COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MASTER

DEVELOPMENT OF AN INTERACTIVE WEB-BASED PLATFORM FOR ENERGY NETWORK TRACE ANALYSIS AND VISUALIZATION

Autor: Álvaro Lastra Aragoneses

Director: Bruce Stephen

Director Industrial: Ciaran Higgins

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

DEVELOPMENT OF AN INTERACTIVE WEB-BASED PLATFORM FOR ENERGY NETWORK TRACE ANALYSIS AND VISUALIZATION

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Álvaro Lastra Aragoneses Fecha: 14/ 8/ 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Bruce Stephen Fecha: …14…/ …08…/ …2025…

Ciaran Higgins

Fdo.: Ciaran Higgins Fecha: …15…/ …08…/ …2025…

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D._____Álvaro Lastra Aragoneses_____

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: DEVELOPMENT OF AN INTERACTIVE WEB-BASED PLATFORM FOR ENERGY NETWORK TRACE ANALYSIS AND VISUALIZATION, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Glasgow, a 14 de agosto de 2025

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MASTER

DEVELOPMENT OF AN INTERACTIVE WEB-BASED PLATFORM FOR ENERGY NETWORK TRACE ANALYSIS AND VISUALIZATION

Autor: Álvaro Lastra Aragoneses

Director: Bruce Stephen

Director Industrial: Ciaran Higgins

Madrid

Agradecimientos

Gracias a mi familia por su apoyo y confianza.

DESARROLLO DE UNA PLATAFORMA WEB INTERACTIVA PARA EL ANÁLISIS Y LA VISUALIZACIÓN DE TRACES EN REDES ENERGÉTICAS

Autor: Lastra Aragoneses, Álvaro.
Director: Stephen, Bruce.
Entidad Colaboradora: Scottish Power Energy Networks (SPEN)

ABSTRACT

Esta tesis presenta el NAVI Trace Toolkit, una plataforma web desarrollada para SPEN con el objetivo de modernizar el flujo de trabajo de análisis de traces. Extiende el marco teórico de Hoel et al. [1] con un algoritmo que es usado para calcular el camino crítico de instalación. La plataforma mejora tanto la velocidad de ejecución como la eficiencia del usuario.

Palabras Clave: Análisis de Traces, Visualización de Redes Eléctricas, Desarrollo Web

1. Introducción

Scottish Power Energy Networks (SPEN) emplea la plataforma NAVI para visualizar su red energética y respaldar la toma de decisiones técnicas, siendo el análisis de traces una de sus herramientas más potentes. No obstante, SPEN carece de un entorno específico para el desarrollo de traces, dependiendo exclusivamente de scripts en Python que limitan la flexibilidad y ralentizan el proceso iterativo. Esta tesis presenta el NAVI Trace Toolkit, una solución web que permite a los ingenieros desarrollar traces de forma ágil y eficiente. El proyecto integra ingeniería de software y un diseño centrado en el usuario para ofrecer una plataforma práctica, escalable y orientada a las necesidades reales del equipo de desarrollo.

2. Metodología

La metodología adoptada para este proyecto se basa en principios agile y en un enfoque centrado en el usuario. El proceso se estructuró en cuatro fases: recopilación de requisitos, diseño de wireframes, desarrollo iterativo y evaluación final.

La investigación inicial permitió identificar los principales puntos críticos en el flujo de trabajo de desarrollo de traces en SPEN, como la ausencia de interfaz gráfica, baja accesibilidad y una arquitectura monolítica.

Los wireframes se diseñaron en Figma con el objetivo de responder a las necesidades del usuario en cuanto a la interfaz visual.

Durante la fase de desarrollo iterativo se incorporaron diversas sugerencias, como mejoras en la estética de la interfaz, persistencia de sesión y abstracción de la lógica de traces.

Una vez completada la plataforma, se llevó a cabo una evaluación con cinco ingenieros de SPEN, quienes realizaron tareas predefinidas y participaron en encuestas y entrevistas abiertas para valorar la experiencia de uso y compararla con el flujo de trabajo anterior.

3. Implementación

El NAVI Trace Toolkit se ha implementado como una aplicación web modular, adaptada al entorno operativo de SPEN. El backend se desarrolló en Python utilizando el framework

Flask, mientras que el frontend se construyó con HTML y CSS. Para la representación geoespacial se emplea MapLibre GL JS. Las subredes se presentan mediante capas de MapLibre con fuentes en GeoJSON, y la lógica de traces se programa directamente en la interfaz de usuario a través del editor Ace integrado.

El backend expone endpoints RESTful para la ejecución de traces, la carga de subredes y el acceso a la guía de usuario. Además, incorpora una biblioteca personalizada de traces basada en el marco teórico de Hoel et al. [1], que además incluye funciones de estilo de traces, mensajes, utilidades y exploración de redes. Entre las funciones de exploración de redes destaca la de camino crítico diseñada en este proyecto.

La clase TraceContext gestiona el estado del grafo y el estilo de visualización, lo que permite flujos de trabajo modulares y la aplicación de lógica de traces en múltiples pasos. La instalación se ha hecho más intuitiva mediante scripts .bat y una guía de usuario, garantizando la accesibilidad para ingenieros con distintos niveles técnicos.

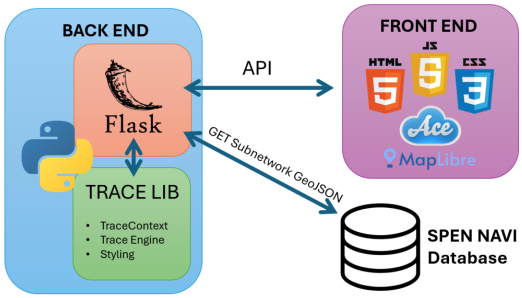

Figure 1 – Arquitectura del Sistema de NAVI Trace Toolkit

4. Resultados

El NAVI Trace Toolkit fue evaluado mediante un análisis comparativo, pruebas de usuario y un caso de uso realista, con el objetivo de validar tanto su rendimiento como su usabilidad. La plataforma ofrece una interfaz rica e interactiva, que incluye visualización geoespacial, estilos dinámicos y ejecución de traces en tiempo real.

4.1. Análisis Comparativo

En comparación con el anterior flujo de trabajo monolítico, la nueva plataforma mejora de forma significativa los tiempos de ejecución de traces y descarga de subredes (3,92 segundos frente a 10,73 segundos), incluso al gestionar hasta nueve subredes simultáneamente. Su arquitectura modular permite un desarrollo de traces más eficiente, rápido y flexible.

4.2. Evaluación del Usuario

Cinco ingenieros de SPEN participaron en una prueba de usabilidad, para completar la instalación de la plataforma y cinco tareas definidas en menos de 30 minutos. El NAVI Trace Toolkit alcanzó un 100 % de efectividad y una eficiencia de 1,64 tareas por minuto. El tiempo medio de instalación fue de 5,05 minutos, en contraste con la versión anterior que tomaba más de una hora.

Para evaluar la experiencia de usuario, se realizó una encuesta tipo Likert (escala 1–5) que abarcó nueve dimensiones de usabilidad. Todas las dimensiones tuvieron puntuaciones

superiores a 4 de media. Los participantes destacaron especialmente la claridad de la interfaz y la utilidad de las funciones integradas. Las sugerencias de mejora incluyeron la incorporación de un módulo introductorio para nuevos usuarios, funciones de importación/exportación de scripts y pequeños ajustes en la interfaz, como la incorporación de zoom automático y que el panel de código que se pueda redimensionar.

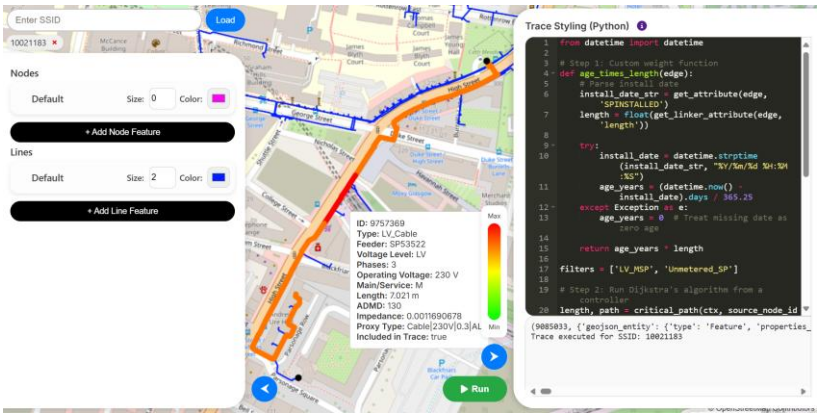

Figura 2 – La IU de NAVI Trace Toolkit con el Caso de Uso CIC

4.3. Caso de Uso: Camino de Instalación Crítico (CIC)

El caso de uso CIC demuestra la capacidad del NAVI Trace Toolkit para evaluar la vulnerabilidad de infraestructuras envejecidas usando el análisis de traces. Se implementaron dos enfoques: uno basado en DFS, que explora todos los caminos simples, y otro más escalable basado en el algoritmo de Dijkstra. Este segundo enfoque utiliza la función critical_path(), desarrollada en el marco de esta tesis, que calcula los caminos más cortos desde un nodo controlador hasta todos los clientes, seleccionando el más expuesto según el valor CIC. El método tiene en cuenta principios de las redes eléctricas como la jerarquía y la tolerancia a fallos, y reduce la complejidad computacional a O(E + J log J), lo que lo hace adecuado para redes de gran escala. La implementación demuestra la capacidad de la plataforma para realizar análisis de traces escalables, reforzando su utilidad como herramienta técnica en entornos operativos reales.

5. Conclusión

El NAVI Trace Toolkit moderniza con éxito el desarrollo de traces en SPEN, reemplazando flujos de trabajo desactualizados basados en scripts por una plataforma web modular. Integra ejecución en Python en tiempo real, visualización geoespacial y funciones integradas como el algoritmo critical_path() desarrollado en esta tesis. Las pruebas de rendimiento y la evaluación con usuarios confirmaron mejoras significativas en velocidad, usabilidad y accesibilidad. El caso de uso CIC validó la capacidad analítica y la escalabilidad de la plataforma, posicionándola como una solución práctica para el análisis de redes energéticas e infraestructuras.

6. Referencias

D. Oliver and E. G. Hoel, “A Trace Framework for Analyzing Utility Networks: A Summary of Results (Industrial Paper),” Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 249-258., 2018.

DEVELOPMENT OF AN INTERACTIVE WEB-BASED PLATFORM FOR ENERGY NETWORK TRACE ANALYSIS AND VISUALIZATION

Author: Lastra Aragoneses, Álvaro.
Supervisor: Stephen, Bruce.
Collaborating Entity: Scottish Power Energy Networks (SPEN).

ABSTRACT

This thesis presents the NAVI Trace Toolkit, a web-based platform developed for SPEN to modernise trace workflows. It extends Hoel et al.'s framework [1] with a new critical path function and validates its performance through a realistic use case. Compared to the previous implementation, the platform improves execution speed and delivers high user satisfaction.

Keywords: Trace Analysis, Energy Distribution Networks, Web, Geospatial visualisation

7. Introduction

Scottish Power Energy Networks (SPEN) uses the NAVI platform to visualise the energy network and support engineering decisions, with trace analysis being one of its most powerful tools. Despite its potential, SPEN lacks a dedicated trace development environment, relying on manual Python scripts that limit flexibility and slow iteration. This thesis develops the NAVI Trace Toolkit, a web-based solution that enables engineers and data scientists to develop and manage traces efficiently. The project combines energy network analysis, software engineering, and user experience design to deliver a practical and scalable platform.

8. Methodology

The methodology followed a user-centred and agile approach, structured into four phases: requirements gathering, wireframing, iterative development, and stakeholder feedback.

Initial research identified key pain points in SPEN's trace development workflow, such as a lack of real-time feedback, poor accessibility, and a monolithic architecture. These insights shaped the functional requirements of the NAVI Trace Toolkit, including modular Python execution, UI-based trace configuration, and real-time map interaction.

Wireframes were designed in Figma to align user needs with visual layout, and agile sprints ensured continuous refinement based on developer and stakeholder input.

Stakeholder feedback during agile development led to improvements in key UI aesthetics, responsiveness, session persistence, and trace logic abstraction.

Once core functionality was complete, the platform was tested by five SPEN engineers through task-based performance tests, surveys, and open feedback. Results confirmed the toolkit's usability, efficiency, and compatibility with SPEN's corporate environment, validating its practical value.

9. Implementation

The NAVI Trace Toolkit was implemented as a modular web application tailored for SPEN's operational environment. It uses a Flask Python backend and a responsive HTML/CSS frontend, with MapLibre GL JS for map rendering. Subnetworks are visualised using GeoJSON layers, and trace logic is executed in real time via a built-in Ace Editor.

The backend exposes RESTful endpoints for trace execution, subnetwork loading, and user guidance, and integrates a custom trace library based on Hoel et al.'s framework [1]. This library includes functions for trace styling, messaging, utilities, and network exploration, including the critical path algorithm designed in this project. The TraceContext class manages graph state and styling, enabling multi-step workflows and modular trace development. Setup is simplified through .bat scripts and a user guide, ensuring accessibility for engineers regardless of their technical background. The implementation prioritises usability, modularity, and compatibility with SPEN's infrastructure, supporting efficient trace development and visualisation.

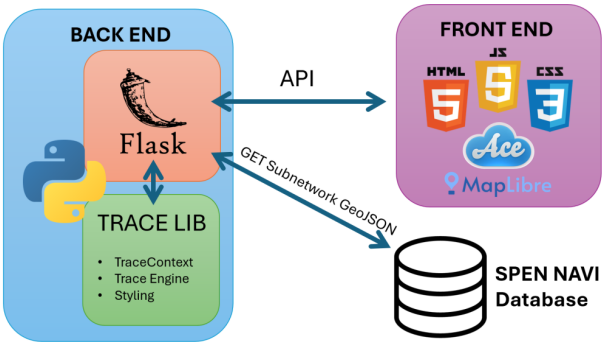

Figure 3 – System Architecture of the NAVI Trace Toolkit

10. Results

The NAVI Trace Toolkit was evaluated through comparative analysis, user testing, and a realistic use case to validate its performance and usability. The platform delivers a rich, interactive interface with geospatial visualisation, dynamic styling, and real-time Python scripting. It supports multi-subnetwork analysis, responsive design across desktop resolutions, and intuitive user flows.

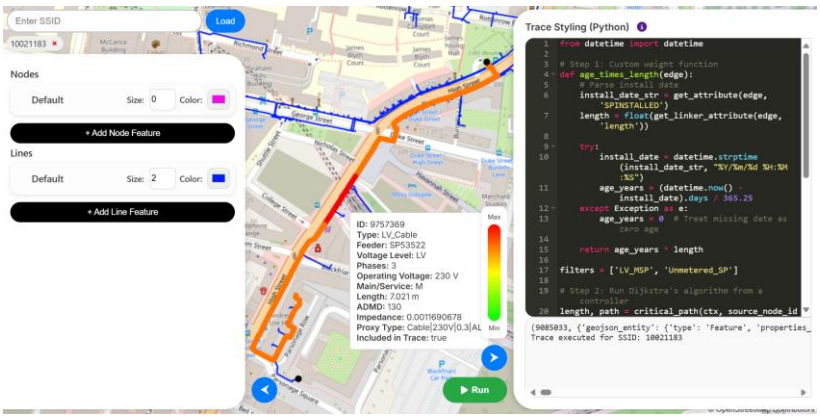

Figure 4 – The NAVI Trace Toolkit UI Showing Critical Installation Path (CIP) Trace

10.1. Comparative Analysis

Compared to the previous monolithic workflow, the toolkit significantly improves trace execution speed and subnetwork loading time (3.92s vs. 10.73s), even when handling up to nine subnetworks. The modular architecture enables flexible trace development, while built-in functions and integrated documentation enhance accessibility.

10.2. User testing

A user study with five SPEN engineers evaluated the NAVI Trace Toolkit's usability, efficiency, and user experience. Participants completed setup and five trace-related tasks within 30 minutes. The toolkit achieved 100% effectiveness (all tasks completed) and an efficiency score of 1.64 tasks per minute, confirming its intuitive design and fast task execution. Setup time averaged 5.05 minutes, a significant improvement over the previous version, which took up to an hour.

A Likert-scale survey (1–5) assessed nine usability dimensions, all scoring above 4. Users highlighted the toolkit's clarity, responsiveness, and built-in trace functions. Open feedback suggested future improvements, including onboarding new users, import/export features for trace scripts and data, and UI enhancements like initial zoom-to-fit and a resizable code panel.

10.3. Use Case: Critical Installation Path (CIP)

The CIP use case demonstrates how the NAVI Trace Toolkit can assess ageing infrastructure in energy networks. Two approaches were implemented: a DFS-based method that explores all simple paths to find the highest cumulative CIP weight, and a more scalable Dijkstra-based method. The second approach uses the custom critical_path() function developed in this thesis, which computes shortest paths from a controller node to all customers and identifies the most exposed one based on CIP. This method respects energy network principles like hierarchy and fault tolerance, and reduces complexity to O(E + J log J), making it suitable for large-scale analysis. The implementation highlights the toolkit's support for scalable algorithms, attribute-driven analysis, and operational relevance.

11. Conclusions

The NAVI Trace Toolkit successfully modernises trace development at SPEN by replacing rigid, script-based workflows with a modular, web-based platform. It integrates real-time Python execution, geospatial visualisation, and built-in trace functions, including the custom critical_path() algorithm developed in this thesis. Performance tests and user evaluations confirmed improved speed, usability, and accessibility. The CIP use case validated the toolkit's analytical capabilities and scalability, making it a practical solution for energy network analysis and infrastructure assessment.

12. References

[1] D. Oliver and E. G. Hoel, "A Trace Framework for Analyzing Utility Networks: A Summary of Results (Industrial Paper)," Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 249-258., 2018.

ABSTRACT

This thesis presents the development of the NAVI Trace Toolkit, a web-based platform designed to modernise trace analysis workflows within energy distribution networks. Trace analysis is a powerful technique to visualise and diagnose the structure and behaviour of electrical grids. However, the lack of a dedicated trace development environment has limited its implementation at Scottish Power Energy Networks (SPEN). Existing trace development workflows rely on executing manual Python scripts, which lack real-time feedback, flexibility, and accessibility for trace developers.

The NAVI Trace Toolkit integrates energy network analysis with modern web software development and a user-centred design approach to address these limitations. Built using Python, Flask, HTML5, CSS3, and JavaScript, the platform provides an interactive interface that supports real-time trace execution, dynamic styling, and modular architecture. It allows the developers to load multiple subnetworks, customise visual outputs, and develop trace logic using built-in functions within a responsive and intuitive environment.

The theoretical foundation of the toolkit is based on Hoel et al.'s trace framework, adapted to energy networks with rich attribute metadata. This project contributes to the trace framework by integrating a new function for identifying critical paths using a custom weight function.

The platform leverages GeoJSON for geospatial data encoding and integrates MapLibre GL JS for high-performance map rendering. The Critical Installation Path (CIP) use case demonstrates a DFS-based and a scalable Dijkstra-based approach, balancing analytical depth with computational feasibility. This realistic use case validates the platform's capabilities and illustrates a practical application of the critical path function.

Industrial constraints were considered to ensure compatibility with SPEN's corporate environment. The application runs locally, requires no administrative privileges, and reduces setup time to under five minutes. A user experience evaluation assessed the platform's efficiency, achieving 1.64 tasks per minute, and effectiveness, reaching 100% task completion. User survey feedback from participants reflected high customer satisfaction.

Índice de la memoria

Índice de figuras

Índice de tablas

1. INTRODUCTION

1.1. BACKGROUND AND MOTIVATION

Scottish Power Energy Networks (SPEN) utilises the NAVI platform to visualise the energy network and provide engineers with insights. Among its various capabilities, one of the most powerful is trace analysis—an analytical tool used to understand, visualise, and diagnose the structure and behaviour of the energy network. This involves highlighting nodes and lines on the network topology to visually represent data, such as tracing upstream from a given impedance to the source. This process helps to determine the network's path, components and electrical characteristics.

Trace Analysis can be extended to various energy network applications; however, SPEN lacks a dedicated trace development platform. The current workflow for trace developers relies on executing basic Python scripts locally, which is time-consuming and does not have real-time feedback. This particularly frustrates users when they want to make minor changes and iterate quickly to refine the code, hindering productivity and experimentation.

This master's thesis aims to create the NAVI Trace Toolkit, a web-based platform that enables data scientists and engineers to develop and manage traces efficiently. This interdisciplinary project combines novel energy network trace analysis approaches with software engineering principles and user experience design to create an innovative and practical application.

This thesis explains the motivation, development, and evaluation of the NAVI Trace Toolkit. Chapter 2 describes the literature review and theoretical background, discussing the growing importance of analytical tools in distribution networks and reviewing traditional approaches

such as GIS tools. This chapter also explores the role of web technologies in smart grids and their user experience design principles. Limitations of existing tools are identified, and the trace theoretical background based on graph theory is described.

The project's methodology is described in Chapter 4, which covers requirements gathering, wireframing, the agile development process, and user testing and evaluation strategy. The following chapter describes the technical implementation, divided into front-end and back-end components.

Chapter 6 presents the project's results, including a visual interface description and an analysis of user flows and execution times. This chapter also reports the results of the user study and a detailed use case demonstration. The conclusion of the thesis is in Chapter 7, which summarises the findings and proposes future work.

1.2. SCOPE AND OUTLINE

The work scope is focused on developing a web-based platform that demonstrates how to build an application using Python and JavaScript to enable trace developers to create complex and specific energy network trace analyses. Although the application is built on SPEN's NAVI platform, it is designed to be replicable, allowing any person or organisation to build the platform for their own trace development needs.

The platform provides a flexible, developer-friendly environment that supports the creation and management of trace logic. The application ultimately bridges the gap between rigid commercial GIS software and overly generic network graph libraries, offering a custom solution for energy network analysis.

Security hardening is considered beyond the scope of the thesis. Although the platform is web-based, it is intended to be executed locally; therefore, advanced security measures are not a priority of this project. As such, server deployment is also excluded from the scope of the project; however, the web design is modular and extensible, allowing for future deployment on a server if required.

2. LITERATURE REVIEW AND THEORETICAL BACKGROUND

2.1. TRENDS AND IMPORTANCE OF ANALYSIS TOOLS IN DISTRIBUTION NETWORKS

The evolution of distribution grids—driven by decentralisation, digitalisation, and decarbonisation—has led to a growing need for advanced analytical tools to support network optimisation and decision-making [1]. These tools are essential for integrating Distributed Energy Resources (DER), Electric Vehicles (EVs), and Smart Grid technologies, while maintaining grid reliability, efficiency, and resilience. [2]

IEEE literature highlights the importance of new analytical tools for tasks such as load forecasting, protection, electric power quality, and power factor improvement [2]. Networks are evolving from passive, radial configurations to more dynamic and bidirectional networks. Traditional manual or static analysis methods may no longer be sufficient.

Furthermore, new privatisation trends and deregulation of the distribution grid have increased the risk of the electrical grid becoming uncompetitive in a new market that can no longer rely on regulatory protection. In this new context, innovative tools are required to remain competitive, and making better use of spatial data is key to achieving efficiency in a capital-intensive market. [1]

2.2. WEB TECHNOLOGIES IN ELECTRICAL GRID

Bui et al. present a new paradigm for energy networks in which the Smart Grid (SG) benefits from internet technologies to become more interoperable and accessible [3]. The paper highlights the importance of referencing internet standardisation bodies such as the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). When implementing web technologies, utilising widely adopted internet protocols such as IP, TCP, and UDP is essential. These protocols ensure SG's compatibility and scalability, for example, when integrating new energy sources and consumers (e.g., solar panels, EVs).

Standard web protocols like HTTP and RESTful APIs enable remote access to grid components and data. The Representational State Transfer (REST) paradigm organises an Application Programming Interface (API) to facilitate data exchange using HTTP operations such as GET, POST, DELETE, and UPDATE [4]. This architecture benefits the SG, as web-based platforms are recognised for their versatility and interoperability across different domains.

The IETF created the Constrained RESTful Environments (CoRE) working group to develop a RESTful protocol for constrained environments, resulting in the Constrained Application Protocol (CoAP) [5]. This protocol facilitates the implementation of the REST protocol in hardware with limited resources, achieving both computational and data efficiency. CoRE's web-oriented binary protocol allows an HTTP request of several tens of bytes to be reduced to 4 bytes. [3]

Other working groups in the IETF and W3C also developed standards to ease the integration of web-based protocols into the smart grid. 6LoWPAN working group efforts were focused on delivering IPv6 internet connectivity to constrained WPAN devices [6]. Meanwhile, the W3C's

Efficient XML Interchange (EXI) worked on XML data compression, achieving up to 90% storage reductions [7].

One of the significant advantages of web technologies is the accessibility of SG services via a web browser, which eliminates the need for software installation. Web technologies also offer interoperability with devices designed for general-purpose networks such as PDAs, mobiles, and servers. [3]

Bui et al. developed a proof of concept for IoT web visualisation developed using Java and Google Maps, in which they present the network topology of connected devices. This work demonstrates the scalability and interoperability of web-based solutions. Similarly, Eren et al. present a web-based dispatcher information system for the electrical grid, YTBS (Yük Tevzi Bilgi Sistemi) [8]. It includes tools for data analysis, forecasting, and long-term investment planning. This real-time, centralised monitoring platform presented by the TSO validates the viability of web-based platforms in grid operations.

In summary, web technologies offer a compelling response to the energy networks' needs. CoAP and data compression standards such as EXI highlight the efforts from the IETF and W3C to enable efficient web technologies implementation in energy grids. At the same time, applications like YTBS and the IoT platform by Bui et al. demonstrate the practical viability of web-based grid management platforms.

2.2.1. USER EXPERIENCE AND HUMAN-CENTRED DESIGN IN WEB APPLICATIONS

User experience is key in adopting and using web-based applications in the energy sector. As distribution networks become increasingly complex and data-driven, the ability of users to interact with digital applications becomes essential. User experience is crucial not only in

usability, but also in improving decision-making, reducing errors, and increasing employee satisfaction [9].

There is a move from human-machine interfaces (HMI) to human-computer interfaces (HCI) [10]. HCI offers more flexibility, capabilities, and interoperability than HMI, satisfying the need for control systems with dynamic interaction and complex data representation on a real-time basis. As the electrical grid continues to grow, manual operation becomes difficult, and new HCI are needed to represent large amounts of data. The electric industry's many stakeholders and business areas have also led to different tools to control, analyse, simulate, and visualise. Each application represents a small part of the energy industry; they gather data from a bigger pool of data sources, and the output of one application might be the input of other business areas, bringing interoperability issues [10].

Visualisation is the representation of the data into information, creating discernible, human-readable patterns. Significant challenges in data visualisation applications include cleaning, organising, and formatting messy data. Open-source repositories and libraries ease the implementation of visualisations. Organisations must focus on building visualisation platforms that are presented simply and appropriately. [11]

Nielsen's usability heuristics provide a framework with principles such as visibility of system status, consistency with standards, match between the system and the real world, and user control, all of which are essential for creating user-centred web applications [12]. In the context of energy networks, we fulfil these principles by enabling real-time feedback on results, consistent visual language, and data fidelity.

Kluge et al. applied a human-centred design (HCD) process to develop a fault-finding mobile application [9]. The work assessed the workers' acceptance of the technology and the

enhancement in fault detection performance, resulting in an improved operational efficiency and detection time [9]. Incorporating feedback and interviewing loops with end users—where users continuously test and refine the tool— ensures that the final product aligns with the changing operational needs. HCD not only improves usability but also strengthens user engagement.

Responsive Web design must also be considered to ensure accessibility across various devices. Furthermore, developing fixed-width websites increases development costs and hinders maintainability [13]. Best practice is to implement responsive web features in the initial steps. Seth A. Blanchard highlights that data analysis tools should also be responsive [11]. In the specific case of the NAVI Trace toolkit, mobile devices have not been considered; however, users might work with different computers and monitor resolutions, which need to be considered in developing the platform.

HTML5 and CSS3, along with the responsive web design methodology, enable web developers to implement the One Web vision [13]. Key elements of responsive design are [13]:

1. Fluid layout uses a flexible grid to be screen-resolution agnostic.
2. Flexible images, whether by themselves or controlled through external logic.
3. CSS3 Media queries, which adapt the design depending on screen resolution and address usability problems.

2.3. TRADITIONAL APPROACHES FOR ENERGY NETWORK TOPOLOGICAL AND SPATIAL ANALYSIS

In the past three decades, topological and network analysis of the grid have been performed using GIS, which private companies typically develop. GIS platforms offer powerful tools for

managing and analysing spatial data, making them ideal for visualising and operating complex energy infrastructures [1].

SPEN adopted the NAVI platform, its GIS application to visualise and operate its distribution network. NAVI supports SPEN in making both business and technical decisions, ultimately leading to more efficient grid management.

Among the various capabilities offered by NAVI, trace analysis attracts attention as a particularly valuable tool. GIS proprietary software platforms, such as ESRI ArcGIS [14] or GE Smallworld [15], often include similar trace tools. Trace analysis is applied across utility networks to better understand their structure and behaviour; in the context of energy networks, typical applications include finding the nearest upstream protective device, calculating voltage drops, and optimising the balance of power flows in the network.

These GIS platforms include trace functions like finding the de-energised features and optimal switch configurations. The ESRI ArcGIS framework is the most advanced tool available for developing traces in proprietary software, which currently has eight built-in functions for tracing utility networks [16]. These trace function algorithms will be discussed in detail in the theoretical background section (2.5).

2.3.1. WEB TECHNOLOGIES FOR GIS

Integrating web technologies into geospatial applications has significantly transformed how networks are visualised and analysed. Web GIS platforms are emerging as a solution in the energy network management to enable dynamic map rendering, collaborative user interaction, and data-driven decision making.

Kuridža's research about the benefits of web GIS applications highlights the advantages of web-based platforms, including cross-platform execution regardless of the operating system, interface with external services, open standards, simplicity, and ubiquity on top of the technologies [17].

GIS web platforms can be categorised into two types [17]. The first type includes web GIS platforms developed by traditional GIS software vendors, such as ArcGIS Online. The second type consists of completely web-based GIS platforms that do not rely on the desktop applications of traditional vendors. These two previous groups can be further divided into two categories: companies that develop applications for GIS end users (e.g., MangoMap, eSpatial) and vendors that focus on building platforms for developers (e.g., MapBox GL, CARTO) [17].

Developers can define their architecture based on the previously mentioned GIS developer platforms, JavaScript open-source libraries (e.g., Turf, Leaflet, OpenLayers), or a combination of those tools. Own developed platforms include server-side data handling and RESTful APIs for communication [17].

One of the most prominent JavaScript libraries is MapLibre GL JS [18], an open-source tool for rendering vector maps using WebGL. MapLibre is a fork of MapBox GL JS, which is a proprietary software for developers that was once open source. It supports GPU-accelerated rendering of vector tiles, allowing for smooth zooming and real-time dynamic styling of map layers. MapLibre is an appropriate solution for many applications, including energy networks, due to its:

1. Custom styling capabilities,
2. Integration with open standards like GeoJSON and Mapbox Style Specification, and

3. Interoperability with frameworks like React, Angular, and maps like MapTiler and OpenStreetMap.

Fournier et al. developed an interactive decision support tool based on web maps for equitable energy planning, which was developed with stakeholders caring about social and environmental justice [19]. This project demonstrates the viability of web technologies for visualising energy networks for DER deployment. The web mapping tool was able to report imbalances between DER supply potential and grid capacity limits, helping in energy transition efforts. The web mapping platform used in this solution was ArcGIS Online, although open-source web-mapping software was considered.

2.3.2. WEB DATA FORMAT FOR GIS

GeoJSON is a widely adopted standard format, defined by the IETF in RFC 7946 [20], for encoding geographical data using JavaScript Object Notation (JSON). Its design focuses on being lightweight, human-readable, and easily parsed by web applications. It supports several types of geometry. Point and LineString are the main types in the energy network context, but it also supports other geometry types such as Polygon, MultiPoint, MultiLineString, and GeometryCollection. The geographic feature has associated metadata, which is crucial for storing electrical data, and coordinates, using WGS84 (EPSG: 4326), which ensures compatibility with most web mapping tools. The JSON format enables smooth integration with REST APIs, databases, and front-end libraries.

2.4. LIMITATIONS OF EXISTING TOOLS

There is currently no solution in the market that offers the flexibility to develop trace analysis tailored to specific cases in the electrical grid, while also being compatible with SPEN's distribution network data format. Existing Python network libraries are too generic to be applied

to energy networks, and commercial Geographic Information System (GIS) software solutions are either too rigid to enable developers to create custom traces or lack a trace analysis tool.

For example, ArcGIS, which has the best trace tool in the market, is still constrained by the user interface, which has no code panel [21]. It does not allow users to combine multiple functions, preventing users from pipelining and creating tailored traces. While the user interface may be accessible to someone unfamiliar with trace development, it might limit developers who want to implement a particular trace logic or require a sandbox to experiment with it.

Commercial GIS platforms are mostly closed-source and license-restricted, hindering their adaptability for research. In many cases, their tracing functions are hardcoded and not easily extensible. Aside from the Oliver and Hoel (Esri) framework [22], there is currently no widely recognised theoretical framework, and it is not currently implemented in an open-source platform where developers can create traces upon those functions.

Additionally, open-source libraries such as NetworkX [23] or Graph-tool [24] provide powerful graph analysis functions, including shortest path, predecessors' retrieval, and cycles detection. Nevertheless, these libraries are too generic for direct application in electrical trace analysis. These libraries lack specific electrical domain applications such as impedance modelling and protective device behaviour. As a result, applying them to an energy network requires large amounts of custom code.

Moreover, SPEN has already developed its trace analysis tool in NAVI, eliminating the need for an external license platform. The NAVI Trace Toolkit addresses these challenges by building on SPEN's existing NAVI infrastructure. This project is based on SPEN NAVI developers' current Python scripts for trace development. Updating the script-based workflow is a logical step to enhance usability, accessibility, and long-term maintainability, since the

current toolkit implementation lacks a unified trace framework and a user-friendly interface. A web-based implementation can provide an intuitive graphical interface that simplifies interaction with trace functions, reduces the learning curve for non-trace developers, and supports modular development.

In conclusion, existing commercial GIS platforms like ArcGIS offer built-in tracing functions; however, they are limited by rigid user interfaces and closed-source architectures. Open-source Python libraries like NetworkX and Graph-tool offer strong capabilities for graph analysis; however, they do not include the specific features needed for trace analysis in energy networks. These limitations prevent SPEN engineers and data analysts from innovating and customising traces.

The NAVI Trace Toolkit has been created to overcome these limitations. This toolkit provides a strategic solution that effectively connects usability with programmability through a flexible framework. It allows engineers to develop, test, and refine trace logic specifically tailored to the unique characteristics of SPEN's distribution network, all without needing external licensed platforms.

2.5. THEORETICAL BACKGROUND: MODELLING ENERGY NETWORKS FOR TRACE ANALYSIS

The theoretical background is based on Hoel et al.'s trace framework for utility networks [22], which is the only formal theoretical framework for trace analysis. Their framework will be adapted to the energy networks domain for the NAVI Trace Toolkit.

The energy network can be formally defined as a graph $U = (J, E)$, where:

- J are the junctions representing physical components such as transformers, switches, fuses, meters or intersection points of lines. Each junction $j \in J$ has a geospatial coordinate (x, y) in a Euclidean plane.
- E are the physical connections between junctions, i.e., the electric lines. Each edge $e \in E$ is modelled as a pair (u, v) connecting junctions u and v.

This graph abstraction allows efficient trace analysis by leveraging graph algorithms.

The model's connectivity defines whether two features are logically connected, regardless of geometric coincidence. For example, a transformer does not directly touch a line but is connected. Two logically connected junctions may not be traversable if, for example, a protective device is open, or a line is disabled due to a fault condition.

Each junction and edge carries a dictionary of network attributes; nullable numeric, string, or Boolean values representing real-world properties such as:

- Physical: length or shape length of the line, conductor material, and cross-sectional area.
- Electrical properties: impedance, reactance, rating in KVA and Amps, operating voltage, and number of phases.
- Logical: entity id, feeder id, predecessors, successors, and circuit hierarchy.
- Simulation and analysis attributes: OpenDSS simulation outputs and ADMD metrics.
- Metadata: unique identifier, installation metadata, financial tracking, and construction details.

Attributes are used to compute metrics for analysis or to control traversability—whether electricity can flow in a path between connected features. For example, the ENABLE line

attribute set to zero is not traversable. The protective device NORMALPOSITION_[A|B|C] attribute indicates if it is closed. If the normal position attribute is set to one, it is closed and traversable; if set to zero, it is open and not traversable. Additionally, if the breakpoin_phase[a|b|c] attribute is false, no breakpoint (i.e., open or fault condition) is present on any phase, thus not blocking traversal. The same attributes work at other junctions, such as switches and fuses, which are also normally closed.

The barriers are specific locations $B \subseteq U$ where traversability must terminate, they are based on physical device types or logical state constraints. Barriers are fundamental for modelling:

- Protection zones: trace operations simulate fault propagation and containment strategies with barriers.
- Islanded operation: Barriers enable the simulation of autonomous network elements that operate independently from the main grid. Trace operations identify the boundaries of islanded zones by terminating at devices configured for disconnection or isolation.
- Maintenance planning: ensures safety for field crews in maintenance operations. Trace analysis can simulate the isolation process to identify which customers and assets are affected.

Finally, the filters are applied after traces are executed; thus, they do not affect traversability. They allow the subnetwork to be discovered. The filters can consist of a set of barrier nodes or categories.

2.5.1. PROBLEM FORMULATION AND STATEMENT

The core problem in trace analysis of energy networks can be formalised as follows:

Given:

- An energy network $U = (J, E)$, where:
  - J is the set of junctions (e.g., transformers, protections, switches, fuses)
  - E is the set of edges (i.e., electric cables or logical connections)
  - Each $j \in J$ and e $\in E$ have associated network attributes (e.g., status, voltage rating, type)
- A set of starting points $s \in S$ from which the trace begins the analysis.
- A set of barriers $b \in B$ that restricts traversability at determined locations.
- A traversability expression T, which stops the traversal based on a barrier function Boolean expression.
- The analysis function type (e.g., upstream, downstream, loops).

Find:

- The subnetwork $U \subseteq U'$ that is reachable from S under the abovementioned constraints and the traversability definition.

2.5.2. TRACE TYPES

The trace framework supports multiple trace types and allows configurable and scalable trace analysis on energy networks. The main trace types developed by Hoel et al. are:

- Shortest path trace, which finds the shortest path between two starting points.
- Loops trace detects loops in the network and helps detect redundancies.
- Subnetwork trace extracts all traversable lines and devices from a subnetwork controller.
  - In distribution energy networks, controllers are the transformers or the substations, the network's energy source.

- A subnetwork $SN \subseteq U$ is defined as a connected subset of the network that includes at least one network subnetwork controller. All junctions and edges in the subnetwork must be traversable, and subnetworks may overlap.

- Subnetwork controller trace, which identifies the controllers of the subnetwork.
- Upstream trace, which finds all controllers supplying power to a location.
- Downstream trace, which identifies all features receiving power from a location. It results from all reachable junctions and edges not belonging to the upstream trace.

2.5.3. TRACE CONFIGURATION EXAMPLE

To configure traces for tailored operations, we must include control over traversability, starting points, barriers, filters, and functions. Here is a brief explanation on how to configure a trace for calculating the grid and the number of customers affected by an electric fault:

- First, we need to define the fault location. That would be the starting point S.
- Then we set the barriers B to include open switches, faulted lines or any device that blocks traversal due to the fault or its state.
- For this operation, we run a downstream trace to identify all the junctions that are no longer reachable due to the fault. It simulates the propagation of the fault and identifies the disconnected areas.
- Finally, a filter is applied to get the junctions representing the customer nodes, and we count the number of customers based on the customer nodes' attributes.

This fault trace example can be applied to many business cases. It can be used for fault response prioritisation depending on the number of high-priority customers (e.g., hospitals). It can improve customer communication and transparency by informing affected customers about expected restoration times and service updates. The fault trace map can help maintenance teams to identify affected areas and prioritise restoration efforts quickly. This trace operation can

simulate the impact of a fault or a planned maintenance, assisting engineers in assessing which customers will be affected.

2.6. THEORETICAL BACKGROUND: DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is a foundational method in graph theory and network optimisation for finding shortest paths from a single node in a network. It is a node labelling and greedy algorithm that only works with non-negative edge weights.

Dijkstra's algorithm progressively selects the node with the smallest tentative distance from the source and updates the distances of its neighbours. Each node is assigned a label consisting of two attributes $\big(d(i),\ p(i)\big)$, where $d(i)$ is the current shortest known distance from the source to the node i, and $p(i)$ is the predecessor node of the current shortest path to the node i.

The step-by-step algorithm for finding the shortest paths from the starting node s to all other nodes in the network is defined as follows [25]:

1. Assign an initial tentative upper bound length to each node. Initially, the source node is assigned a distance of zero $d(s) = 0$, and infinity to the rest of the nodes $d(i) = \infty\ \forall\ i \neq s$. Label node s with $(0, -)$.
2. Assuming c_{ij} is the weight between the two nodes i and j, select the labelled node i with $d(i)$ minimum. Node i is now scanned, and scanned nodes can never be labelled again. For each edge with weight c_{ij}, compute $d(j) = min\{d(j), d(i) + c_{ij}\}$. Each neighbour node of i is marked as labelled.
3. Repeat step 2 until all nodes are scanned, not labelled.

The correctness of the algorithm assumes that all weights are non-negative, meaning that once a node is scanned, the shortest path cannot improve any further.

Dijkstra's algorithm is widely used in routing protocols, transportation systems, and utility networks. Its time complexity is $O\,(E + J \log J)$ when implemented with Fredman and Tarjan's Fibonacci heap priority queue, making it suitable for large-scale graphs [25]. This algorithm is presented because it will be utilised for the critical path function.

3. NAVI TRACE TOOLKIT CONTRIBUTIONS

NAVI Trace Toolkit is a demonstrator platform for trace development in distribution grids, which makes the following key contributions:

3.1. TRACE DEVELOPMENT DEMONSTRATOR

The NAVI Trace Toolkit showcases a proof-of-concept application demonstrating how modern web technologies can be applied to build a trace development environment. It provides a sandbox environment for developers to experiment with trace logic, visualise real-time results, and iterate quickly, addressing significant limitations in the current SPEN workflow.

The platform proves how web technologies (HTML5, CSS3, JavaScript, Python Flask, and MapLibre GL JS) can be used to develop a responsive and real-time trace analysis platform for distribution networks. This new approach lowers the entry barrier for new developers and enhances the productivity for experienced trace developers.

3.2. PYTHON-BASED TRACE FRAMEWORK

The backend of the NAVI Trace Toolkit is built on a custom Python framework inspired by the framework proposed by D. Hoel et al. The framework is extended by:

- Including a new function for finding the critical path.
- Introducing new functions tailored for SPEN's operational needs.
- Supporting modular trace logic, allowing scalability and complex trace logic.
- Specific implementation for electrical utilities.

3.3. UX EVALUATION AND FEEDBACK

A user experience study was conducted with SPEN trace developers to analyse the usability and effectiveness of the NAVI Trace Toolkit. Feedback was collected through task-based testing and user open feedback, providing insights into:

- Interface clarity,
- Ease of trace configuration,
- Effectiveness and efficiency,
- And overall perceived satisfaction.

3.4. USE CASE DEMONSTRATION

A realistic use case was developed in this work to demonstrate the platform's capabilities in a practical scenario. The installation date is used to calculate the subnetwork's Critical Installation Path (CIP) to prioritise maintenance tasks and prevent faults. This trace analysis example proved the correctness and usability of the toolkit, showcasing its value as a trace development tool.

3.5. INDUSTRIAL RELEVANCE

Industrial constraints were considered for its design:

- Local application runs on Python, is accessible through Microsoft Store, and does not require administrative privileges, making it easily deployable in the SPEN corporate environment with strict IT policies.

- Application setup and running in five minutes, considerably reducing the previous setup time and learning curve.
- Business operations are run independently, reducing reliance on external tools.

4. METHODOLOGY

The methodology combines user-centred design, agile development, and testing in a corporate environment with trace developers. The process was divided into four key phases:

4.1. REQUIREMENTS GATHERING AND DOMAIN EXPLORATION

The project began with researching the traces domain and exploring the business necessities. This phase of the project involved:

- Researching trace development: learning about current trace analysis tools in the market and realising there was no platform for trace development.
- Deciding that the NAVI Trace Toolkit would be built on the foundations of the D. Hoel et al. framework.
- Reviewing current trace development workflow in SPEN: understanding current Python scripts and NAVI platform.
- Conducting informal interviews: identifying pain points in the current trace development workflow:

Table 1 - Summary of Key Usability Pain Points Identified During Initial NAVI Trace Toolkit Development

PAIN POINT	DESCRIPTION
MANUAL SCRIPT EDITING FOR REPETITIVE TASKS	Users must modify Python scripts to define traces and set SSID subnetworks. Locating and modifying the correct parameters is often confusing.
LACK OF REAL-TIME UI FEEDBACK	There is no graphical interface with real-time feedback of trace results.
MONOLITHIC ARCHITECTURE	The running shell script executes all Python scripts simultaneously, even when only one task is needed. This is time-consuming and reduces flexibility.
ERROR-PRONE SCRIPTS	Users reported many errors during NAVI Trace initial execution (e.g., curl request failing, map loading issues).
ACCESSIBILITY	Non-developers struggled to engage with trace development due to technical barriers. For example, the shell running script was not executable for Windows users, requiring a Linux terminal environment. Linux terminal environment installation required an IT request, delaying onboarding.

- The pain points identified during the interview phase were translated into platform limitations to address those issues in the development phase.

Table 2 – Summary of Platform Limitations Identified During Initial Development

LIMITATIONS	DESCRIPTION
NO PARAMETER ABSTRACTION	Trace parameters are hardcoded, making customisation difficult.
LACK OF MODULAR EXECUTION	Scripts are tightly coupled, preventing modular execution. Modular architecture solves problems like running several SSIDs simultaneously.
NO BUILT-IN ERROR HANDLING	Poor user error feedback prevents the user from understanding issues. Scripts are not sufficiently tested, providing a poor user experience.
PLATFORM DEPENDENCY	The shell-based execution model is incompatible with Windows, limiting accessibility to most users.
NO VISUAL INTERFACE	Users cannot interact with trace logic and results through UI, reducing efficiency and usability.

This phase helped to define the main general requirements of the platform:

Table 3 – Functional Main Requirements of the NAVI Trace Toolkit

REQUIREMENTS	DESCRIPTION
UI SSID	The application must include a UI input where users can load a subnetwork by introducing its SSID. The application must support several SSID handling.
REAL-TIME EXECUTION AND FEEDBACK	The application must provide real-time feedback when the user runs a trace—for example, notifying the user when trace logic is loading, showing error traceback, or displaying traces on a map when loading is finished.
UI FOR REPETITIVE TASKS	UI must have a section for changing the size and colour of nodes and lines by type. It should also allow the user to configure the features of default nodes and lines.
PYTHON CODE EXECUTION	The application must support real-time Python execution code. It should allow the user to input trace scripts, execute them, and view output (i.e., errors and results).

TRACE FRAMEWORK	The application must provide functions that allow the user to write trace scripts efficiently. It must give comprehensive documentation with examples to help understand the functions.

4.2. WIREFRAMING AND INITIAL PROPOSAL

Once requirements were gathered, wireframes were created to get the application's visual representation and illustrate the intended user flow. These wireframes were presented to the stakeholders, including the manager, for feedback and approval. The wireframes served as a graphic tool to refine scope, align expectations with stakeholders before development began, and as a roadmap to guide the development toward the final product.

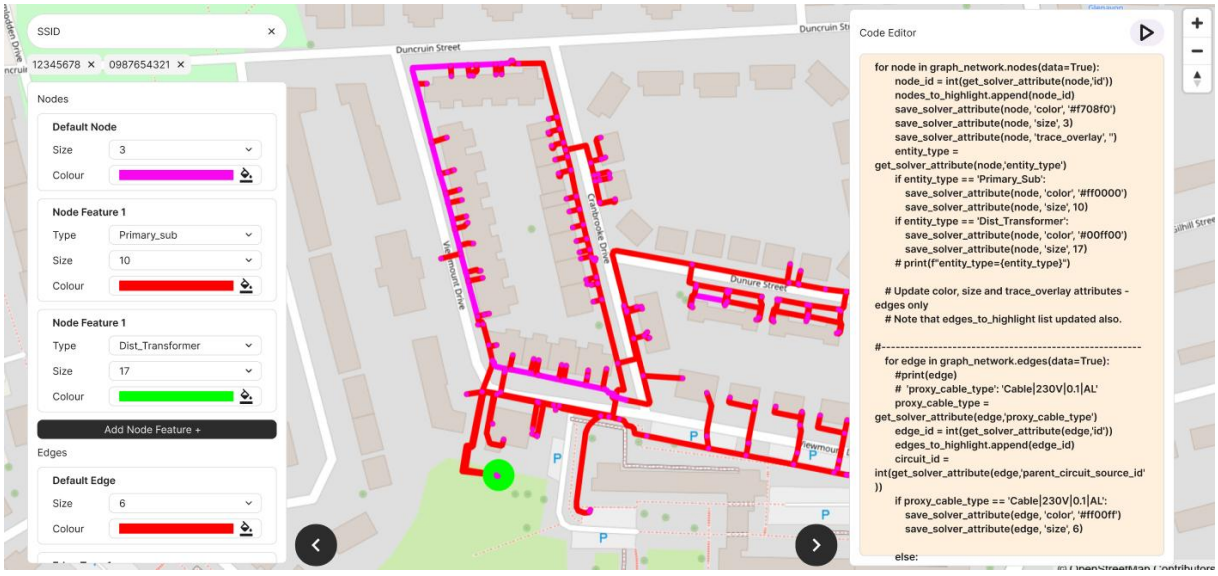

Figure 1 - Wireframe proposal

The wireframe proposal was developed using Figma [26], and the result is shown in Figure 1. This user interface wants to fulfil user requirements while providing a great user experience. Besides the map, the UI comprises four main elements that the user can interact with:

1. Map: The interactive map on the background is the primary visualisation canvas of the traces, enabling intuitive interaction. When the user hovers over the traces, information about the nodes and lines is displayed to facilitate the understanding of the network.
2. Subnetwork input: A search bar is included at the top, where the user can input an SSID to load the subnetwork on the map. The user can load as many subnetworks as they want. A list below the search bar informs the user of the subnetwork SSIDs loaded on the map. The user can remove any subnetwork by clicking its SSID card's cross icon.
3. Features panel: The features panel enable the user to configure basic nodes and lines settings. This panel has a card for setting the default size and colour of the nodes and lines. By clicking “Add Node/Line Feature +”, the user can create as many feature configurations as they want. This panel allow the user to do repetitive tasks with an intuitive UI.
4. Code panel: This panel will include a Python code editor for script editing. Code highlighting and editor functionalities are crucial, since the user needs a smooth coding experience that does not hinder their performance. Code output is also included in this panel.

The buttons positioned along the bottom of the two panels allow the user to hide the panels, providing a bigger and cleaner view of the map. The SSID section remains visible; however, it does not interfere with usability due to its compact size and position.

4.3. AGILE DEVELOPMENT PROCESS

The application's development followed an agile methodology, with 2-week sprint cycles for continuous refinement of features based on feedback. This phase was characterised by regular check-ins, incremental implementation of features based on priority, and updates to the requirements based on user input and technical feasibility. This user-centred approach ensures user involvement on the platform and responds to real user needs. Agile methodology kept the criteria aligned with SPEN's operational context.

The feedback the stakeholders gave during the two months of the development process was:

Table 4 - Stakeholder Feedback and Corresponding Improvements During Agile Development of the NAVI Trace Toolkit

FEEDBACK	DESCRIPTION
IMPROVE UI AESTHETICS	Application components appeared outdated due to limited CSS customisation. UI components were improved by rounding borders, adding shadows, and updating colour schemes.
IMPROVE RESPONSIVENESS	Several errors due to panel hiding were reported, specifically when you change the window size during the same session. These issues were addressed to ensure smooth interaction.

STORE SESSION IN LOCAL STORAGE	Upon closing, the user lost application status, including traces displayed, subnetworks loaded, UI feature configuration, and Python script. Local storage was implemented to preserve session data and restore it upon relaunch.
WRAPPING CODE	Suggestions were made to improve the wrapping of the legacy trace code. These suggestions were solved, resulting in a better trace logic abstraction.
RESIZING CODE PANEL	Users wanted the right-hand code panel to be resizable for an improved coding experience. This enhancement was noted but could not be developed due to time constraints.

4.4. USER TESTING AND EVALUATION

Once the application was fully operative and the core functionality was implemented, the application was tested by five SPEN engineers and trace developers. The evaluation consisted of three categories:

- Quantitative Task-Based Testing: Users were asked to complete tasks (e.g., installing the app, launching it, running a trace), and we recorded their completion times. This test provided valuable insights into the platform's efficiency and effectiveness, based on measurable performance data.

- Quantitative Feedback Survey: A questionnaire with a Likert scale was conducted to assess user experience on usability and performance. They were also asked to report the time spent running the previous Python scripts for trace development.
- Qualitative Feedback: Users provided open feedback on the usability, interface aesthetics, and overall experience. This helped identify the platform's strengths and guided the implementation of future iterations.

The evaluation was conducted on SPEN corporate computers to ensure the platform's operation in corporate settings. This demonstrates that the application runs without administrative privileges and does not require external dependencies beyond Python.

5. IMPLEMENTATION

The NAVI Trace Toolkit was implemented as a modular web application tailored for SPEN's operational environment. It uses a Flask Python backend and a responsive HTML/CSS frontend, with MapLibre GL JS for map rendering. Subnetworks are visualised using GeoJSON layers, and trace logic is executed in real time via a built-in Ace Editor.

The backend exposes RESTful endpoints for trace execution, subnetwork loading, and user guidance, and integrates a custom trace library based on Hoel et al.'s framework. This library includes functions for trace styling, messaging, utilities, and network exploration, including the critical path algorithm designed in this project. The TraceContext class manages graph state and styling, enabling multi-step workflows and modular trace development. Setup is simplified through .bat scripts and a user guide, ensuring accessibility for engineers regardless of their technical background. The implementation prioritises usability, modularity, and compatibility with SPEN's infrastructure, supporting efficient trace development and visualisation.

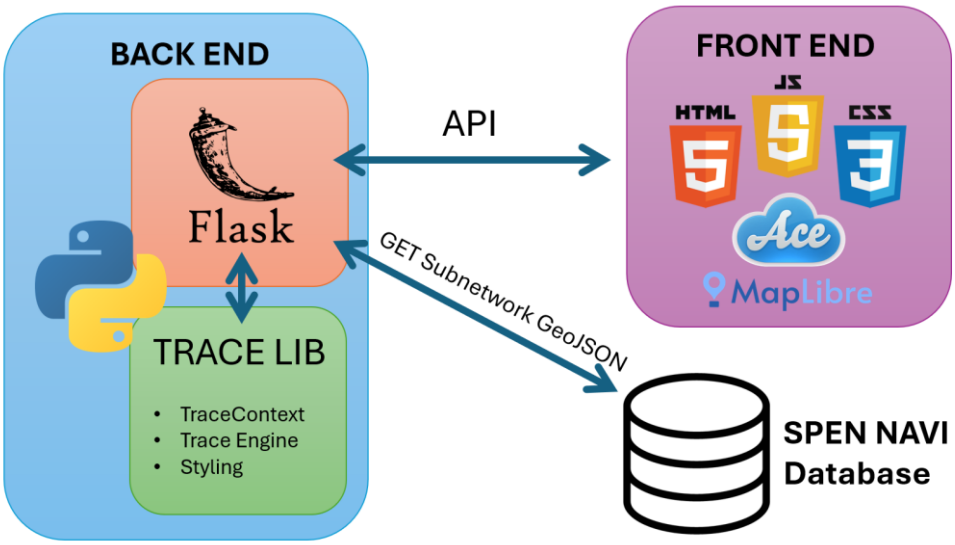

Figure 2 - The NAVI Trace Toolkit System Architecture

5.1. FRONT-END IMPLEMENTATION

The front-end was developed using standard web technologies—HTML and CSS—providing a clean and responsive interface. It allows the user to interact with network data and run trace logic.

Other solutions were explored for map visualisation and user interactivity. One alternative was using Dart, a Python library built on Plotly, enabling interactive web data visualisations. This solution was discarded due to low rendering efficiency when visualising multiple subnetworks and compatibility issues with NAVI.

5.1.1. MAP

The interactive map is implemented with MapLibre GL JS, using OpenStreetMap as the source of map tiles. Traces are displayed on the map using MapLibre GL JS layers. The layers are

rendered using GeoJSON sources; each subnetwork has a GeoJSON associated with it. Two layers are created for each subnetwork: one for junctions, using Point GeoJSON geometry, and another for lines, using LineString GeoJSON geometry. If the user wants to display a message on the map with the Python script, a new layer is created for messages.

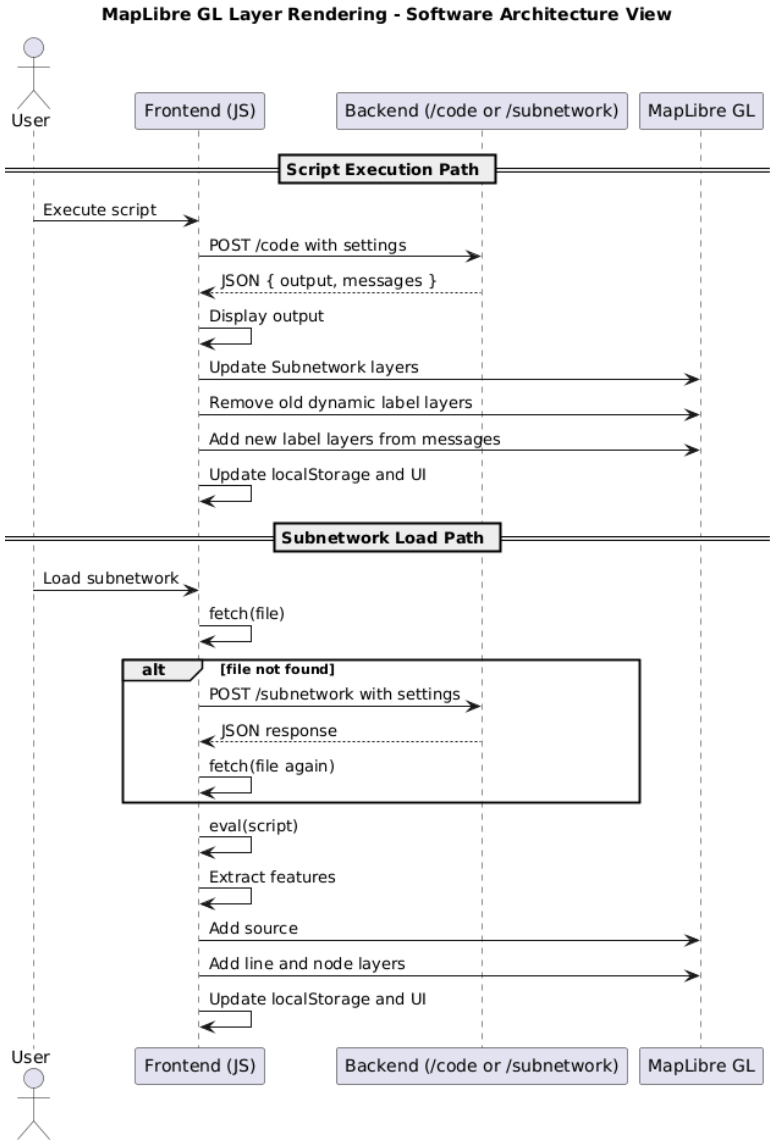

Figure 3 - Display layers on MapLibre GL

Figure 3 illustrates the workflows for loading trace layers on the map. The first workflow occurs when the user executes a script, triggering a POST request to the /code endpoint. The backend processes the script, updating the GeoJSON files with subnetwork properties and returning an output text, which is the output of running the script, and messages, a JSON file with messages that the user might want to display on the map. The front-end refreshes the subnetwork layers with the updated GeoJSON files, removes the previous message layers and updates the message layers.

The second workflow is initiated when the user loads a new subnetwork. First, the front-end checks if the file is already available locally. The front-end sends a POST request to the /subnetwork endpoint if the file is not downloaded. The front-end adds a new source and creates two new layers corresponding to the nodes and lines of the subnetwork.

5.1.2. CODE EDITOR

The code panel is implemented using the Ace Editor, an embeddable, lightweight, and highly customizable code editor written in JavaScript. It provides features commonly found in plain-text native editors, such as Sublime Text or Vim. Key features of the Ace Editor are:

- Syntax highlighting, improving readability and error detection,
- Custom Styling, the Monkai theme was selected for the application, providing a dark background with vibrant syntax colours, improving visual comfort,
- Editing capabilities include line wrapping, line numbering, multiple line editing, and tab spacing control.

5.1.3. OTHER FRONT-END IMPLEMENTATIONS

A hiding functionality was implemented in both panels to enhance user interaction and optimise screen map view, combining CSS transitions, JavaScript, and local storage management (Figure 4). This solution provides a smooth and persistent user experience. The sliding function is encapsulated in a JavaScript function, which selects the element, retrieves the current state, and uses the CSS translateX function to change the panel's position.

To improve data exploration, listeners were added to the map to detect when the user hovers over the lines and nodes of the network. Upon hover, a function that displays the element's information is triggered (Figure 4).

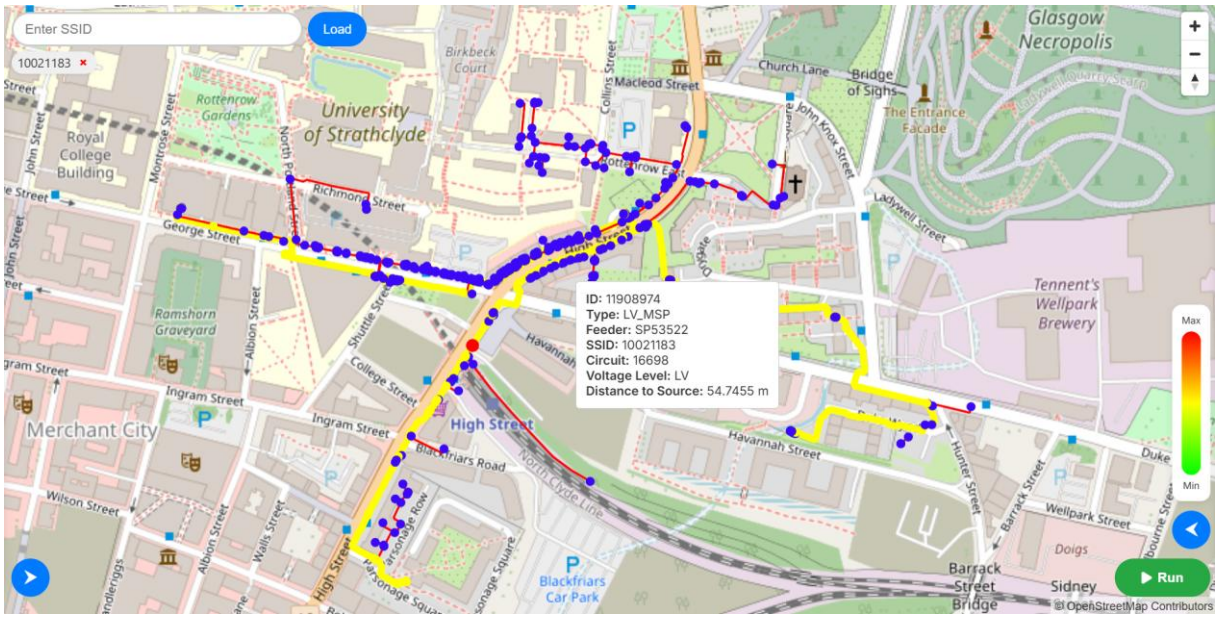

Figure 4 - Panels Hidden with Node Data Displayed on Hover

Local storage is implemented to persist the user-defined configurations and session data:

- Node and lines UI configuration.
- SSIDs of the loaded subnetworks.
- Python script.
- The hiding state of the panels.

This approach retains user preferences and session context, providing a more personalised user experience.

5.2. *BACK-END IMPLEMENTATION*

The back-end system was developed using Flask, a Python framework for web development. The back-end serves as the front-end interface's computational engine and data orchestrator. Its main tasks are handling front-end user requests, parsing from the NAVI API fetched data to the NAVI Trace Toolkit data format, executing trace logic, and generating geospatial outputs for visualisation.

5.2.1. API REST ENDPOINTS

The API exposes multiple RESTful endpoints that support data retrieval and custom trace execution. The endpoints are designed to be stateless and modular, enabling seamless integration with the front-end and scalability. The backend has five endpoints described below:

1. / (Index Route): It loads the main HTML interface using Flask's render_template function. This route does not process data and is only used for initial page load.
2. /subnetwork (GET/POST): This route handles requests to load specific subnetwork data based on its SSID. If the corresponding file does not exist locally, it fetches it on the

NAVI SPEN API, stores it, and processes it using the execute_trace() function. The trace result is formatted to GeoJSON using the create_geojson() function and returned to the front-end for rendering. This endpoint supports code customisation and node and line styling, so that when the user loads the subnetwork, it has the user-configured styling.

3. /code (POST): This endpoint executes user-configured styling and Python code across one or multiple subnetworks. It reads the graph data from local storage and applies the user-configured styling and code using the execute_trace() function. It generates a GeoJSON file for each subnetwork. The output of the user's code execution is captured using Python's contextlib.redirect_stdout, allowing the endpoint to return the code output and trace result. Additionally, messages for map display are handled through helper functions.
4. /delete-ssid (POST): This route removes cached graph and GeoJSON files associated with an SSID. This endpoint handles proper data lifecycle management and maintains a clean work environment.
5. /user-guide (GET): This endpoint dynamically renders a Markdown-based user guide into HTML using the markdown Python library.

Additionally, the backend implements basic error handling mechanisms like validation of the required parameters (e.g., SSID), graceful fallback upon external API failure, and structured JSON error responses to the front-end.

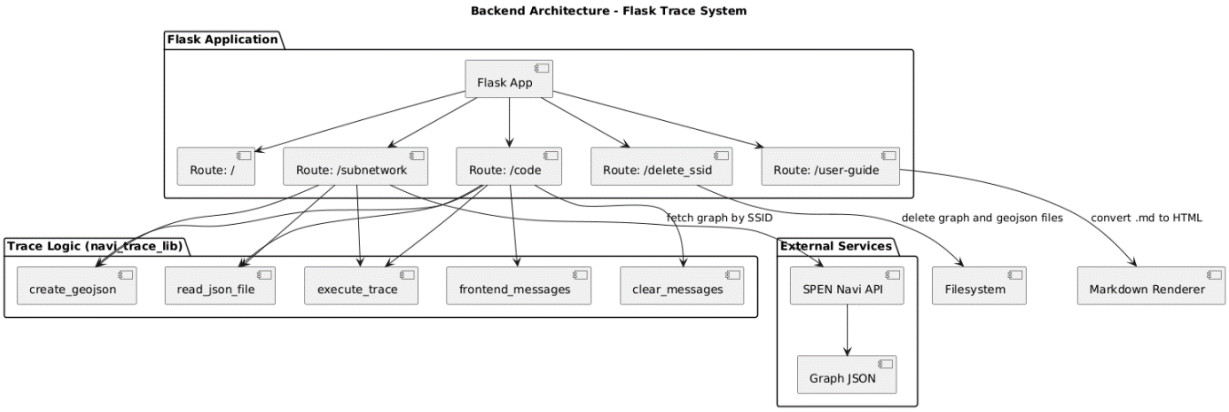

Figure 5 - Backed architecture and Flow

5.2.2. TRACE ENGINE

The backend uses a custom trace library (navi_trace_lib), encapsulating the energy network trace analysis logic. Key helper functions used by the REST API are:

- read_json_file(): Parses from JSON format to NetworkX graph.
- execute_trace(): Executes the trace algorithm based on the user's code and configuration.
- create_geojson(): Converts trace results into GeoJSON format suitable for front-end MapLibre GL rendering.
- frontend_messages and clear_messages(): Manage displayable messages on MapLibre GL.

The trace library includes built-in functions that support advanced energy analysis and visualisation, making the development of new traces more efficient. These functions are based on the theoretical framework proposed by Hoel et al. There are four categories of built-in trace functions:

- Trace Styling: These functions allow users to customise the visual representation of network elements. Examples include:
    - highlight_all_nodes(ctx, color, size): Applies styling to all nodes uniformly.
    - highlight_nodes_by_type(ctx, entity_type, color, size): Applies styling to nodes of a specific type.
    - highlight_nodes_by_attribute(ctx, attr_name, attr_value, color, size): Applies styling to nodes based on custom attributes.
- These functions use helper utilities created for the previous Trace Toolkit Python scripts, like get_solver_attribute() and save_solver_attribute(), to interact with node and edge metadata. Trace styling functions create contextually rich maps, improving visual communication and decision-making.
- Messaging: support user map annotation. The send_message() function lets the user display text labels on the map by setting the coordinates.
- Utilities: Additional functions such as get_color_gradient_continuous() provide gradient colour scheme styling based on data values (e.g., installation date). Other functions like get_node_by_id() and get_entity_id() facilitate entity lookup.
- Network exploration: This category comprises the functions that implement trace algorithms for interactive network exploration. These algorithms support deeper topological analysis and enhance the user's ability to create tailored traces. The trace() function serves as the unified interface that allows the user to run the trace algorithms grounded in the framework of Hoel et al.:
    - Upstream and downstream tracing.
    - Loop detection.
    - Shortest path computation.
    - Subnetwork extraction and control.

- Additionally, a custom critical_path() function was developed to compute the shortest path from a specified source to each node in the network. This function includes a filter mechanism that allows the user to target one or more entity types. After computing all the shortest paths, the function evaluates the cumulative weight and identifies the maximum total weight among filtered entities. This enables the user to analyse the most critical path based on a custom weight. The pseudo code is defined as follows:

  Input:

  - ctx: TraceContext object with the network graph
  - source_node_id: ID of the starting node
  - weight_func: Function to compute edge weights
  - filters: Set of entity types to filter

  Output:

  - max_distance: Maximum shortest-path weight among filtered nodes
  - path_nodes: List of nodes forming the critical path

Table 5 - Critical Path Function Pseudocode Using Weighted Dijkstra Traversal

```
1  Initialize an empty weighted graph G'

2  For each edge (u, v) in the original graph:
3     Try:
4        Compute weight ← weight_func(u, v, edge_data)
5     Catch error:
6        Set weight ← ∞
7     Add edge (u, v) with weight to G'

8  Run Dijkstra's algorithm on G' from source_node_id
9  Store shortest path distances and paths: lengths, paths
```

```
10 Initialize max_distance ← 0
11 Initialize max_node ← null

12 For each node in the original graph:
13     If node.entity_type ∈ filters:
14         If node is reachable (node ∈ lengths):
15             If lengths[node] > max_distance:
16                 Update max_distance ← lengths[node]
17                 Update max_node ← node

18 Retrieve path_nodes ← paths[max_node]

19 Return (max_distance, path_nodes)
```

These functions enhance a clean and expressive trace library, enabling users to concentrate on analysis instead of low-level implementation specifics.

Central to this implementation is the TraceContext class, an essential component of the trace library. It is a simple, stateful container for managing trace operations and styling. This object encapsulates the current graph and the list of nodes and edges to be highlighted during trace operations. Maintaining a persistent reference to the graph facilitates multi-step trace workflows, where intermediate steps can be styled.

This design facilitates a structured and modular workflow. For example, the result of executing the 'upstream' function can be stored in a new TraceContext, allowing the user to isolate and manipulate this subgraph independently.

This library's modular design decouples the trace logic from the back-end API and front-end, enhancing reusability and testability.

5.2.3. LIBRARIES

The development of the NAVI Trace toolkit relies on Python libraries that were carefully selected to support its capabilities. The Python libraries used for developing the application are:

- Markdown [27]: Markdown is a lightweight language developers use to write documentation using plain-text editors. The NAVI Trace Toolkit uses the Python Markdown library to render Markdown plain text to HTML through the /user-guide endpoint.
- Requests [28]: This project uses HTTP requests to external APIs to retrieve the network graph from the SP Energy Networks NAVI API. It provides reliable GET and POST request functions, including error handling and response parsing methods.
- Ujson [29]: Ultra JSON is a parsing library that reads and writes JSON files efficiently. It provides faster serialisation and deserialization than the built-in JSON module. The NAVI Trace Toolkit use this module to manipulate the network's GeoJSON files.
- Networkx [23]: A library for creating, manipulating, and analysing networks. The toolkit uses its data format to store the subnetwork data, and the trace algorithms are implemented using networkx's transversal utilities—such as loopfinding, shortest path, and graph transversal.
- Flask [30]: The core web framework used to build the back-end of the NAVI Trace Toolkit. Flask offers routing, request handling, endpoint exposure, and HTML rendering capabilities.

5.3. SETUP AND APPLICATION LAUNCHER

Two .bat files were created to prevent users from manual command-line interaction, which raises the technical bar for non-developer users. These files enable users to install and launch the NAVI Trace Toolkit through the Windows operating system's interface. The first file, install.bat, creates a new Python environment and installs all the necessary libraries. This step is crucial for maintaining an isolated and consistent library version, which helps prevent compatibility issues arising from system-wide Python configurations.

The second file, run.bat, launches the application within the previously created Python environment, ensuring that the toolkit operates with the correct dependencies.

A comprehensive guide was developed to support the user through the setup process. This guide includes step-by-step instructions for installing Python, executing the install.bat script to configure the environment, and launching the NAVI Trace Toolkit using the run.bat. This guide aims to make the toolkit accessible to all engineers regardless of their technical background.

6. RESULTS

A comparative analysis has been performed to evaluate the effectiveness and improvements of the NAVI Trace Toolkit compared to previous scripts for trace development. A user evaluation was also conducted to gather feedback and assess the application's intuitiveness. Finally, the Critical Installation Path (CIP) use case demonstrates the platform's correctness and capabilities.

6.1. VISUAL RESULT AND INTERFACE DESCRIPTION

The NAVI Trace Toolkit provides a rich, interactive interface (Figure 6) that comprises geospatial visualisation, trace styling configuration, and trace scripting capabilities. The visual output includes the main components designed in the wireframe phase: the map display, the SSID loader, the feature sidebar, and the code editor panel.

The central element of the interface is the dynamic map rendered using MapLibre GL. The map visualises the energy network, with nodes and edges styled according to parameters defined by the user.

On the left side of the interface, a configuration panel enables the user to define the properties for nodes and lines. In Figure 6, default nodes are turned off by setting their size to 0, while transformers are highlighted in black. Lines are displayed in blue. This styling capability enhances the clarity of the network representation and aids in exploratory analysis.

Above the styling panel, the user finds the SSID loader to display multiple subnetworks. The SSIDs of these subnetworks are shown below the SSID input. This feature enables multi-network and comparative analysis, improving analytical flexibility.

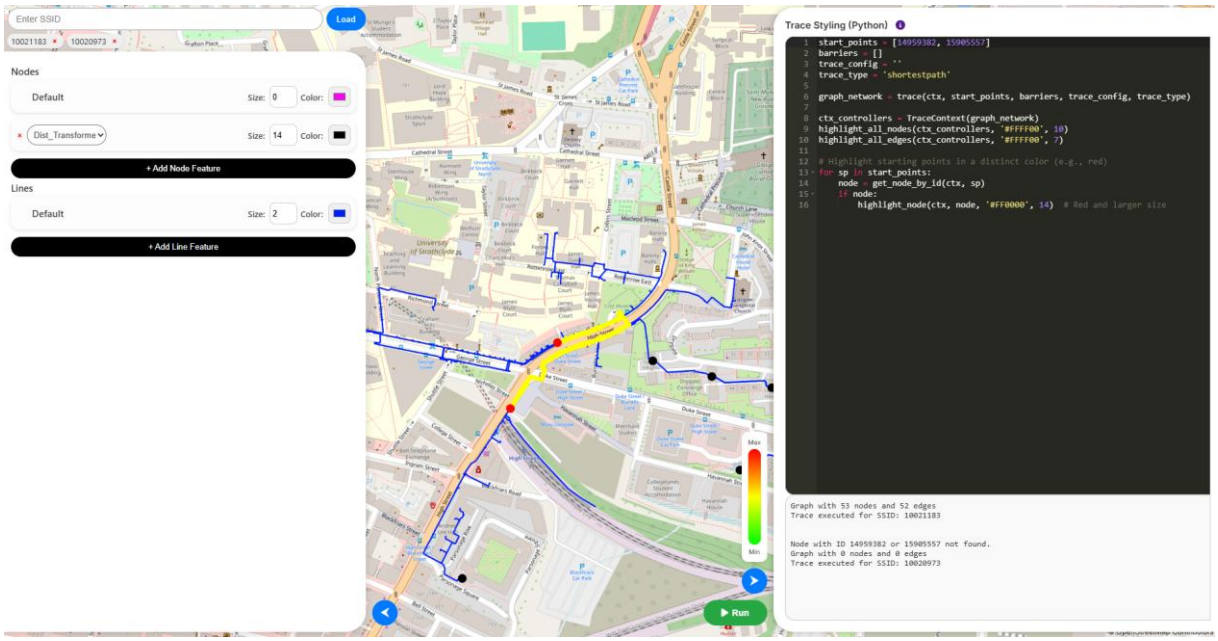

Figure 6 - The NAVI Trace Toolkit Application

The right side of the interface features the Python code editor, where the user can write and execute trace scripts. The Python editor supports the built-in functions developed in this project for tailored trace development—such as trace(), highlight_node(), and TraceContext (). The user can execute traces, providing real-time updates directly on the map by pressing the run button next to the right panel. In Figure 6, a shortest path trace is executed between two nodes, with styles applied to the results. Below the editor, the user can find UI feedback, including trace outputs and error messages (e.g., missing nodes).

6.1.1. RESPONSIVE ACROSS COMPUTER AND MONITOR DISPLAYS

The user interface is responsive across screen resolutions and varying window sizes, which is a key feature for usability in web-based applications. Given that the toolkit is intended for desktop only, not for mobile devices, the design has been tested for standard monitor resolutions (ranging from 1280 x 720 (HD) to 1920 x 1080 (Full HD)). Additionally, the application has been tested for different window sizes (Figure 7).

The layout showed resolution and size responsiveness indicators such as consistent element positioning, no overlapping panels, and no scroll overflow. This test demonstrates effective responsive UI, confirming the application is well-suited for desktop environments with varying resolutions.

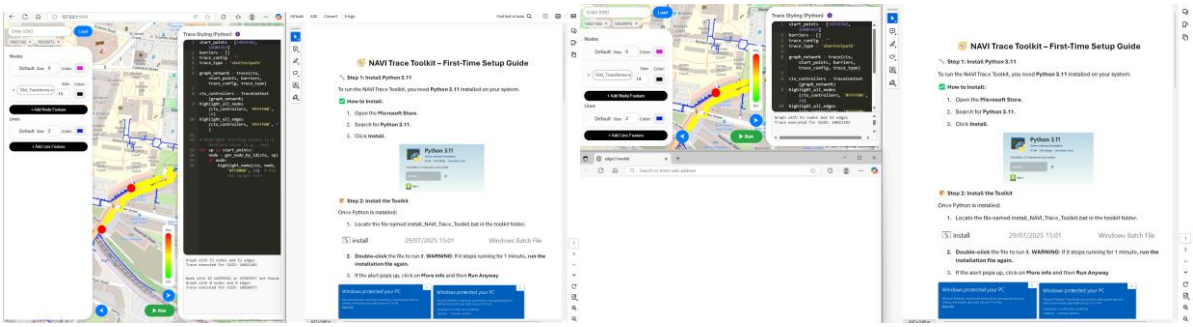

Figure 7 - Multiple Window Sizes Responsive test

6.2. APPLICATION COMPARISON

6.3. USER FLOWS

The NAVI Trace toolkit has significantly enhanced user flows compared to the previous implementation. The system now supports a modular, interactive, and user-friendly interface that enables users to perform more flexible and efficient trace analysis.

6.3.1. CURRENT USER FLOW

The NAVI Trace toolkit's current flows allow the user to:

- Load subnetworks individually by selecting the SSIDs, enabling the user to select multiple subnetworks within a single session.
- Modify node and line features dynamically, depending on their attributes and entity types.
- Run Python scripts using the code panel and the built-in functions to execute trace logic and apply styling.
- Refer to the markdown documentation to understand how to use the functions.

All these features are integrated into a web interface that instantly displays results, without needing to refresh or reload external files.

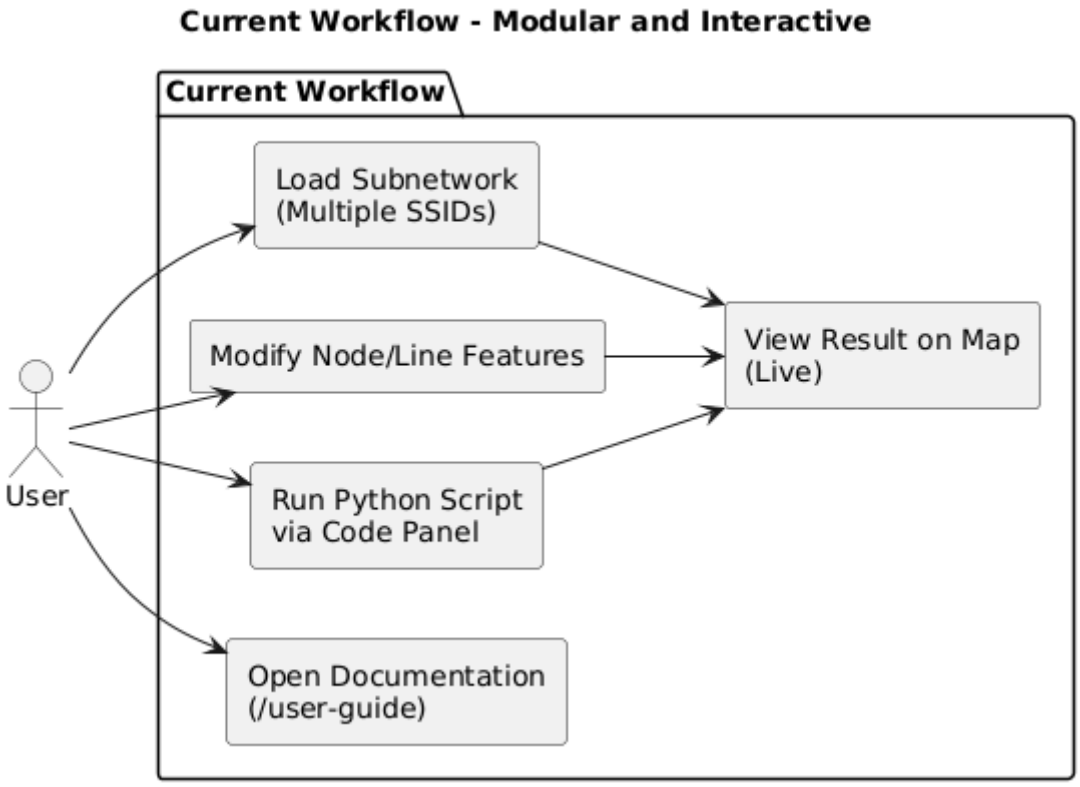

Figure 8 - Current User Flow

6.3.2. PREVIOUS USER FLOW

By contrast, the previous implementation had no user interface or modularity. Thus, the user was required to:

- Execute all steps in a single shell script, which runs all Python scripts, including data loading, styling, trace execution, and HTML rendering.
- Refresh the static HTML file generated at the end of execution to view the results.

- Limit your analysis to one network at a time, constraining the comparison and user flows across multiple networks.
- Operate without integrated documentation or examples, depending solely on guidance from expert trace developers.

The previous workflow was functional but rigid and time-consuming, only accessible to users with specialised trace programming skills.

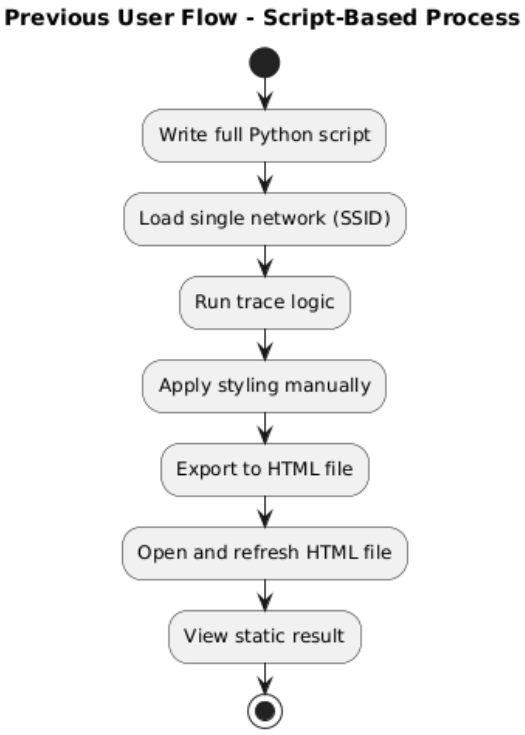

Figure 9 - Previous User flow

6.3.3. COMPARATIVE USER FLOW SUMMARY

Table 6 – User Flow Comparison Between Previous Implementation and the NAVI Trace Toolkit

Feature	Previous Implementation	NAVI Trace Toolkit (Current)
Subnetwork Loading	Manual, one at a time	Interactive, multiple SSIDs
Node/Edge Styling	Script-based and static	Dynamic, UI-driven or scripted
Script Execution	Full script required	Modular, panel-based
Result visualization	Refresh the HTML file	Real-time map rendering
Documentation Access	No formal documentation	Documentation with new functions and integrated with a Markdown HTML viewer.
User Interface	None	Interactive web-based UI

The NAVI Trace Toolkit transforms the user experience from a monolithic style, heavy code into a modular and interactive workflow that facilitates smooth trace analysis.

6.3.4. EXECUTION TIME COMPARISON

Performance experiments were conducted using a corporate Dell laptop with an Intel i7 processor. Each test was repeated 100 times to ensure statistical reliability.

Trace workflows' performance with a single subnetwork:

- Trace Execution: The average time to execute a trace in the NAVI Trace toolkit on a single network is 1.25 seconds, with a standard deviation of ±0.27 seconds. This metric reflects the time to run the trace logic and render the result on the user interface.
- Subnetwork Loading: Loading and rendering of a single subnetwork, i.e. input an SSID in the NAVI Trace Toolkit, takes 3.92 seconds on average with a standard deviation of ±0.55 seconds. This workflow comprises retrieving the subnetwork from the SPEN NAVI server, generating the GeoJSON, and rendering it in the application. This metric compares the application performance with the previous bulk implementation, which followed a similar workflow but lacked a user interface.
- Previous Monolithic Workflow: The previous Python-based workflow took an average of 10.73 seconds to complete, equivalent to the current application's subnetwork loading workflow, with a standard deviation of ±1.28 seconds. This included all operations executed in a single shell script.

Additionally, experiments were conducted with two to nine subnetworks loaded simultaneously to assess network trace scalability performance. The subnetwork() function was selected for the experiment because it traverses the entire subnetwork, exploring all nodes and edges. The results show that the execution time increases exponentially with the number of subnetworks. Despite this performance, the NAVI Trace Toolkit is still more efficient than the previous implementation with one subnetwork, even when handling up to 9 subnetworks. This

improvement in multi-network rendering and loading times is attributed to the modular architecture of this project, as well as the use of efficient algorithms and high-performance modules such as ujson.

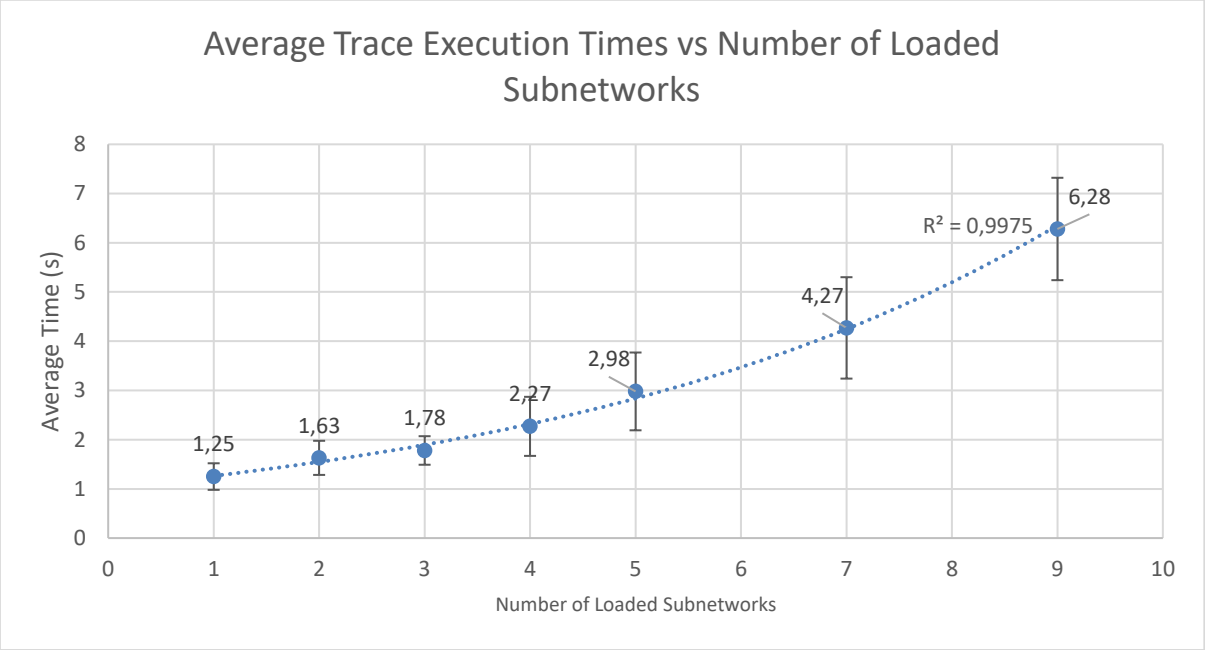

Figure 10 - Average Trace Execution Times vs Number of Loaded Subnetworks

6.3.5. EXECUTION TIME SUMMARY

Table 7 – Execution Time Comparison Between the NAVI Trace Toolkit and Previous Implementation

Scenario	Average Time (s)	Standard Deviation (s)
Current Trace Execution (1 subnetwork)	1.25	0.27
Current Subnetwork Loading	3.92	0.55
Previous Workflow (1 subnetwork)	10.73	1.28
Current Trace execution (9 subnetworks)	6.28	1.04

6.4. USER STUDY RESULTS AND EVALUATION

6.4.1. TASK-BASED PERFORMANCE EVALUATION

A user test was conducted with five SPEN engineers to evaluate the usability and efficiency of the NAVI Trace Toolkit. The engineers were asked to complete the initial setup and five additional tasks in the app. The five tasks were:

- Task 1: Add or Remove Subnetwork SSIDs.
- Task 2: Change All Node and Line Colours.
- Task 3: Add Colour and Size to Dist_Transformer Nodes

- Task 4: Locate the Code Documentation
- Task 5: Run a Trace from an Example in the Code Documentation.

Each participant had 30 minutes to complete the setup and the five tasks. ISO 9241 defines usability of the users based on effectiveness and efficiency metrics [31]. On one hand, effectiveness measures how wholly and accurately the users achieve their goals. This will be calculated as the percentage of completed tasks per user.

$$Effectiveness(\%) = \frac{Number\ of\ completed\ tasks}{Number\ of\ Tasks} 100$$

On the other hand, efficiency measures the time expended in relation to the results. It will be calculated through the overall efficiency metric, which gives the tasks per minute the user can do in our application. The formula is expressed as:

$$Efficiency = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} n_{ij}}{\sum_{j=1}^{R}\sum_{i=1}^{N} t_{ij}}$$

Where:

- R: Number of users.
- N: Number of tasks.
- n_{ij}: result for tasks i by the user j.
    - $= 1$ If the task is completed.
    - $= 0$ If the task is not completed.
- t_{ij}: time spent by the user j to complete i.
    - If the task is not completed, time is measured until the user gives up.

Initial setup took an average of 5.05 minutes with a standard deviation of 0.64 minutes, indicating a consistent and streamlined onboarding process.

The effectiveness of the user test was 100%, meaning all the users could achieve all the tasks within 30 minutes. The effectiveness of the test was calculated to be 1.64 tasks per minute, demonstrating that the users could complete the tasks efficiently and reliably. These results confirm that the NAVI Trace Toolkit offers a high-level, intuitive interface that supports both rapid task execution and successful task completion.

6.4.2. COMPARATIVE SETUP TIME ANALYSIS

In contrast, the previous toolkit implementation required significantly more time to set up:

- 40% of the users reported setup times between 1 and 1.5 hours.
- 40% required between 30 and 60 minutes.
- 20% completed setup between 15 and 30 minutes.

The time reported by developers using the previous toolkit highlights the substantial reduction in time with the updated NAVI Trace Toolkit, from an average of one hour to just over five minutes.

6.4.3. SURVEY-BASED UX EVALUATION

A structured user experience survey was conducted using a 1-5 Likert scale (1 = Strongly Disagree, 5 = Strongly Agree). The survey was designed to cover nine key dimensions of usability and performance. Evaluation sent to the engineers comprised the following nine statements:

1. The toolkit helps me complete trace-related tasks efficiently.

2. The interface is clear and easy to navigate.
3. The visual design helps me understand the trace results and the energy network.
4. The toolkit loads quickly and responds promptly.
5. The interface works well on my screen resolution and setup.
6. It is faster than the previous version.
7. It is more intuitive than the previous version.
8. I prefer the new toolkit because it provides built-in trace functions.
9. Overall, I prefer the new toolkit over the previous one.

The average ratings of each question are plotted in Figure 11.

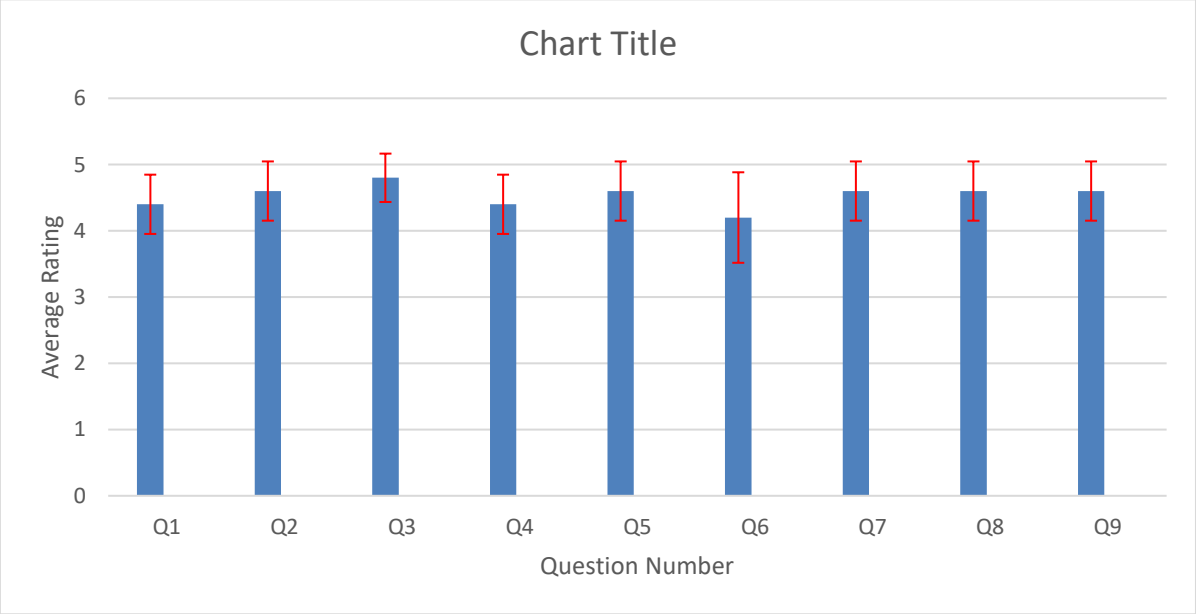

Figure 11 - Average Likert Scores for User Experience Question

All dimensions in the survey were highly rated, averaging over 4. Key findings from the results are:

1. Efficiency: Users reported they could complete all the tasks efficiently.
2. Interface and Design: The interface was reportedly intuitive and easy to navigate. Trace results were displayed efficiently and clearly. Question 3 received the highest rating, emphasising the platform's intuitive interface for helping users understand trace operations.
3. Performance: The toolkit was perceived to respond quickly.
4. Resolution and compatibility: Users found that the application performs well on their screens based on question 5.
5. New Features: Built-in trace functions were positively rated.
6. Overall preference: Most users reported a better experience using the new toolkit version.

6.4.4. OPEN-FEEDBACK AND FEATURE RECOMMENDATIONS

An anonymous open-ended feedback question was included in the user survey to gather suggestions for future improvements. This evaluation offered valuable insights into current user needs and expectations. The most common features that the users want to be included in the NAVI Trace Toolkit can be summarised as follows:

- Onboarding for new trace developers: New users might find it challenging to understand what traces are and what they are used for. Since trace-based workflows represent a novel analytical technique, the users do not understand the benefits and practical applications of using trace analysis in energy networks. An onboarding module or tutorial could improve accessibility for non-trace developers.

- Import/export feature: Users expressed interest in a feature to save and share trace Python scripts efficiently, enhancing collaboration among engineers. This improvement might be added along with code tabs, which enable the user to work with multiple scripts in the same session.
- Export trace data structure: The users would like to export the resulting graph data structure after the trace analysis. For instance, users might want to export subnetworks created during mini-island grid simulations to assess their self-sufficiency. The users suggested a CSV export file, which can be extended to other data file formats.
- Minor feature details: Some users suggested that initial zooming should fit all subnetworks loaded upon launch. Others suggested that the code panel be resizable, not only hideable, to improve the coding experience on small screens or multi-monitor setups.

These suggestions highlight the users' strong interest in implementing collaborative capabilities and lowering the bar for new users. Incorporating these features in future iterations would enhance the user experience and broaden the toolkit's applicability.

6.5. USE CASE

In energy distribution networks, maintaining the integrity and reliability of infrastructure is crucial. As these networks age, the likelihood of faults and safety hazards increases [32]. The components of the distributed energy network deteriorate over time, making proactive identification and intervention essential for preventing failures. Traditionally, energy distribution companies have relied on periodic inspections or reactive maintenance after problems occur; however, these methods can be costly and risky.

To analyse the age of the network using data traces, we introduced the concept of the Critical Installation Path (CIP). This data-driven metric helps identify the most vulnerable path within the network. The CIP is defined as the electrical path composed of the oldest and longest connected lines, calculated by multiplying the age of each line (years since installation) by its length (meters). The CIP weight metric serves as a proxy for infrastructure exposure, capturing both the temporal degradation and the spatial extent of the network. The age of the lines indicates how components degrade and become obsolete over time, while the length of the lines represents their physical footprint and the potential area affected by any failures. Therefore, older and longer lines are more likely to disrupt service continuity.

6.5.1. FIRST APPROACH: DEPTH-FIRST SEARCH (DFS) FOR LONGEST WEIGHTED PATH

In the initial method for computing the CIP, a custom Depth-First Search (DFS) algorithm was implemented to traverse the network and identify the longest simple path with the highest cumulative CIP weight. The DFS algorithm explores all possible simple paths starting from each node in the subnetwork. The largest CIP is tracked through the iteration and updated when a path with a higher CIP is found. After the DFS algorithm is run with all the nodes, the most critical installation path is highlighted using a continuous colour gradient based on age. The code implementation of the Dijkstra-based CIP is provided in Appendix C.

This approach provides a global view of the subnetwork's longest CIP infrastructure chain, which may require attention. However, since DFS explores all possible paths, it is computationally intensive in large or densely connected networks. Additionally, it does not account for energy network logic, which is relevant in real-world energy distribution scenarios.

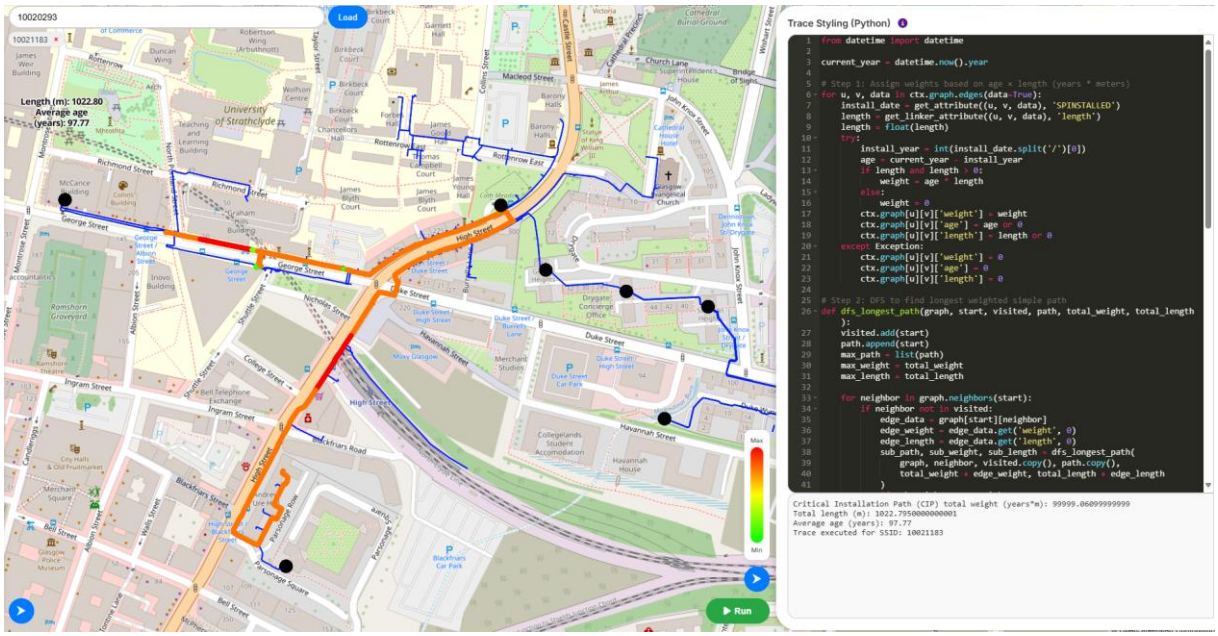

Figure 12 - Critical Installation Path (CIP) Trace with First Approach (DFS), with Transformers Highlighted in Black

Let J be the number of junctions and E the number of edges, the time complexity of this DFS approach is:

- Overhead complexity due to computing the weights of each edge adds a linear time complexity $O\ (E)$.
- Standard DFS has $O\ (J + E)$ time complexity. However, in this approach, the DFS is not used to traverse the graph, but to explore all possible simple paths from every node. A simple path is a path that does not repeat any nodes.
- In an undirected graph with V nodes, the number of simple paths can be up to $O\ (2^J)$.
- Since the algorithm runs a DFS simple path search on every node, the total time complexity becomes: $O\ (J\ 2^J)$.

This approach is not scalable for large and dense networks. It serves as a conceptual baseline but requires a computationally feasible and operationally relevant reformulation.

6.5.2. SECOND APPROACH: DIJKSTRA WITH CUSTOM WEIGHT FUNCTION

Identifying the longest path in a network is a well-known NP-hard problem; that is, the computational effort increases exponentially with the size of the network. In the context of energy distribution, where networks can have thousands of nodes and edges, using an exhaustive search method, such as Depth First Search (DFS), to find the longest weighted path becomes impractical.

To address this, we introduced constraints grounded in energy network logic. During initial testing, we observed that some paths identified as critical (high CIP) did not represent vulnerable customer connections. For instance, if a transformer lies in the middle of a path (Figure 12 - Critical Installation Path (CIP) Trace with First Approach (DFS), with Transformers Highlighted in Black, transformers in black), a fault on one side may not impact customers on the other side due to protective devices that isolate the faulted section. This insight led to a more realistic formulation of the problem: instead of searching globally, we define an energy source point, typically a transformer or network controller, and trace outward using Dijkstra's algorithm.

Although Dijkstra is designed to find shortest paths, not longest ones, it becomes useful when we reinterpret the problem: we are not seeking the longest path overall, but rather the critical path from an energy source to a customer. We first compute with Dijkstra the shortest path for each customer, then look for the longest path among the shortest.

If a customer can be reached via multiple paths—one with high CIP and another with low CIP—the presence of the low CIP prevails over the high CIP because it ensures resilience. If the high

CIP fails, the low CIP remains functional; therefore, the chance of a customer not being served represents the probability of both CIPs failing. However, although combining both CIPs would more accurately represent the probability of a customer not being served, we consider only the low CIP for this analysis. This is because it is less likely to fail and represents the most reliable path available.

Given the previous assumption, the lowest CIP is the most significant for each node, which makes Dijkstra's algorithm appropriate for solving this problem. Consequently, the focus is not on the longest path within the subnetwork, but on identifying the most vulnerable customer—specifically, the one whose shortest path to the controller has the highest CIP.

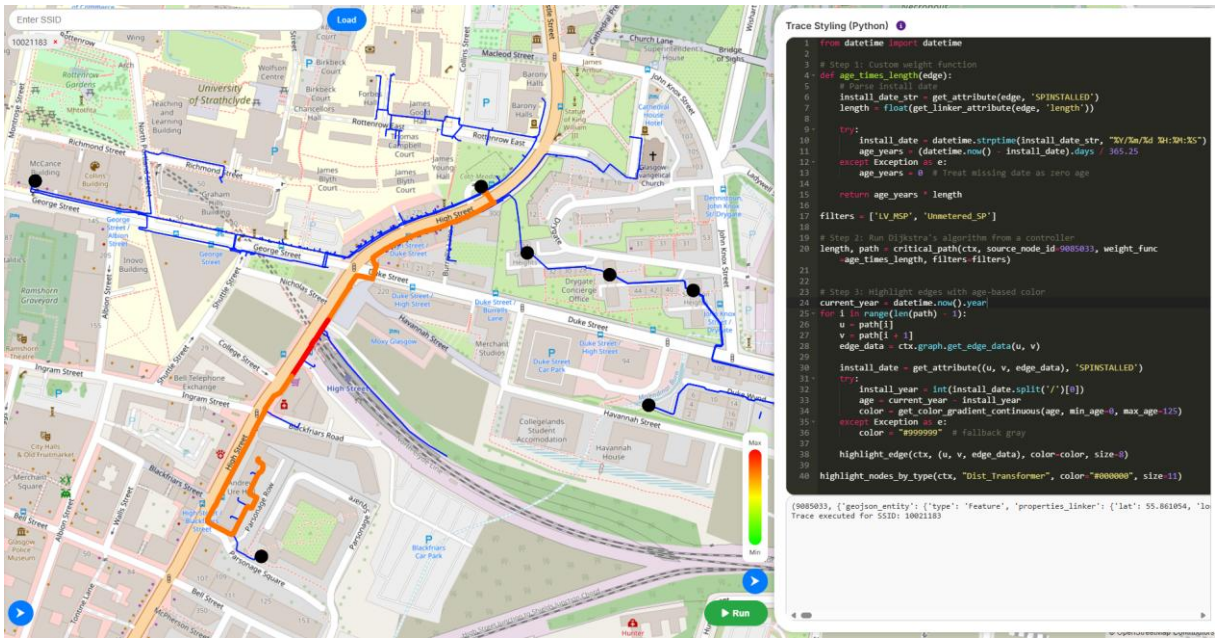

Figure 13 - Critical Installation Path (CIP) Trace Using Second Approach (Dijkstra), with Transformers Highlighted in Black

This approach dramatically reduces computational complexity by avoiding exhaustive path calculation, leveraging Dijkstra's efficiency for shortest path computation, and filtering only

customer nodes to assess risk. It also aligns with energy network principles, where hierarchy, redundancy and fault tolerance are built into the topology. By focusing on reachable customers and their minimum critical shortest path, we balance computational feasibility and operational relevance, making the NAVI Trace Toolkit a practical solution for ageing infrastructure analysis. The code implementation of the Dijkstra-based CIP is provided in Appendix C.

The step-by-step breakdown describing the CIP analysis using the Dijkstra function is:

1. Define the custom weight function, similarly we did with the previous approach, the metric for CIP calculation is the age multiplied by the length.
2. Using the previous custom weight function, we run the critical_path() function with Dijkstra's algorithm from a controller node. This computes the shortest paths to all nodes based on the last weight. Then, it identifies customer nodes by filtering only those representing customers and gets the customer with the maximum CIP value. This return, the customer is most exposed to the ageing infrastructure and its path.
3. Highlight the critical path, with edges coloured using a gradient based on age.

This algorithm avoids NP-hard complexity using efficient shortest-path computation and focusing on energy network logic. The time complexity of this algorithm is:

- Dijkstra's algorithm has a time complexity of $O(E + J \log J)$ when using a Fibonacci heap priority queue [25], where J is the number of junctions and E is the number of edges.
- The custom weight function is computed once per edge, adding a linear overhead of $O(E)$.
- Filtering the customer nodes and finding the maximum CIP among them is $O(V)$ as we need to pass through all the nodes.

- Therefore, the resulting total complexity is: $O(E + J \log J)$. This is polynomial time, making it highly scalable and suitable for large energy networks.

This second approach is more efficient and realistic; it avoids the NP-hard path algorithm by not searching for all possible paths and respects network logic based on hierarchical control.

The pseudocode of the CIP second approach is defined as follows:

Input:

- A network graph with metadata (installation date, length)
- A defined source node (i.e., transformer or controller)
- A set of customer node types (e.g., LV_MSP, Unmetered_SP)

Output:

- The shortest path to the customer with the highest CIP value
- Visual highlights of the path and its components

Table 8 – Dijkstra-Based CIP Trace Pseudocode with Custom Weight Function

```
1  Define CIPWeight(edge):
2      installDate ← get installation date from edge
3      length ← get physical length from edge
4      age ← compute age from installDate using current date
5      return age × length
```

```
6  filters ← CustomersEntityTypes

6  sourceNode ← predefined transformer or controller
7  (Distance, Path) ← critical_path(graph, sourceNode, weightFunction = CIPWeight, filters=filters)

8   for each consecutive edge (u, v) in criticalPath:
9       age ← compute age from edge installation date
10      color ← map age to gradient colour
11      highlightEdge(u, v, color = color, size = medium)

12  return criticalPath
```

6.5.3. USE CASE DISCUSSION

The CIP use case showcases the NAVI Trace Toolkit's ability to go beyond traditional topological tracing and demonstrates a critical_path function application. This use case demonstrates the capabilities of the platform to build tailored traces. This trace developer tool transforms traces from a spatial logic into a multi-dimensional trace tool.

The key capabilities demonstrated in the use case are:

- Attribute-driven edge weighting, allowing flexible schemes that can be extended to other applications. For example, weights could be based on load capacity, fault frequency, maintenance cost based on the ground and material, etc.
- Critical_path() custom algorithm tailored for energy network. This function enables scalability across large networks.

- Visual feedback mechanisms, such as colour gradients based on age, enhance interpretability for operations and maintenance crews. This colour gradient helps to prioritise older lines over newer ones.

7. CONCLUSION AND FUTURE WORK

The NAVI Trace Toolkit marks a transformative step in energy network trace development, evolving from a rigid, script-based workflow into a modular, interactive, and user-friendly platform. By incorporating a user-friendly web interface, real-time Python execution, and integrated tracing functions, the toolkit enables engineers to conduct intricate trace analyses with enhanced efficiency and accessibility.

The toolkit now includes support for multi-network analysis, a previously unavailable feature. Quantitative performance benchmarks indicate significant improvements over the previous implementation, with subnetwork loading times reduced from 10.73 seconds to 3.92 seconds, and trace execution times reduced to 1.25 seconds. User testing confirmed high usability, with a 100% task completion rate and an average efficiency of 1.64 tasks per minute. Survey results further validated the platform's intuitive design, responsiveness, and preference over the legacy system.

The Critical Installation Path (CIP) use case demonstrates the toolkit's real-world analytical capabilities. The initial DFS-based approach, while correct and conceptually valuable, was computationally intensive. The refined Dijkstra-based method offers a scalable, practical solution that aligns with energy network principles. It enables targeted assessments of infrastructure risk based on the age of the lines. This method also includes the custom critical_path function to help developers compute additional critical paths, enhancing replicability and scalability.

Overall, the NAVI Trace Toolkit delivers a robust, scalable, and user-centric solution for energy network analysis, laying the groundwork for future innovation in trace development and infrastructure management.

7.1. FUTURE WORK

Several proposed areas for future development aim to enhance the NAVI Trace Toolkit and expand its applicability. One key direction is the introduction of onboarding modules to assist new users in understanding trace concepts and their practical applications. Since traces are a novel analytical technique, educating engineers about their value and application is essential. Testing the platform with students could also be a valuable initiative, allowing them to learn about trace analysis while providing feedback on usability and learning outcomes.

Additionally, the user feedback and recommendations outlined in section 6.4.4 should be considered for the continued development of the platform. These suggestions, such as adding export options, multi-tab traces, and enhancing collaborative features, reflect user needs and guide future iterations.

Expanding the trace function library is another critical area of focus. By introducing new trace analytical tools, the toolkit can better assist developers and increase its utility across multiple operational scenarios. This includes enhancing the critical path function by incorporating more accurate probabilistic failure models. Instead of selecting the shortest CIP when a node has multiple paths, future work could explore a modified Dijkstra algorithm where the CIP is computed as a weighted combination of all CIPs. This approach may yield a more accurate representation of network reliability, especially under partial failure scenarios.

Integrating with external systems would greatly expand the platform's capabilities. In addition to its connection with NAVI, the platform can interface with SCADA and other energy management systems. The toolkit could adopt data source formats like the Common Information Model (CIM) to facilitate data exchange and interoperability with industry standards.

These future developments will help position the NAVI Trace Toolkit as a comprehensive and scalable energy network analysis solution, supporting operational decision-making and educational outreach.

8. REFERENCES

[1] P. A. Longley, M. F. Goodchild, D. J. Maguire and D. W. Rhind, Geographical Information Systems: Principles, Techniques, Management and Applications, West, Sussex: Wiley Publishing Company, 1999.

[2] A. A. Sallam and O. P. Malik, Electric Distribution Systems, 2 ed., Hoboken, NJ: Wiley-IEEE Press, 2019.

[3] N. Bui, A. P. Castellani, P. Casari and M. Zorzi, "The Internet of Energy: A Web-Enabled Smart Grid System," *IEEE Network,* vol. 26, p. 39–45, July 2012.

[4] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7231/. [Accessed Aug 2025].

[5] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)," 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252. [Accessed Aug 2025].

[6] P. Thubert, C. Bormann, L. Toutain and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header," 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8138. [Accessed Aug 2025].

[7] D. Peintner and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)," 2014. [Online]. Available: https://www.w3.org/TR/exi/. [Accessed Aug 2025].

[8] S. Eren, D. Küçük, C. Ünlüer, M. Demircioğlu, Y. Yanık, Y. Arslan, B. Özsoy, A. H. Güverçinci, İ. Elma, Ö. Tanıdır, Y. C. Ölmez and S. Sönmez, "A ubiquitous Web-based dispatcher information system for effective monitoring and analysis of the electricity transmission grid," *Energy,* vol. 116, p. 1044–1056, October 2016.

[9] A. Kluge and A. Termer, "Human-centered design (HCD) of a fault-finding application for mobile devices and its impact on the reduction of time in fault diagnosis in the manufacturing industry," *Applied Ergonomics,* vol. 65, p. 234–245, Nov 2017.

[10] D.-J. Kang and S. Park, "A Conceptual Approach to Data Visualization for User Interface Design of Smart Grid Operation Tools," *International Journal of Energy, Information and Communications,* vol. 1, p. 64–75, Nov 2010.

[11] S. A. Blanchard, "Giving Data a New Face," *Sigma: Collaborate. Innovate. Deliver,* vol. 11, 2011.

[12] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, Boston, 1994.

[13] B. S. Gardner, "Responsive Web Design: Enriching the User Experience," *Sigma: Collaborate. Innovate. Deliver,* vol. 11, 2011.

[14] Esri, "ArcGIS Geospatial Platform Overview," 2025. [Online]. Available: https://www.esri.com/en-us/arcgis/geospatial-platform/overview. [Accessed Jul 2025].

[15] G. E. Vernova, "Geospatial Network Management (Smallworld GIS)," 2025. [Online]. Available: https://www.gevernova.com/software/products/geospatial-network-management-smallworld-gis. [Accessed Jul 2025].

[16] Esri, "Configure a trace," 2025. [Online]. Available: https://pro.arcgis.com/en/pro-app/3.3/help/data/utility-network/configure-a-trace.htm. [Accessed Jul 2025].

[17] B. Kuridža, "Potentials and Limitations of Web GIS in the Utility Industry," MSc in Geoinformatics, Aalborg University, Copenhagen, 2019.

[18] "MapLibre GL JS Documentation," 2025. [Online]. Available: https://maplibre.org/maplibre-gl-js/docs/. [Accessed Aug 2025].

[19] E. D. Fournier, F. Federico, R. Cudd and S. Pincetl, "Building an interactive web mapping tool to support distributed energy resource planning using public participation GIS," *Applied Geography,* vol. 152, p. 102877, Jan 2023.

[20] "GeoJSON Specification," 2025. [Online]. Available: https://geojson.org/. [Accessed Aug 2025].

[21] D. Oliver and E. Hoel, "Exploring the ArcGIS Utility Network Trace Framework," 2023. [Online]. Available: https://www.esri.com/arcgis-blog/products/utility-network/data-management/exploring-the-arcgis-utility-network-trace-framework. [Accessed Jul 2025].

[22] D. Oliver and E. G. Hoel, "A Trace Framework for Analyzing Utility Networks: A Summary of Results (Industrial Paper)," Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 249-258., 2018.

[23] A. Hagberg, D. Schult and P. Swart, "NetworkX," 2025. [Online]. Available: https://networkx.org/. [Accessed Jul 2025].

[24] T. Peixoto, "Graph-tool," 2025. [Online]. Available: https://graph-tool.skewed.de/. [Accessed Jul 2025].

[25] M. L. Fredman and R. E. Tarjan, "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms," in *25th Annual Symposium on Foundations of Computer Science*, Singer Island, 1984.

[26] Figma, "Figma," 2025. [Online]. Available: https://www.figma.com/. [Accessed Aug 2025].

[27] Python-Markdown, "Python-Markdown — Python-Markdown 3.8.2 documentation," 2025. [Online]. Available: https://python-markdown.github.io/. [Accessed Aug 2025].

[28] K. Reitz, "requests · PyPI," 2025. [Online]. Available: https://pypi.org/project/requests/. [Accessed Aug 2025].

[29] UltraJSON, "Ultra fast JSON decoder and encoder written in C with Python bindings," 2025. [Online]. Available: https://github.com/ultrajson/ultrajson. [Accessed Aug 2025].

[30] Pallets, "Welcome to Flask — Flask Documentation (3.1.x)," 2025. [Online]. Available: https://flask.palletsprojects.com/en/stable/. [Accessed Aug 2025].

[31] International Organization for Standardization, "ISO 9241-11:2018," 2018. [Online]. Available: https://www.iso.org/standard/63500.html. [Accessed Aug 2025].

[32] L. K. Mortensen, H. R. Shaker and C. T. Veje, "Relative fault vulnerability prediction for energy distribution networks," *Applied Energy,* vol. 322, p. 119449, Jun 2022.

9. APPENDIX A: ALIGNMENT WITH SUSTAINABLE DEVELOPMENT GOALS (SDGS)

This project aligns with several United Nations Sustainable Development Goals (SDGs), focusing on grid digitalisation, infrastructure, and decarbonisation. The NAVI Trace Toolkit contributes to the energy sector's digital transformation, supporting smarter, more resilient, and more inclusive energy systems.

SDG 7 – Affordable and Clean Energy

By improving trace analysis tools, this project enhances the optimisation of electrical distribution networks, leading to more efficient energy delivery and better integration of Distributed Energy Resources (DERs). Trace analysis is critical for enabling DER deployment, decarbonising the grid, and transitioning to low-carbon energy systems.

SDG 9 – Industry, Innovation, and Infrastructure

The project enhances infrastructure management by modernising outdated workflows through web-based technologies. It facilitates the digitalisation of the energy sector, contributing to developing a smarter grid and a more resilient and adaptable infrastructure.

SDG 11 – Sustainable Cities and Communities

Reliable and efficient power distribution is essential for sustainable urban development. The NAVI Trace Toolkit facilitates accurate and accessible analysis of distribution networks,

making it a valuable resource for supporting smart grids and meeting the demands of evolving urban environments.

SDG 13 – Climate Action

The platform promotes the integration of low-emission technologies into the grid, indirectly leading to the grid's decarbonization. Conducting proper trace analysis offers more profound insights into network behaviour, helping to identify imbalances between distributed energy resource (DER) generation and grid capacity. This information contributes to more informed planning and operation of sustainable energy systems.

10. APPENDIX B: DFS-BASED CIP ANALYSIS CODE

```
from datetime import datetime

current_year = datetime.now().year

# Step 1: Assign weights based on age × length (years * meters)
for u, v, data in ctx.graph.edges(data=True):
    install_date = get_attribute((u, v, data), 'SPINSTALLED')
    length = get_linker_attribute((u, v, data), 'length')
    length = float(length)
    try:
        install_year = int(install_date.split('/')[0])
        age = current_year - install_year
        if length and length > 0:
            weight = age * length
        else:
            weight = 0
        ctx.graph[u][v]['weight'] = weight
        ctx.graph[u][v]['age'] = age or 0
        ctx.graph[u][v]['length'] = length or 0
    except Exception:
        ctx.graph[u][v]['weight'] = 0
        ctx.graph[u][v]['age'] = 0
        ctx.graph[u][v]['length'] = 0

# Step 2: DFS to find longest weighted simple path
def dfs_longest_path(graph, start, visited, path, total_weight,
total_length):
    visited.add(start)
    path.append(start)
    max_path = list(path)
    max_weight = total_weight
```

```
    max_length = total_length

    for neighbor in graph.neighbors(start):
        if neighbor not in visited:
            edge_data = graph[start][neighbor]
            edge_weight = edge_data.get('weight', 0)
            edge_length = edge_data.get('length', 0)
            sub_path, sub_weight, sub_length = dfs_longest_path(
                graph, neighbor, visited.copy(), path.copy(),
                total_weight + edge_weight, total_length + edge_length
            )
            if sub_weight > max_weight:
                max_path = sub_path
                max_weight = sub_weight
                max_length = sub_length

    return max_path, max_weight, max_length

# Step 3: Try all nodes as starting points
longest_path = []
max_total_weight = 0
max_total_length = 0

for node in ctx.graph.nodes():
    path, weight, length = dfs_longest_path(ctx.graph, node, set(), [],
0, 0)
    if weight > max_total_weight:
        longest_path = path
        max_total_weight = weight
        max_total_length = length

# Step 4: Highlight the critical path
for i in range(len(longest_path) - 1):
    u, v = longest_path[i], longest_path[i + 1]
    age = ctx.graph[u][v].get('age', 0)
```

```
    color = get_color_gradient_continuous(age, min_age=0, max_age=125)  # Adjust max as needed
    highlight_edge(ctx, (u, v, ctx.graph[u][v]), color=color, size=8)


# Step 5: Compute average age
average_age = max_total_weight / max_total_length if max_total_length > 0 else 0
print(f"Critical Installation Path (CIP) total weight (years*m): {max_total_weight}")
print(f"Total length (m): {max_total_length}")
print(f"Average age (years): {average_age:.2f}")
```

11. APPENDIX C: DIJKSTRA-BASED APPROACH CIP

ANALYSIS CODE

```
from datetime import datetime

# Step 1: Custom weight function
def age_times_length(edge):
    # Parse install date
    install_date_str = get_attribute(edge, 'SPINSTALLED')
    length = float(get_linker_attribute(edge, 'length'))

    try:
        install_date = datetime.strptime(install_date_str,
"%Y/%m/%d %H:%M:%S")
        age_years = (datetime.now() - install_date).days / 365.25
    except Exception as e:
        age_years = 0  # Treat missing date as zero age

    return age_years * length

filters = ['LV_MSP', 'Unmetered_SP']

# Step 2: Run Dijkstra's algorithm from a controller
length, path = critical_path(ctx, source_node_id=9085033,
weight_func=age_times_length, filters=filters)



# Step 3: Highlight edges with age-based color
current_year = datetime.now().year
for i in range(len(path) - 1):
    u = path[i]
```

```
    v = path[i + 1]
    edge_data = ctx.graph.get_edge_data(u, v)

    install_date = get_attribute((u, v, edge_data), 'SPINSTALLED')
    try:
        install_year = int(install_date.split('/')[0])
        age = current_year - install_year
        color = get_color_gradient_continuous(age, min_age=0,
max_age=125)
    except Exception as e:
        color = "#999999"  # fallback gray

    highlight_edge(ctx, (u, v, edge_data), color=color, size=8)

highlight_nodes_by_type(ctx, "Dist_Transformer", color="#000000",
size=11)
```