

Conversational Hydraulic Modelling: An Open-source Model Context Protocol Server for EPANET

Dennis Zanutto^{1,2}, Christos Michalopoulos^{1,3}, Dragan Savić^{1,4}

¹ KWR Water Research Institute, Groningenhaven 7, Nieuwegein, 3433 PE, the Netherlands

²Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy

³Department of Water Resources and Environmental Engineering, National Technical University of Athens, 15773 Athens, Greece

⁴Centre for Water Systems, University of Exeter, Exeter EX4 4QD, UK

Corresponding author: dennis.zanutto@kwrwater.nl

Abstract. Digital twins of water distribution systems provide control room operators with automated control capabilities and enable faster, more detailed system diagnostics. These tools typically rely on the open-source EPANET simulation engine for accurate hydraulic simulations. However, the technical expertise required to run or integrate the EPANET library and similar alternatives can be substantial, limiting their adoption and hindering operators from making evidence-based decisions. This work introduces the EPANET Model Context Protocol (MCP) server, an open-source solution that integrates EPANET with Large Language Models and other MCP-compatible applications. By implementing the MCP, a standard that provides a common interface for AI to interact with external tools, this software enables practitioners and researchers to run complete hydraulic simulations, retrieving system information and results through a conversational interface. This work focuses on establishing the architectural groundwork and demonstrating its potential through a few examples, but it already enables the use of EPANET across all major LLM-powered applications.

Keywords: Model Context Protocol, Large Language Models, Hydraulic Simulator, Optimisation

1 Introduction

Hydraulic modelling sits at the heart of hydroinformatics research and engineering practice. Developing programming and technical skills to interrogate models and extract actionable insights from output data has become a core competency for the modern hydraulic engineer and researcher [1]. The field of hydroinformatics has made significant strides in lowering the entry barrier, leveraging more intuitive interfaces and open-source frameworks to make advanced hydraulic analysis more widely accessible [2].

Modern interfaces for common operational tools, such as SCADA systems and high-level digital twins, shield operators from the underlying technical complexity. While these digital tools provide significant benefits through faster diagnostics and detailed monitoring, their features are inherently rigid, limiting users within the predefined boundaries of the user interface (UI). This cognitive load imposed by existing interfaces is significant, and advanced ad-hoc analysis and visualisations remain confined to researchers and academics, accessible only through laborious, non-reusable scripting and programming [3].

This technical burden is further aggravated by the challenge of training operators to develop reliable intuition about complex system behaviour. The nuanced “what and when to look at,” and most importantly, the “what to do when” are skills traditionally built through years of hands-on experience. Without the “gut feeling” developed through prolonged exposure to the system, there is a pressing need to offload some of these responsibilities to the tools themselves. Modern software must evolve

beyond merely displaying data; it should process information and guide decision-making toward accurate interpretations by adhering to industry best practices.

Coding assistance and “copiloting” are precisely the tasks for which Large Language Models (LLMs) excel [4]. Besides the marketing, the recent explosion of tools speaks to the nature of this opportunity. General-purpose LLMs excel at coding tasks and contextualising technical information, but a lack of domain-specific grounding constrains their utility. Without proper context, hallucinations can persist, and the practical gains quickly decrease [5].

The application of LLMs to water systems is still in its early stages, with only a handful of works exploring this direction. The strength of Natural Language Processing techniques lies in text-oriented tasks, such as consumer complaint processing [6], and education and outreach [7]. Less explored is how to leverage these tools for more complex, context-specific tasks. Wang et al. [8] proposed a multi-agent LLM framework to automate water distribution network optimisation routines, with examples of network calibration and pump optimisation. However, to enable the LLM to interact with EPANET and run simulations, the authors had to design and implement two specialised agents from scratch (one of which was specifically instructed to code with the EPANET Python library). While effective, this approach is difficult to reproduce, requires significant implementation effort, and demands careful tuning. Ideally, one would simply connect a validated, specialised tool directly to the LLM.

MCP (Model Context Protocol) is an open-source standard for connecting LLMs-enabled applications to external systems, such as data sources (e.g., local files, databases) and tools (e.g., search engines, calculators) [9]. The MCP protocol’s goal is to provide structured, queryable access to “applications”; it does not extend what an LLM knows; it extends what it can do by connecting its language and reasoning capabilities to the necessary tools, at the right moment, with the appropriate context. The architecture follows a client-server model, where an AI-powered application (e.g., a chatbot) establishes connections to multiple external systems via the MCP servers they expose.

Exposing an MCP server for an existing application provides two immediate benefits for developers. First, it makes their application AI-ready; it becomes accessible to any MCP-compatible platform, allowing users to interact with it through natural language without needing to learn the native UI or master the underlying Application Programming Interface (API). Second, it enables seamless integration across the broader MCP ecosystem: rather than managing costly and brittle direct API integrations, an application can rely on other specialised MCP servers. For example, a hydraulic simulation tool could focus solely on producing raw results, while a separate visualisation tool (another MCP server) could handle ad-hoc rendering, with the LLM providing the shared context that binds them together.

This work’s primary contribution is the introduction of an open-source MCP server for EPANET [10], the dominant open-source engine for water distribution network modelling. Despite its widespread adoption across research and practice, EPANET carries significant technical debt, and its native graphical interface dates back decades. While the broader ecosystem of tools built around it remains fragmented, the industry naturally moves toward more polished, specialised commercial platforms, leaving EPANET’s flexibility primarily in the hands of researchers and developers comfortable enough to exploit it programmatically. The implementation presented here operates at the application layer rather than at the engine core, leveraging mature Python bindings [11] to orchestrate simulation configuration, execution, and results extraction and visualisation. The outcome is an MCP server that brings AI-ready capabilities to the world’s most widely adopted open-source hydraulic simulation engine, allowing practitioners and researchers to interrogate water distribution models, run scenario analyses, and extract insights through natural language, without writing a single line of code or navigating a legacy interface.

Rather than a finished product, we establish the architectural groundwork and demonstrate its potential, inviting the broader community to extend and build upon it. The server’s capabilities are shown through a series of tailored examples on established benchmark networks. For the water industry, this work presents an opportunity to experiment with what next-generation interfaces could

look like, identify early opportunities, and detect risks and vulnerabilities before AI systems are given agency over critical infrastructure models. For the research community, it is a further step towards lowering the entry barrier for hydraulic modelling and keeping pace with a technology landscape that will not wait.

The remainder of this paper is structured as follows. We begin in Section 2 by establishing the software architecture and design, with particular attention to the core elements of the Model Context Protocol (MCP) standard and the role of the EPANET MCP server within this ecosystem. Section 3 then details the technical implementation of our software. To demonstrate its practical utility, we present in Section 4 a series of illustrative examples on publicly available water distribution network models. Section 5 presents the main risks of this technology, and Section 6 offers concluding remarks.

2 Software Architecture and Design

In this section, we introduce the software architecture by outlining the core elements of the Model Context Protocol (MCP) standard and the EPANET MCP server role and capabilities in this ecosystem.

2.1 The Model Context Protocol (MCP)

The Model Context Protocol (MCP) is an open-source standard for connecting Large Language Models (LLMs) to external software tools. The specifications outline a client-server architecture in which the MCP host is an LLM-enabled application that can connect to one or more MCP servers. It does so by creating one or more MCP clients, each maintaining a dedicated connection to its corresponding server.

The protocol works equally well for both local and remote MCP servers, with the only difference being the transport layer: STDIO for local communication and Streamable HTTPS for remote communication. More details regarding the Transport Layer of the protocol can be found in the official documentation [9].

The Data Layer of the protocol, instead, defines the schema and semantics between MCP clients and servers, with the core concepts covering life-cycle management, primitives, and notifications. For this work's purpose, we will focus only on the MCP server primitives. The keen reader is referred to the official (and evolving) documentation for a more comprehensive and up-to-date treatment.

The three primitives that a server can expose are:

- Tools: routines or executables that the LLM can invoke to perform an action (examples include querying a database or searching for a file)
- Resources: Data Sources providing the context to the LLM (e.g., schema of a database)
- Prompts: Reusable templates that help structure interactions with LLMs (e.g., examples for interacting with the tools)

As shown in Figure 1, the primary user interface is the LLM (MCP Host), which is responsible for intent recognition by interpreting high-level queries (e.g., “Simulate this inp file”). It also determines whether to call any tool and which one to provide an answer. Guided by the server's resources and prompts as context, the LLM activates the correct client, which calls the requested tool from its corresponding server. With extreme simplification, the MCP server serves as a standardised bridge for an LLM to run an executable or retrieve a resource.

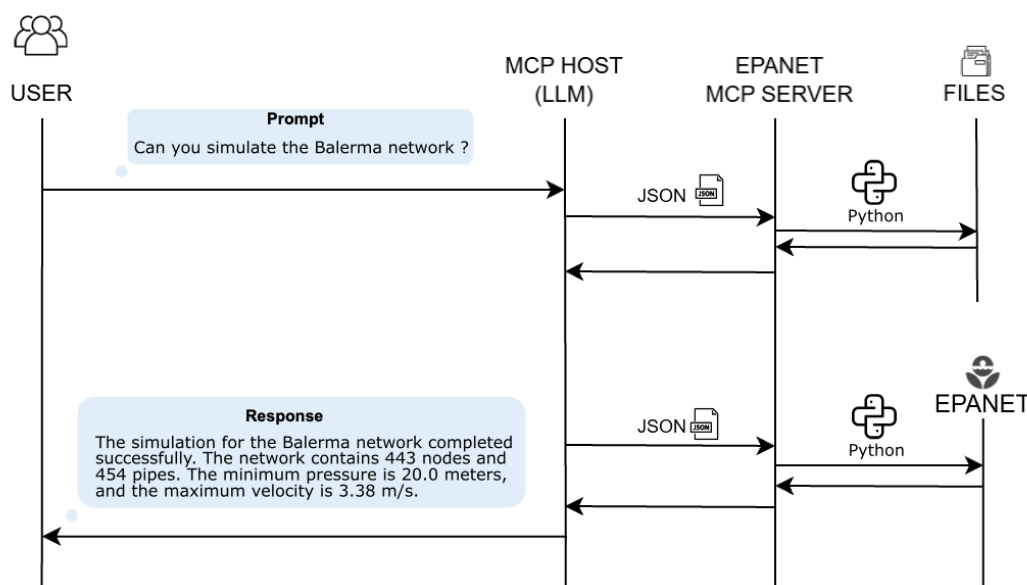


Figure 1: Visual representation of a user interaction with the EPANET MCP server.

2.2 The EPANET MCP Server

As detailed in Section 2.1, the MCP servers act as intermediaries between the LLM and resources or executables. EPANET is a hydraulic modelling software and a hydraulic solver (engine). Therefore, the EPANET MCP server tools should expose these same functionalities:

- modelling a water distribution network (e.g. adding and removing pipes, changing demands, etc.),
- solving the hydraulics (e.g., running a steady-state simulation),
- visualising the results and the model.

Meanwhile, the EPANET MCP server prompts would provide the user with examples of how to use them. These do not differ significantly from the examples in the EPANET documentation, but are directly integrated into the MCP host interface, thereby simplifying access and improving reusability for the user.

Finally, the EPANET MCP server resources represent the biggest difference between this technology and the EPANET programming interface (API) or user interface (UI) and offer new opportunities for interacting with hydraulic models. Resources provide context for the LLMs and enable more grounded, domain-specific responses. First, they explain the components and characteristics of the hydraulic model. For example, the differences between node types (reservoir, junction, tank) and the components' properties (e.g., elevation, demand, etc.). For this purpose, the EPANET documentation is a perfect starting point and requires minimal refinements to optimise it for LLMs. While LLMs are trained on vast amounts of data, which likely includes the EPANET documentation, integrating these documents as direct resources ensures the model's responses are anchored in the technical documentation and in hydraulic engineering practices, such as strategies and workflows for handling warnings and errors, and interpreting results. For example, upon return of the "network contains unconnected nodes" error, the LLM can go beyond merely displaying the error and, after querying a specialised resource on error handling, provide the user with a follow-up action in line with best practices (e.g., localise the unconnected junction, then query the status of the connected pipes, etc.). This knowledge is usually available to operators and researchers in different forms, scattered across documentation sources, and as institutional knowledge. This architectural shift simplifies knowledge retrieval by presenting information directly in the user's workflow. This promises to be particularly useful for water utilities, as they can complement the MCP server's resources with their own documentation on workflows and strategies, providing operators with more structured, contextual, and reusable information.

3 Implementation Details

Several official Software Development Kits (SDKs) in different programming languages are available for MCP server development. By overlapping them with the available implementations in the EPANET ecosystem of libraries, we opted for a Python implementation of the EPANET MCP server. Precisely, in this first implementation, we use the official MCP Python SDK with FastMCP and the EPyT library [11].

The primary advantage of implementing MCP over a standard REST API is the substitution of a static, predefined interface with a dynamic, declarative approach. In traditional API development, developers must manually construct and maintain complex schemas that define how clients interact with their software. For every change to the API, clients must reflect the update. In contrast, MCP utilises Pythonic, self-documenting code to allow the system to dynamically describe its capabilities to the LLM. By leveraging the function names, type hints, and docstrings already present in the Python source code, FastMCP automatically generates the necessary context to tell the LLM the available tools and their required arguments.

The outcome is a different approach to software interaction to facilitate LLM operations, favouring a few higher-level actions rather than many tailored entry points.

4 Demonstrative Examples

The EPANET MCP server streamlines hydraulic simulations by allowing users to run models and inspect results through any MCP-compatible AI application. Depending on the application used, the interface can offer basic visualisation tools for static images or more advanced interactive plots. For this demonstration, we use GitHub Copilot Chat, a Visual Studio Code plugin, as the MCP Host, and OpenAI GPT 4.1 as the LLM powering it. The EPANET MCP server source code is available as open source on GitHub and backed up on Zenodo [12]. As the complete capabilities of the tool will be influenced by this technology adoption and user requirements, in the following sections, we demonstrate a handful of possible interactions with the current implementation (Version 0.2.0).

While the MCP-server-based solution's reliability is highly dependent on the LLM provider (and model version), our tests on these examples recorded zero execution failures, whereas direct Python code generation consistently suffered from EPANET API hallucinations.

4.1 Simulation Set-up and Execution

The simulation execution tool automates the solution of the hydraulic equations for a network model starting from EPANET's input files (.inp). A key feature of this implementation, highlighting the flexibility offered by LLMs, is its context-aware file selection: users do not need to provide exact filenames. Instead, the MCP server provides the LLM with the available files, and the LLM selects the documents that best match the user's natural language command. Upon execution, the system provides a high-level summary of the simulation results, including network topology, hydraulic status, and returned errors.

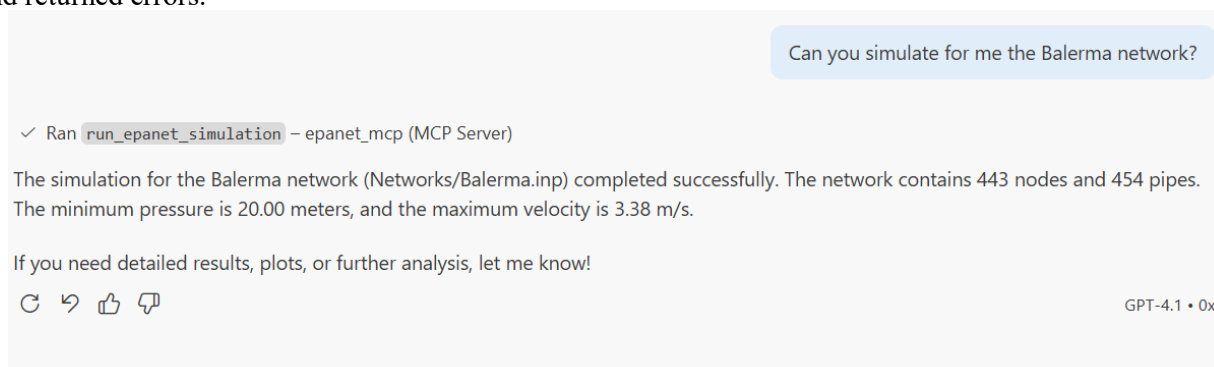


Figure 2: Demonstration of how the EPANET MCP server gives an LLM the ability to run simulations for the user. The LLM agent locates the requested input file, runs the hydraulic simulation in EPANET, and then provides the user with informative details about the network and the simulation.

4.2 Network Status Modification

Users can modify the network architecture, such as changing pipe status (open/closed), adding or removing pipe elements, and adjusting their diameters. Unlike the EPANET API, all interventions are handled by a single, multi-purpose tool of the MCP server. Figure 3 shows the LLM’s schema for this tool. It is the server’s responsibility to validate the input schema and map it to the API calls.

As shown in Figure 3, when a tool needs to be called, the server’s resources can instruct the agent to ask follow-up questions and seek clarification (e.g., by requesting a missing unit of measurement or ensuring it is compatible with the input file).

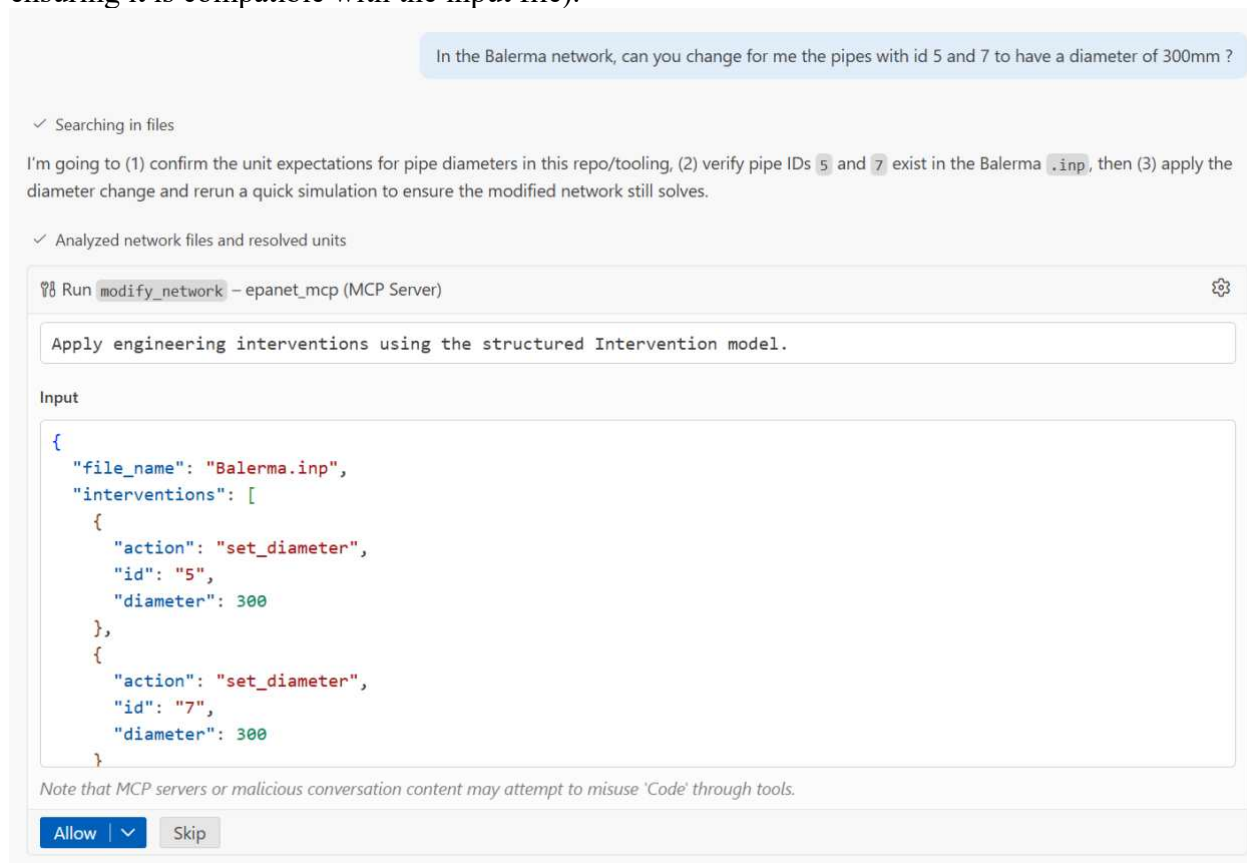


Figure 3: Demonstration of how the EPANET MCP server gives an LLM the ability to modify an EPANET input file following the user request. The LLM is responsible for creating the function-call schema (Input section) based on the user request, while the EPANET MCP server translates this call into the EPANET routines to modify the network model.

4.3 Network Topology and Results Visualisation

If the AI application supports visualisation, EPyT’s plotting functions can be used to create basic visual representations of the water network (see Figure 4). Instead, in Figure 5, we implemented a bar plot to show the pressure across a network’s nodes (Hanoi) for requirements compliance monitoring (e.g., minimum pressure, velocity constraints).

More opportunities are available upon installation by the user of additional specialised MCP servers for complex visualisation tasks. In this case, the EPANET MCP server could return raw results, and the visualisation routines could be offloaded to a dedicated MCP server, removing the constraints of predefined UIs.

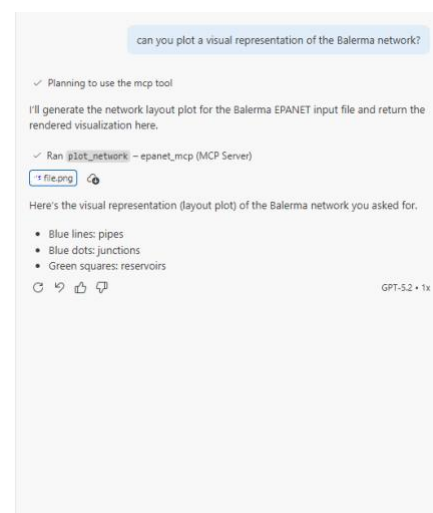
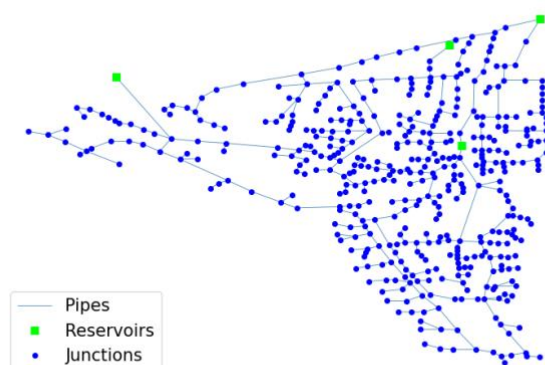


Figure 4: Demonstration of how the EPANET MCP server gives an LLM the ability to visualise an EPANET network model following the user request. This figure shows the Balerma network.

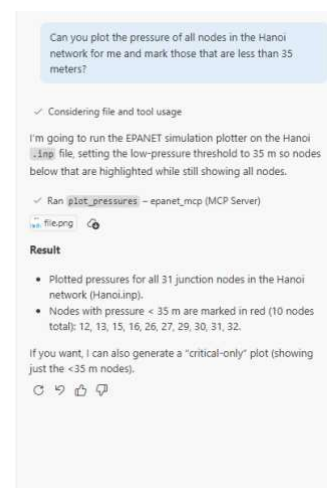
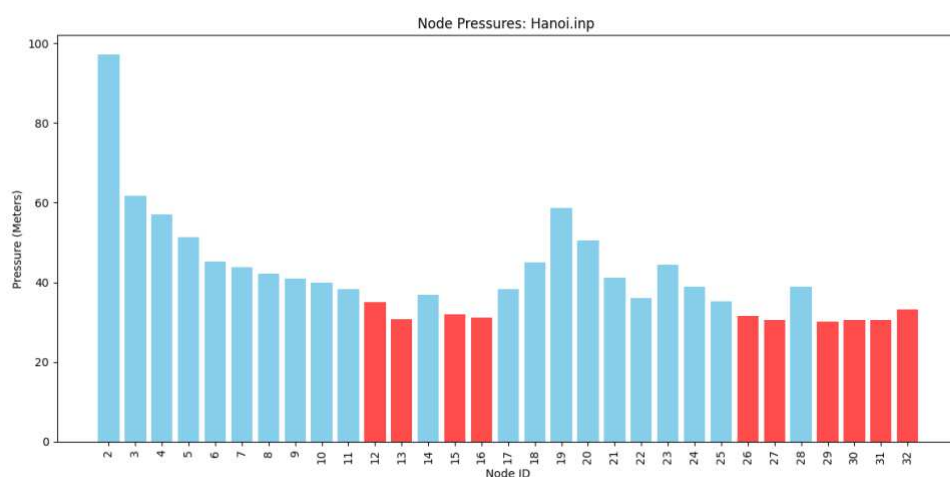


Figure 5: Demonstration of how the EPANET MCP server gives an LLM the ability to display visualisations for requirements compliance monitoring. In this figure, the LLM performs a hydraulic simulation of the Hanoi network and displays the nodal pressures as requested by the user.

5 Discussion

Given that water distribution systems constitute critical infrastructure, data privacy and system security deserve explicit attention. For utilities looking to deploy similar solutions in operational settings, the integration should be assessed in coordination with ICT and cybersecurity departments, as is the case with any new software. This may include, for example, opting for locally hosted AI models or restricting use to cloud providers with whom appropriate data processing agreements are in place.

For researchers and practitioners using this application, a key architectural safeguard is that EPANET input files (.inp) are never transmitted to the external AI provider; all hydraulic simulations execute locally on the user's machine. Nevertheless, users working with sensitive network data should remain aware that simulation outputs surfaced during a session (e.g., a critical node ID), as well as the content of the conversational interface, are subject to the privacy and data retention policies of the chosen LLM provider.

Another significant risk of this technology is that malicious users can employ prompt injection, a technique to manipulate inputs designed to override the AI's safety guardrails or the user's intentions, to compromise system integrity. By tricking the LLM into executing unintended commands, attackers can silently manipulate critical network files or perform unauthorised changes.

For the EPANET MCP server specifically, this means that a lot of attention must be placed in carefully reviewing all contributions to the repository to ensure no harmful logic is introduced. More broadly, anyone working with these technologies should make a habit of critically reviewing any context or external information fed to the model, as unvetted inputs represent the primary attack surface.

6 Conclusions

This work presents the first integration of the EPANET hydraulic engine with the Model Context Protocol (MCP), effectively bridging advanced hydraulic simulation and natural language interaction. Through a series of demonstrative examples, we illustrated the potential benefits for academia and industry. For the former, this tool lowers the entry barrier by removing the need to engage with the complex EPANET programmatic interface. For the latter, the main advantages include the contextual presentation of information to operators via the EPANET MCP server's resources, custom visualisation, and the cognitive offloading of small yet critical steps (as shown in Figure 3, where the LLM confirms with the user the correct unit of measurement).

Although this implementation establishes a robust foundational architecture, it represents only the initial step toward a fully autonomous assistant. Future development should expand the server to fully support EPANET capabilities, allowing users to work on hydraulic models entirely through natural language.

Reference

- [1] T. Wagener *et al.*, “Hydroinformatics education – the Water Informatics in Science and Engineering (WISE) Centre for Doctoral Training,” *Hydrol. Earth Syst. Sci.*, vol. 25, no. 5, pp. 2721–2738, May 2021, doi: 10.5194/hess-25-2721-2021.
- [2] C. Makropoulos and D. A. Savić, “Urban Hydroinformatics: Past, Present and Future,” *Water*, vol. 11, no. 10, p. 1959, Oct. 2019, doi: 10.3390/w11101959.
- [3] E. S. Matrosov, I. Huskova, J. R. Kasprzyk, J. J. Harou, C. Lambert, and P. M. Reed, “Many-objective optimization and visual analytics reveal key trade-offs for London’s water supply,” *J. Hydrol.*, vol. 531, pp. 1040–1053, Dec. 2015, doi: 10.1016/j.jhydrol.2015.11.003.
- [4] L. Banh, F. Holldack, and G. Strobel, “Copiloting the future: How generative AI transforms Software Engineering,” *Inf. Softw. Technol.*, vol. 183, p. 107751, Jul. 2025, doi: 10.1016/j.infsof.2025.107751.
- [5] L. Huang *et al.*, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” *ACM Trans. Inf. Syst.*, vol. 43, no. 2, pp. 1–55, Mar. 2025, doi: 10.1145/3703155.
- [6] X. Tian, I. Vertommen, L. Tsiami, P. Van Thienen, and S. Paraskevopoulos, “Automated Customer Complaint Processing for Water Utilities Based on Natural Language Processing—Case Study of a Dutch Water Utility,” *Water*, vol. 14, no. 4, p. 674, Feb. 2022, doi: 10.3390/w14040674.
- [7] J. S. Ravindran *et al.*, “Application of large language models in developing conversational agents for water quality education, communication, and operations,” *Water Pract. Technol.*, vol. 20, no. 10, pp. 2094–2109, Sep. 2025, doi: 10.2166/wpt.2025.129.
- [8] J. Wang, G. Fu, and D. Savic, “Leveraging large language models for automating water distribution network optimization,” *Water Res.*, vol. 288, p. 124536, Jan. 2026, doi: 10.1016/j.watres.2025.124536.
- [9] P. Anthropic, “Introduction,” *Model Context Protocol*. 2024. Accessed: Nov. 30, 2025. [Online]. Available: <https://modelcontextprotocol.io/docs/getting-started/intro>
- [10] L. Rossman, “Epanet 2 Users Manual,” in *Cincinnati US Environmental Protection Agency National Risk Management Research Laboratory*, vol. 38, 2000.
- [11] M. S. Kyriakou, M. Demetriades, S. G. Vrachimis, D. G. Eliades, and M. M. Polycarpou, “EPyT: An EPANET-Python Toolkit for Smart Water Network Simulations,” *J. Open Source Softw.*, vol. 8, no. 92, p. 5947, Dec. 2023, doi: 10.21105/joss.05947.
- [12] D. Zanutto and C. Michalopoulos, *The EPANET MCP server [Software]*. (Nov. 30, 2025). Zenodo. doi: 10.5281/ZENODO.17771952.