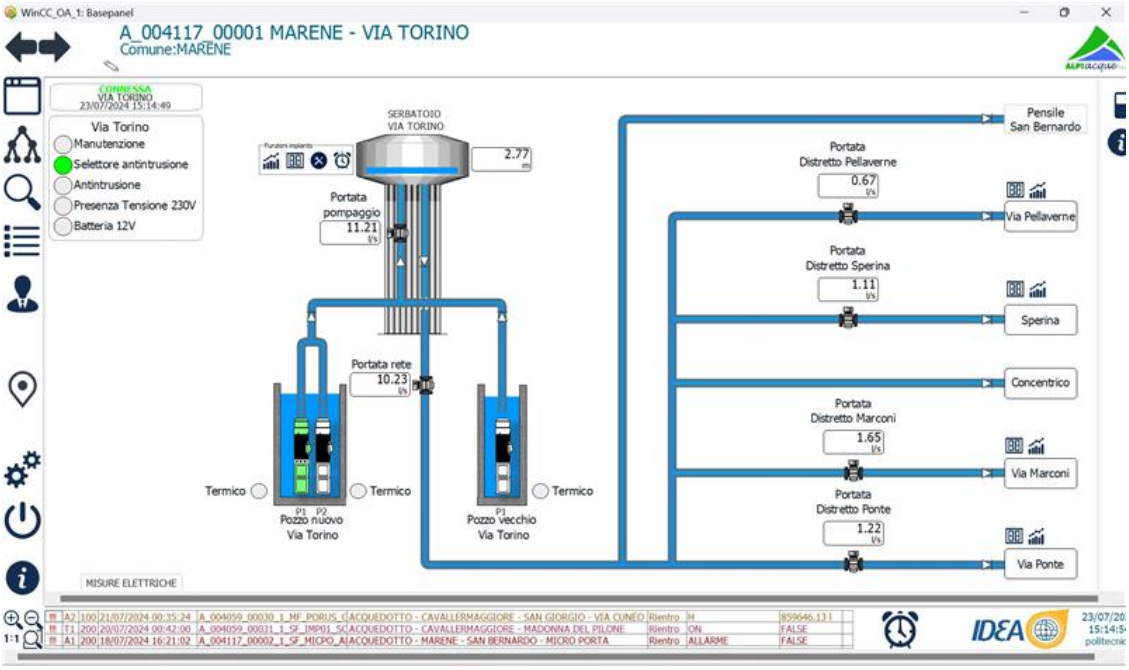

Figure 4.2: SCADA platform provided by IDEA [32].

Figure 4.2 demonstrates the SCADA system interface showing a schematic view of the Via Torino district in the Marene Water Distribution System. The interface displays the main tank in Marene, two well pumps, connections to the Via Torino district, and their respective flow sensors, providing essential operational monitoring capabilities. The visualization provides operators with a comprehensive view of the system topology, including the main storage tank, well pump locations, district connections, and flow sensor positions throughout the network.

4.1.4 Historical Data and Algorithm Validation

The Marene network has comprehensive historical records of leak incidents, repairs, and maintenance activities that provided valuable validation data for the machine learning algorithms integrated within the monitoring system. This historical dataset includes documented leak locations and times, repair records with cost and duration information, pressure anomaly reports, and customer complaint records related to service interruptions.

While the detailed analysis of leak detection algorithm performance falls outside the scope of this thesis, the availability of this historical validation dataset enabled the research colleague developing the machine learning algorithms to verify detection accuracy and calibrate algorithmic parameters for the specific characteristics of the Marene network.

4.1.5 Stakeholder Collaboration and System Validation

The development and validation of the intelligent monitoring system involved extensive collaboration with industry partners and operational stakeholders to ensure practical utility and operational relevance. During the internship period and throughout the thesis development, regular collaboration meetings were conducted with Tesisquare [33], the technology partner responsible for system integration and deployment support, and with AlpiAcque technical staff for operational validation.

The collaboration with Tesisquare provided essential feedback on technical architecture decisions, user interface design choices, operational workflow integration, and system scalability requirements. Tesisquare's expertise in water utility technology implementations provided valuable insights into practical deployment challenges and helped guide architectural decisions that would facilitate real-world adoption and maintenance.

4.1.6 Network Infrastructure and Data Integration

The comprehensive sensor coverage across the Marene districts provides the data foundation for the intelligent monitoring system. Flow, pressure, and tank level sensors positioned throughout Ponte, Via Torino, San Bernardo, Salza, and Concentrico districts deliver measurements at 15-minute intervals, ensuring adequate temporal resolution for both real-time monitoring and historical analysis.

District-specific GeoJSON files, derived from the GISMaster platform, provide precise geographic boundaries that enable sophisticated spatial analysis and intuitive visualization within the Grafana-based dashboard system. The conversational AI interface successfully demonstrates natural language interaction capabilities through district-specific queries and geospatial navigation commands, enabling

operators to quickly focus on specific geographic areas through simple conversational requests.

The integration of sensor data, geographic information, and conversational AI capabilities creates a comprehensive monitoring environment that significantly enhances operational efficiency compared to traditional SCADA-based approaches.

4.2 Visual Dashboard System Results

The implemented system provides comprehensive visualization capabilities through Grafana dashboards that effectively display real-time and historical water distribution data. The visual interface demonstrates practical utility for operational monitoring and decision-making.

4.2.1 Geographic Visualization and District Mapping

The system successfully integrates geospatial data visualization through interactive maps that display real-time network status across different districts. The geomap panels provide comprehensive geographic context for monitoring operations.

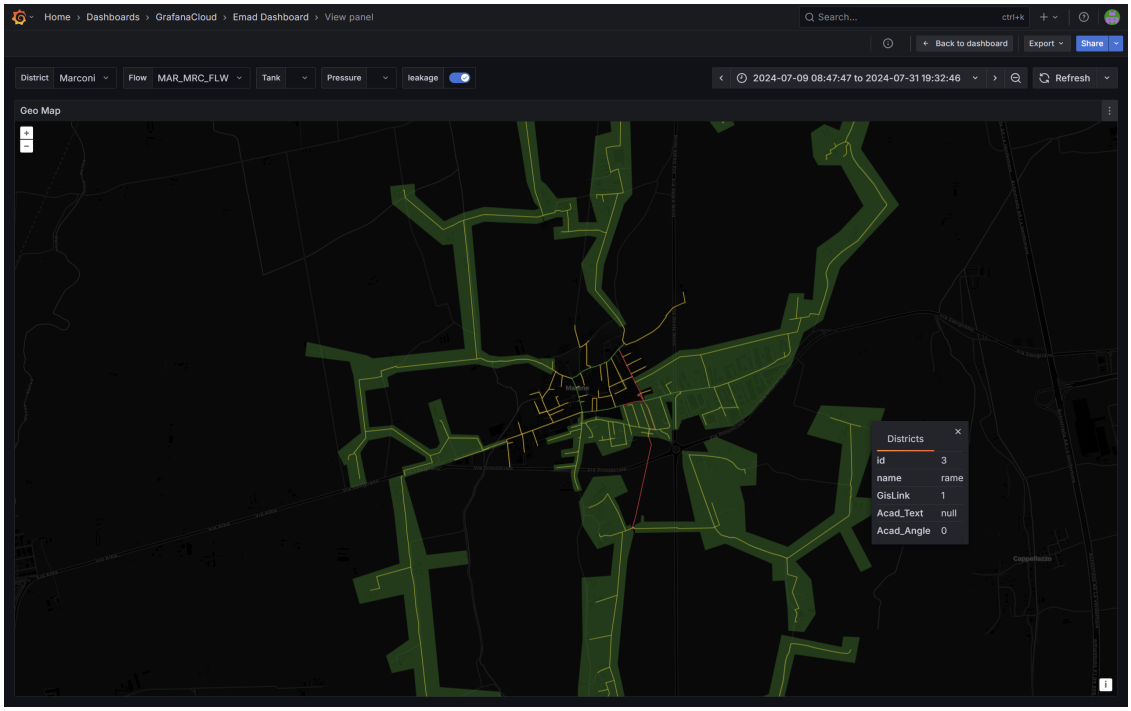

Figure 4.3: Geomap panel of the Marene water distribution network with district boundaries.

Figure 4.3 demonstrates the geomap visualization capabilities, showing the complete network topology with district boundaries clearly delineated. The interactive map displays sensor locations, pipe networks, and enables operators to visualize network topology and quickly identify geographic areas of interest. This interactive exploration functionality supports comprehensive geographic monitoring of different areas within the Marene municipality.

4.2.2 Time-Series Monitoring and Leak Detection

The system provides sophisticated time-series visualization for monitoring key parameters including flow rates, pressure levels, and leak probability indicators. These visualizations enable operators to identify trends, anomalies, and potential issues across different time periods.

Figure 4.4: Time-series panel showing leak probability indicators across districts.

Figure 4.4 illustrates the time-series monitoring capabilities, specifically showing leak probability indicators across different districts over time. The visualization includes threshold lines and automated anomaly detection indicators that enable operators to identify potential issues and make informed operational decisions. These trend analysis tools support continuous performance tracking and early detection of developing problems within the water distribution network.

4.2.3 Time Range Selection and Navigation

The dashboard system provides flexible time range selection capabilities that enable operators to analyze data across different temporal scales, from real-time monitoring to historical analysis spanning weeks or months.

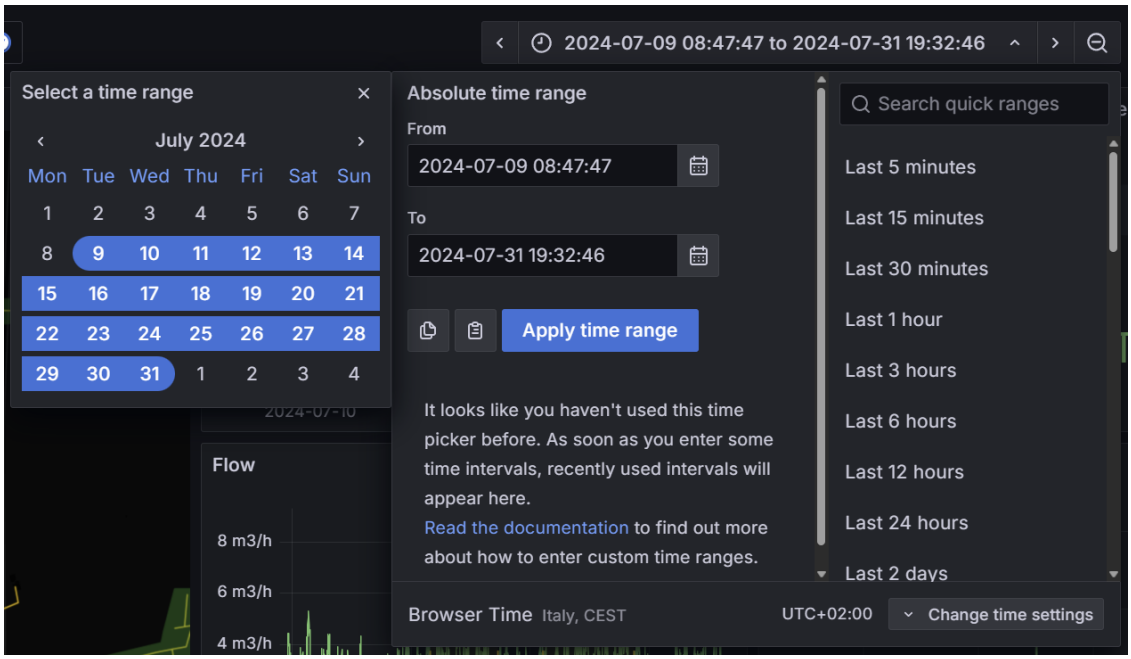

Figure 4.5: Time range selection interface for monitoring data navigation.

Figure 4.5 demonstrates the time navigation capabilities, showing the intuitive interface for selecting different time ranges. The system supports both preconfigured time periods (last hour, last day, last week) and custom date/time selection options. This flexibility enables operators to quickly navigate between different temporal views of the monitoring data, supporting both real-time monitoring and historical analysis spanning weeks or months.

In addition to the graphical interface, the conversational AI system also provides time selection functionality through natural language commands. Operators can use phrases like "show me data from yesterday," "set time range to last 6 hours,"

or "display data from March 10th to March 15th" to dynamically adjust the dashboard's time range. This integration of time selection capabilities within the conversational interface further enhances operational efficiency by allowing temporal navigation through the same unified interaction model.

4.3 Conversational AI Interface Results

The conversational AI interface demonstrates sophisticated natural language processing capabilities for water distribution monitoring operations, providing an intuitive and efficient method for operators to interact with the monitoring system.

4.3.1 Supported Conversational Functions

The chatbot supports a comprehensive set of operations that enable complete dashboard management and data analysis through natural language commands. The core capabilities include:

- **Dashboard Management**: Load specific dashboards by UID, load default dashboards, and navigate between different monitoring views
- **Panel Operations**: Add new panels (timeseries, table, heatmap, gauge), remove existing panels, and modify panel properties (size, position, title, datasource) with user confirmation
- **Data Querying**: Execute natural language queries about network data, with automatic SQL generation and transparent query display
- **Geospatial Navigation**: Zoom geomap panels to specific districts, enabling focused geographic analysis
- **Time Range Management**: Modify dashboard time ranges using natural language (e.g., "last 6 hours", "yesterday", specific date ranges)
- **Variable Management**: Add or remove dashboard variables with user confirmation
- **System Safety**: Undo functionality to revert changes, save confirmation workflows, and version control through Grafana's history

4.3.2 Natural Language Processing Capabilities

The system uses Google's Gemini 2.0 Flash model through LangChain and LangGraph frameworks, implementing a hybrid architecture where natural language understanding is combined with precise technical operations. This approach addresses reliability concerns while maintaining natural language accessibility.

The interface provides real-time communication and transparent query execution, offering immediate feedback and confirmations to user requests. This natural language interaction capability significantly reduces the learning curve for new system users and provides efficient access to complex monitoring functionality.

4.3.3 Use Case Demonstration: Modifying Panel Properties

The system demonstrates practical utility through realistic operational scenarios. One important use case involves modifying existing panel properties to optimize dashboard layouts and improve visualization effectiveness.

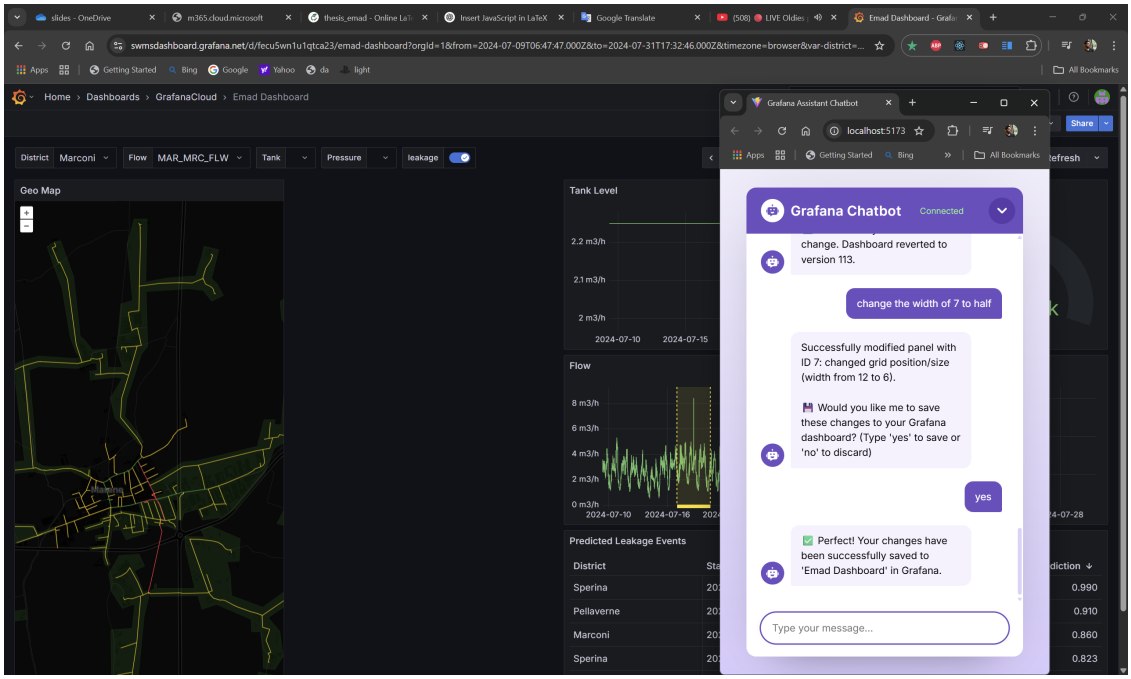

Figure 4.6: Panel modification workflow showing width adjustment of panel on the dashboard.

Figure 4.6 illustrates the complete workflow for modifying panel properties through the conversational AI interface. The interaction shows an operator requesting to change the width of panel ID 7 (a geomap panel) to half of its current size, with the system interpreting the natural language request, extracting the relevant parameters (panel ID, desired width), and providing a confirmation dialogue before implementing the changes. After confirmation, the system displays the updated dashboard with the resized geomap panel, demonstrating how spatial visualizations can be dynamically adjusted to fit specific layout requirements. This panel

modification capability enables operators to adjust various visualization properties including size, position, title, and datasource through simple natural language commands without requiring knowledge of the underlying Grafana configuration syntax. The structured workflow with explicit user confirmation ensures that operators maintain complete control over dashboard modifications while benefiting from the simplicity and efficiency of conversational interactions.

4.3.4 Use Case Demonstration: Data Analysis Queries

The system enables sophisticated data analysis through natural language queries that are automatically translated into SQL operations against the BigQuery data warehouse.

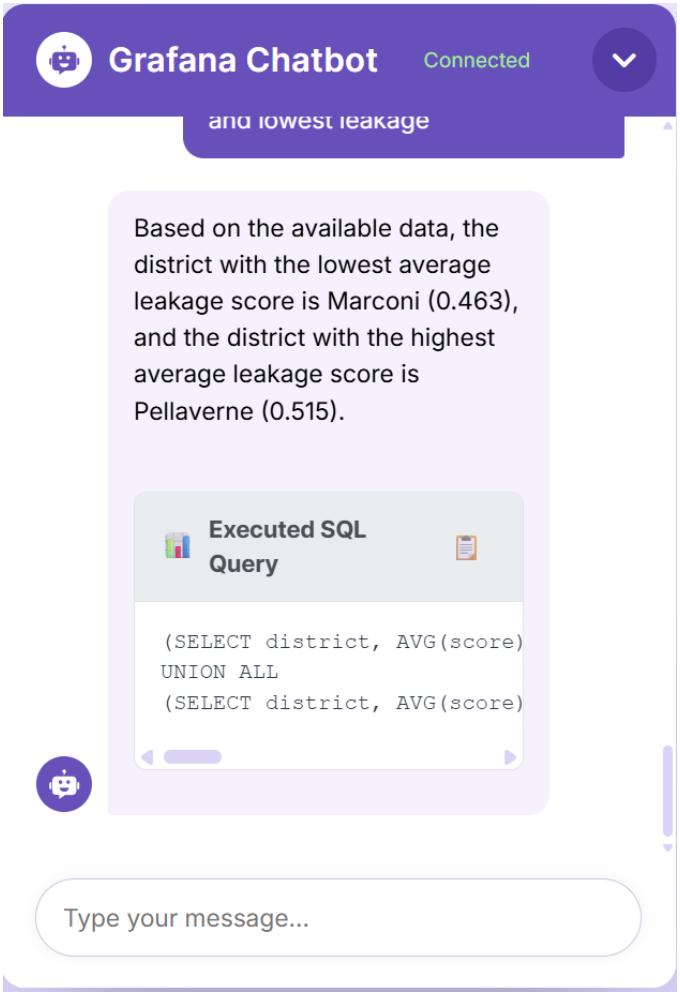

Figure 4.7: Natural language data query for districts with highest and lowest leakage, with transparent SQL generation.

Figure 4.7 demonstrates the data querying capabilities, showing how operators can ask complex questions about network performance using natural language. The example shows a query requesting districts with both the highest and lowest leakage probability, with the system displaying both the natural language interpretation and the generated SQL query for transparency. This approach provides technical visibility while maintaining the accessibility of natural language interaction, enabling operators to verify the system's understanding of their requests.

4.3.5 Use Case Demonstration: Geospatial Navigation

The system supports intuitive geospatial navigation through conversational commands that enable operators to focus on specific geographic areas of interest.

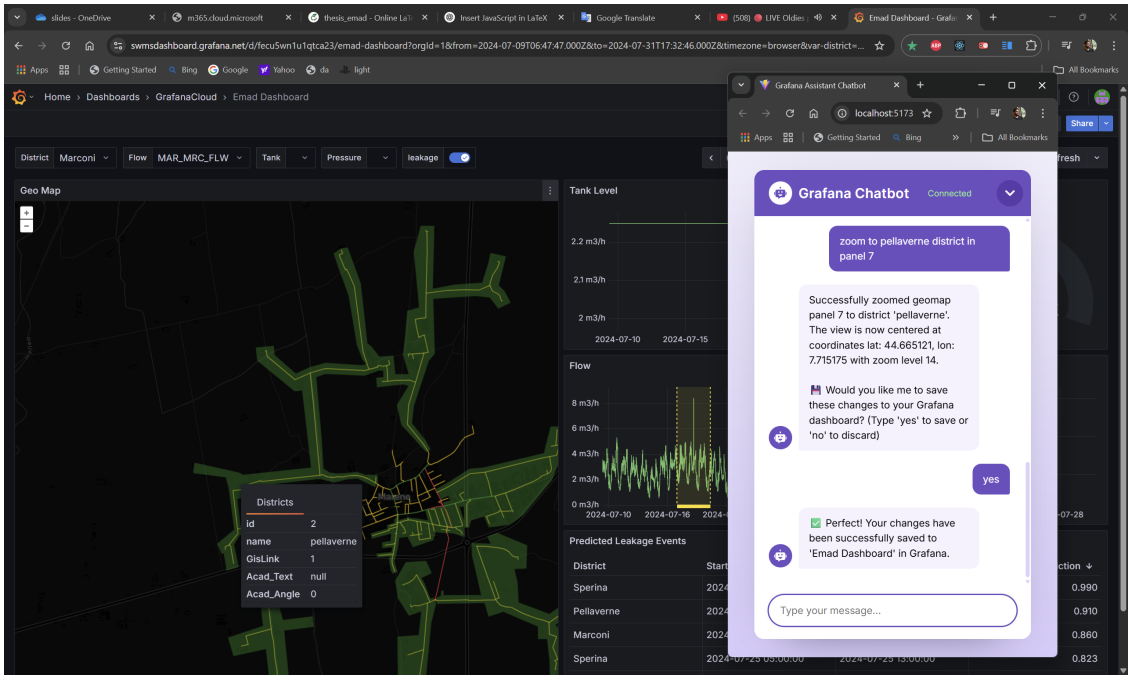

Figure 4.8: Geospatial navigation to Pellaverne district using natural language commands.

Figure 4.8 shows the geospatial navigation capabilities, demonstrating how operators can use natural language to request specific geographic views. The conversational command to zoom a geomap to the Pellaverne district showcases the system's ability to understand geographic references and automatically adjust map views with user confirmation, providing immediate visual feedback to the operator's request.

4.3.6 System Safety and Version Control

The conversational AI includes comprehensive safety mechanisms to prevent accidental modifications and enable easy recovery from unwanted changes.

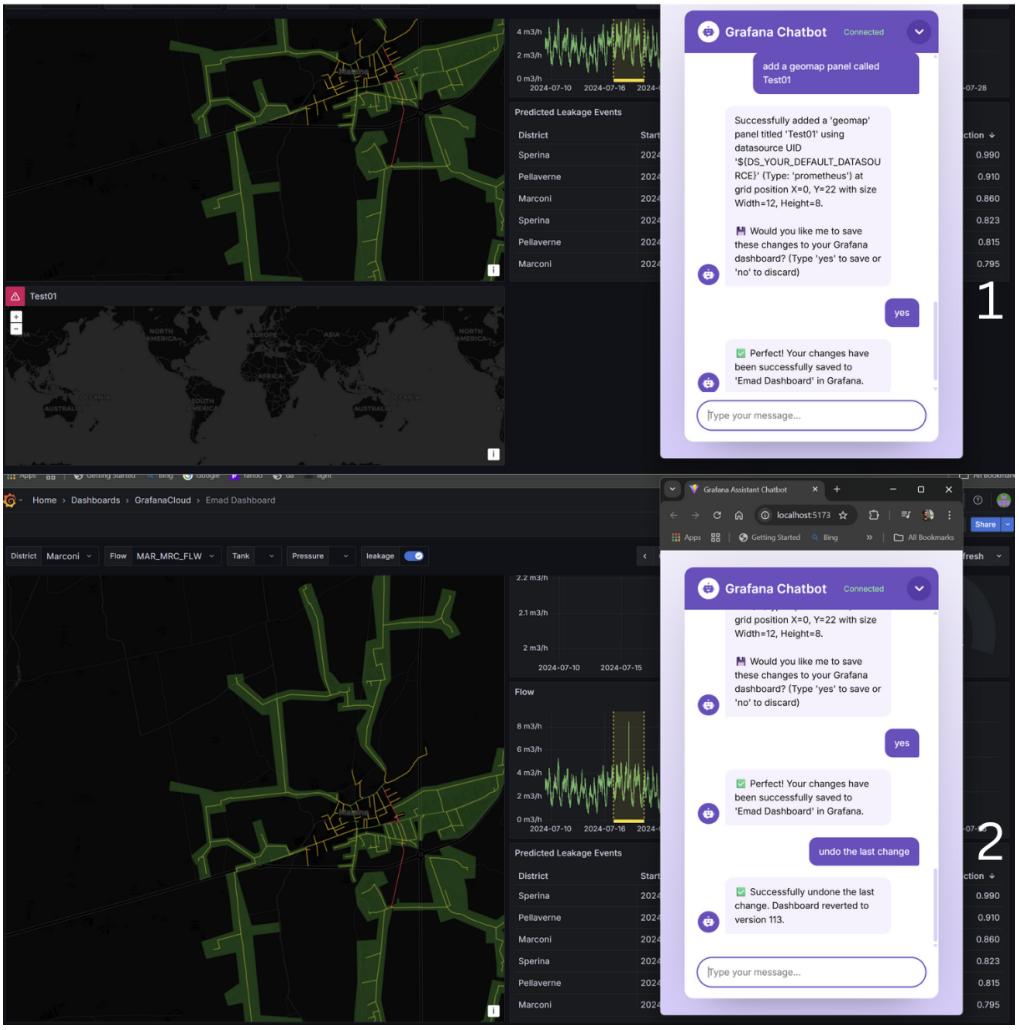

Figure 4.9: Add-then-undo workflow showing panel addition and subsequent change reversal.

Figure 4.9 illustrates the complete workflow of the safety mechanisms within the system. The first image shows the successful addition of a geomap panel named "Test01" through the chatbot interface, demonstrating how operators can add new panels through conversational commands. The second image shows the user reverting these changes using the "undo the latest changes" command, with the system successfully removing the previously added panel and loading the previous version of the dashboard. This safety feature leverages Grafana's version history for reliable

state restoration, ensuring that operators can experiment with dashboard modifications while maintaining the ability to quickly return to previous configurations if needed.

4.4 Technical Implementation Validation

4.4.1 WebSocket Communication and Real-Time Performance

The implementation demonstrates real-time bidirectional communication through WebSocket connections, enabling persistent conversation state and immediate response. The system maintains conversation context across multiple interactions, providing a seamless user experience for complex operational tasks.

4.4.2 Cloud Infrastructure Performance

The Google Cloud Platform implementation provides effective serverless scaling and BigQuery time-series optimization for both real-time monitoring and historical analysis. Automated error detection and recovery mechanisms ensure system reliability, while the cloud-native architecture enables efficient resource utilization and cost management.

4.4.3 System Validation Summary

The implemented system successfully demonstrates comprehensive monitoring capabilities through visual dashboards and conversational AI interface. The validation confirms effective integration of cloud infrastructure, data processing, visualization, and natural language processing components.

4.4.4 Visual Dashboard Validation

The Grafana-based dashboard system provides intuitive visualization of water distribution network data through interactive maps, time-series charts, and flexible time navigation controls. The visual interface enables operators to quickly identify network status, trends, and potential issues across different geographic districts and time periods.

4.4.5 Operational Utility Validation

The system processes time-series monitoring data efficiently, integrates geospatial data for spatial analysis, and provides comprehensive dashboard management capabilities. The hybrid architecture approach effectively balances natural language

understanding with technical precision, while the Marene case study validates practical operational utility for municipal water system management.

The iterative development approach successfully progressed from CLI prototypes to the final web-based system, with each phase providing insights that guided architectural decisions. The comprehensive safety mechanisms, including save confirmation workflows and undo functionality, ensure operational reliability while maintaining ease of use.

Chapter 5

Conclusion and Future Directions

This thesis presents the development and implementation of an integrated cloud-based infrastructure for smart water distribution monitoring with conversational AI capabilities. The work demonstrates significant advancement in creating comprehensive monitoring solutions that integrate cloud technologies, real-time data processing, interactive visualization, and intelligent user interfaces for infrastructure monitoring applications.

5.1 Research Contributions and Achievements

5.1.1 Primary Technical Contributions

The primary contribution is the successful development of an integrated cloud-based monitoring system that combines multiple technologies to create a comprehensive solution for water distribution network management. The system integrates cloud infrastructure, real-time data processing, interactive visualization, and conversational AI capabilities to provide a complete monitoring solution.

Key technical achievements include:

- Cloud-native architecture leveraging Google Cloud Platform services for scalable data processing and storage
- Real-time data integration from multiple sources including SCADA systems and IoT sensors
- Interactive dashboard system using Grafana for comprehensive visualization and monitoring
- Geospatial data integration enabling geographic analysis and district-based monitoring

- Conversational AI interface with domain-specific prompt engineering for technical terminology

The comprehensive system demonstrates effective integration of multiple technologies while maintaining system coherence and reliability, providing a reusable pattern for similar infrastructure monitoring applications.

5.1.2 Methodological and Domain-Specific Contributions

The iterative development methodology established effective practices for developing comprehensive infrastructure monitoring systems, progressing through incremental validation phases from cloud infrastructure setup to data integration, visualization development, and finally AI interface implementation. This methodology provides a reusable framework for similar multi-component system development.

Within water distribution monitoring, this work demonstrates comprehensive system integration that combines:

- Cloud-based data processing and storage for scalable monitoring operations
- Real-time sensor data integration with historical analysis capabilities
- Interactive visualization systems enabling intuitive data exploration
- Geospatial analysis capabilities for district-based monitoring and management
- Conversational interfaces that democratize access to complex technical systems

The integration of hydraulic simulation results with real-time sensor data through both visual and conversational interfaces enables new analyses previously impractical due to system complexity, with applications for predictive maintenance, emergency response, and operational optimization.

5.2 Technical Innovation and Broader Implications

This work represents a comprehensive approach to modernizing infrastructure monitoring through the integration of cloud technologies, interactive visualization, and artificial intelligence. The technical innovations have implications extending beyond water distribution monitoring to other critical infrastructure domains.

The cloud-native architecture demonstrates effective patterns for scalable infrastructure monitoring, while the interactive visualization capabilities provide intuitive interfaces for complex data analysis. The integration of geospatial data

enables sophisticated spatial analysis capabilities, and the conversational AI interface represents the first documented successful deployment of large language models for industrial dashboard management.

The prompt engineering techniques for translating natural language commands into technical operations while maintaining safety demonstrate effective approaches for applying general-purpose language models to domain-specific applications. The transparent query generation and confirmation workflow patterns address fundamental concerns about AI reliability in operational contexts, while the modular architecture enables selective adoption without requiring complete system replacement.

The comprehensive system architecture provides a blueprint for developing modern infrastructure monitoring solutions that balance technical sophistication with operational usability.

5.3 Limitations and Future Research Directions

5.3.1 Current Limitations

The system has several limitations that present opportunities for future enhancement. The cloud infrastructure, while scalable, is currently optimized for Google Cloud Platform services and would require adaptation for other cloud providers. The visualization system demonstrates effective integration with Grafana but would need additional development for other monitoring platforms.

The conversational AI capabilities have limitations with highly specialized technical queries and ambiguous temporal references. Mobile device optimization was not prioritized, affecting utility for field operations. The current data integration focuses on SCADA and sensor data but could be expanded to include additional data sources such as customer service systems and maintenance records.

5.3.2 Future Research Opportunities

Future research directions include:

- Advanced data integration capabilities including IoT sensor networks and smart meter data
- Multi-agent AI systems for comprehensive automated analysis and decision support
- Mobile application development for field operations and emergency response
- Integration with additional monitoring platforms beyond Grafana

- Continuous learning capabilities adapting to operational contexts and user preferences
- Smart city applications for comprehensive urban infrastructure management

5.4 Real-World Validation and Industry Collaboration

A critical aspect of this research was the emphasis on real-world validation through direct collaboration with industry stakeholders rather than purely theoretical development. This approach provided invaluable insights that shaped both the technical implementation and practical applicability of the solution.

5.4.1 Value of Industry Partnership

The collaboration with AlpiAcque and Tesisquare provided essential domain expertise and operational context that fundamentally influenced the system development. Regular feedback sessions revealed user interface requirements, workflow integration needs, and practical constraints that would not have been apparent from technical specifications alone.

This industry partnership enabled:

- Validation of technical approaches against real operational requirements
- Understanding of user interface preferences and workflow integration needs
- Identification of practical deployment challenges and scalability concerns
- Access to authentic data and operational scenarios for testing

The iterative collaboration process ensured that the final system addresses genuine operational needs while maintaining the technical innovation goals of the research, resulting in a system that demonstrates both academic research contributions and practical utility for real-world applications.

5.5 Concluding Remarks

This thesis demonstrates that integrating cloud technologies, interactive visualization, and conversational AI creates comprehensive infrastructure monitoring solutions that provide significant operational improvements while establishing new paradigms for system design and human-computer interaction in technical contexts. The successful implementation validates the potential for multi-component systems

to make complex technical capabilities accessible while maintaining accuracy and reliability.

The emphasis on real-world validation through industry collaboration proved essential for developing a system that addresses genuine operational needs rather than theoretical possibilities. This collaboration model demonstrates the critical importance of industry partnership in developing comprehensive infrastructure monitoring systems.

The work provides a foundation for continued advancement in intelligent infrastructure monitoring, demonstrating that modern cloud technologies combined with interactive visualization and AI can enhance human expertise while improving accessibility and operational effectiveness. The comprehensive system architecture provides a blueprint for future infrastructure monitoring projects, emphasizing the value of integrated approaches and continuous stakeholder engagement throughout the development process. As infrastructure systems become increasingly complex, the approaches developed provide valuable tools for organizations seeking to modernize monitoring capabilities while maintaining reliability and safety requirements essential for critical systems.

Bibliography

[1] Grafana Labs. *Grafana: The Open Observability Platform*. Version 10.0+. 2024. URL: https://grafana.com/.

[2] Google Cloud. *Google Cloud Platform Documentation*. Accessed: 2024-01-15. 2024. URL: https://cloud.google.com/docs.

[3] Google DeepMind. *Gemini: Google's Most Capable AI Model*. Gemini 1.5 Pro. 2024. URL: https://deepmind.google/technologies/gemini/.

[4] LangChain Inc. *LangChain: Building Applications with LLMs through Composability*. Version 0.1+. 2024. URL: https://python.langchain.com/.

[5] LangChain Inc. *LangGraph: Build Language Agents as Graphs*. Version 0.0.26+. 2024. URL: https://python.langchain.com/docs/langgraph.

[6] Sebastian Ramirez. *FastAPI: Modern, Fast (High-performance), Web Framework for Building APIs with Python*. Version 0.104+. 2024. URL: https://fastapi.tiangolo.com/.

[7] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. RFC 6455. 2011. URL: https://tools.ietf.org/html/rfc6455.

[8] Sergey Melnik et al. «Dremel: Interactive Analysis of Web-Scale Datasets». In: *Communications of the ACM* 63.6 (2020), pp. 114–123. DOI: 10.1145/3318464.

[9] Meta (Facebook). *React: A JavaScript Library for Building User Interfaces*. Version 18+. 2024. URL: https://react.dev/.

[10] Shuo Chen et al. «BigQuery and Dremel: Large-scale data analytics at Google». In: *Proceedings of the 2016 International Conference on Management of Data*. ACM. 2016, pp. 2285–2286. DOI: 10.1145/2882903.2903731.

[11] Ioana Baldini et al. «Serverless computing: Current trends and open problems». In: *Research Advances in Cloud Computing* (2017), pp. 1–20. DOI: 10.1007/978-981-10-5026-8_1.

[12] Wes McKinney. *Python for data analysis: Data wrangling with pandas, NumPy, and IPython*. 2nd. O'Reilly Media, 2017. ISBN: 978-1491957660.

[13] Paul Barham et al. «Xen and the art of virtualization». In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 164–177. DOI: 10.1145/1165389.945462.

[14] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. «The Google file system». In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 29–43. DOI: 10.1145/1165389.945450.

[15] Jeffrey Dean and Sanjay Ghemawat. «MapReduce: simplified data processing on large clusters». In: *Communications of the ACM* 51.1 (2008), pp. 107–113. DOI: 10.1145/1327452.1327492.

[16] Apache Software Foundation. «Apache Airflow: A platform to programmatically author, schedule, and monitor workflows». In: *Apache Airflow Documentation* (2016). Version 2.0+. URL: https://airflow.apache.org/.

[17] Charles R Harris et al. «Array programming with NumPy». In: *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. «Isolation forest». In: *2008 eighth ieee international conference on data mining* (2008), pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[19] PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Version 15+. 2024. URL: https://www.postgresql.org/.

[20] PostGIS Development Team. *PostGIS: Spatial and Geographic Objects for PostgreSQL*. Version 3.3+. 2024. URL: https://postgis.net/.

[21] QGIS Development Team. *QGIS: A Free and Open Source Geographic Information System*. Version 3.28+. 2024. URL: https://qgis.org/.

[22] Howard Butler et al. *The GeoJSON Format*. RFC 7946. 2016. URL: https://tools.ietf.org/html/rfc7946.

[23] Tom Brown et al. «Language models are few-shot learners». In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[24] U.S. Environmental Protection Agency. *EPANET: Software for Water Distribution System Modeling*. Version 2.2. 2020. URL: https://www.epa.gov/water-research/epanet.

[25] Lewis A Rossman. *Computer modeling of water distribution systems*. 3rd. American Water Works Association, 2014. ISBN: 978-1583218433.

[26] OpenAPI Initiative. *OpenAPI Specification*. Version 3.1.0. 2024. URL: https://spec.openapis.org/oas/v3.1.0.

[27] Vite Team. *Vite: Next Generation Frontend Tooling*. Version 5.0+. 2024. URL: https://vitejs.dev/.

[28] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Doctoral dissertation. 2000.

[29] QWC Services Development Team. *QWC Services: QGIS Web Client Services*. Web-based GIS client for QGIS Server. 2024. URL: https://qwc-services.github.io/master/.

[30] AlpiAcque S.p.A. *AlpiAcque S.p.A. - Water Utility Company*. Accessed: 2024-12-15. 2024. URL: https://www.alpiacque.it/.

[31] Technical Design. *GISMaster 3 - Advanced GIS Platform for Water Distribution Networks*. Accessed: 2024-12-15. 2024. URL: https://www.technicaldesign.it/gismaster-3/.

[32] IDEA S.r.l. *IDEA S.r.l. - Industrial Data Engineering and Automation*. Accessed: 2024-12-15. 2024. URL: https://www.idea-srl.it/web/.

[33] Tesisquare S.p.A. *Tesisquare S.p.A. - Technology Solutions for Utilities*. Accessed: 2024-12-15. 2024. URL: https://www.tesisquare.com/.