School of Information Technology and
Engineering at the ADA University

School of Engineering and Applied Science
at the George Washington University

URBAN EXPLORATION OF IÇƏRIŞƏHƏR: DATA INTEGRATION AND PATHFINDING SYSTEM.

A Thesis
Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By
Abbas Aliyev
April, 2023

THESIS ACCEPTANCE

This Thesis by: Abbas Aliyev

Entitled: *Urban Exploration of Içərişəhər: Data Integration and Pathfinding System*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

Dr. Stephen Kaisler	
(Adviser)	(Date)
Dr. Abzatdin Adamov	06.05.2025
(Program Director)	(Date)
Dr. Abzatdin Adamov	06.05.2025
(Dean)	(Date)

ABSTRACT

Navigating through historical urban districts as İçərişəhər or the Old City of Baku ushers in new challenges never met in the typical navigation systems. The usual tools are inclined upon path analysis on the basis that the path be least in distance from one spot to another, leaving aside the myriad amounts of cultural references, scenic beacons, and the specific needs of exploring heritage places on behalf of visitors. This thesis went on to fill that gap by proposing data backend support, data processing, and deployment of multi-criteria urban exploration navigation system for İçərişəhər.

The aim was to build a system able to generate distinct paths such as the standard shortest path by distance, "cultural" path for optimizing historical and cultural context, and "geocaching" path for including points of interest from this geocaching game. A comprehensive approach was used to source data from OpenStreetMap (for the road network) and Google Places API for Points of Interest-POIs with respect to a clearly defined area of study. A robust data processing pipeline was designed with Python's geospatial libraries and was used to clean network geometry, calculate segment lengths accurately, find the intersections in question efficiently through R-tree spatial indexing, and generate network graphs accurately. Eventually, the gemini-2.5-pro Large Language Model was used for uniquely filtering over 11,000 POIs to 5,689 really relevant POIs according to context and exploration objectives. Now the "interestingness" of different road segments was reckoned with a new "relevance" metric that came into play after refining of filters and network-based smoothening.

A mix of PostgreSQL/PostGIS and Neo4j hybrid database models was setup with the former used for the exhaustive storage of spatial objects and attributes and the latter for network topology modeling and prompt pathfinding using its Graph Data Science (GDS) library. Proceeding from GDS, the A* search algorithm was employed to do the shortest path search and find a valid geographical path using segment length and relevance score, respectively. The geocaching path is constructed by combining two shortest paths directed through the closest to original shortest path geocache. A FastAPI backend controls all database layer interactions and expose the routing logic, while Docker along with Docker Compose supported containerization of the system, hence achieving reproducibility and ease of continuous deployment.

Experimental outcomes show the generation of various and more than feasible paths for each of the models, which is a clear indication of the success of the relevance factor and hybrid architecture. Testing for observation proved it had reasonably fast response times with an average of 200ms for the API which demonstrates adequate performance. This work has contributed to a carefully crafted data processing workflow using an unexpected LLM POI filter application, a custom metric for relevance, and a functional plug-and-play containerized backend system that certainly paves the way for enhanced exploration and navigation tools for cultural heritage in mixed and rich historical urban environments.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Explanation
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BFS	Breadth-First Search
CORS	Cross-Origin Resource Sharing
CRS	Coordinate Reference System
CSV	Comma-Separated Values
DFS	Depth-First Search
ETL	Extract, Transform, Load
GDS	Graph Data Science (Neo4j Library)
GIS	Geographic Information System
GPS	Global Positioning System
JSON	JavaScript Object Notation
LLM	Large Language Model
MBR	Minimum Bounding Rectangle
NoSQL	Not Only SQL
OGC	Open Geospatial Consortium
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OSM	OpenStreetMap
PBF	Protocolbuffer Binary Format
POI	Point of Interest
PSO	Particle Swarm Optimization
RDBMS	Relational Database Management System
RQ	Research Question
SQL	Structured Query Language
SRID	Spatial Reference Identifier
U.E.	Urban Exploration
UI	User Interface
VLSI	Very Large Scale Integration
WGS	World Geodetic System
WKT	Well-Known Text

1 INTRODUCTION

1.1 Problem Statement

It appears that navigating through the ancient site of the Old City of Baku, İçərişəhər is a task that cannot readily be resolved with conventional point-to-point navigation systems. Most of the traditional navigation tools fail to truly address certain tailored queries that are interest-based and heritage-focused. They are not designed to accentuate the cultural and historical values of the very special sites. This is the gap that could likely be filled by navigation-based technology. Current solutions of the digital world are such that they mostly lack an in-depth view regarding landmark contributions in shaping the contextual fabric within İçərişəhər. The data on which these systems feed have multiple inaccuracies and are widely distributed between various data infrastructures. As a follow-up to this one, the strategies towards spatial data integration can fall along the lines of quality data and attribution resolution. Not addressing this complexity may greatly exacerbate navigation within historical urban districts.

Urban exploration is an active quest to potentially unveil the sub-cartographic layers and lodes of culture of a city. This kind of experience can introduce the visitor to the unassuming but critical part of every city's history and culture that go the unnoticed or overlooked on other prescribed tourist circuits. By following strict tourism, a visitor goes to sites that are frequently taken in a not very sincere manner as far as appreciating their physical and historical presence are concerned; in contrast, an ethnographic approach fringes into 'what is it to appreciate something for the first time, what are the impacts of rawness, how do two travel mates jointly encounter the lesser discussed. This, in turn, tries to encourage the user to create some most meaningful interaction with the place. The aim is for users to begin imprinting their souls with subtle details, which, concerning architectural history and multi-layered human presence, relinquish the well-catalogued attributes of the site's most significant yet underestimated heritage.

Moreover, conventional pathfinding algorithms are generally ill-equipped to handle probing of the multiple criteria fundamental in enhancing an enriching experience of urban exploration in case of heritage sites. They usually prioritize some factors like minimizing the distance and computational efficiency, often neglecting the more vital scenic or historic value. This demarcates an area where the compassion of the network searchers experienced with generalized network algorithms may be adorned to take on the irresistible anatomies necessary for observation.

Most of the current digital mapping crisscrosses the handful of landmarks constituting the medieval layout of İçərişəhər. There are plenty of narrow streets, alleyways yet unmarked, and landmarks steeped in cultural significance. The lack of detail may bear more harm by way of confusion than good toward providing guidance. Crossroads in the area often also require memorizing major landmarks or particular buildings. These examples designate an advanced study area of data collection and validation that goes beyond the navigation of well-established thought processes.

1.2 Research Questions

This research addresses the identified challenges through several key research questions. Satisfactorily answering these questions will enable the application of geospatial data integration capabilities to heritage contexts, facilitate the development of advanced navigation algorithms sensitive to cultural criteria, and support the creation of methodologies for evaluating urban exploration routes based on culturally meaningful tours.

1.2.1 RQ1

How can the A* search algorithm be effectively extended to incorporate qualitative criteria, such as historical importance or scenic value represented by a derived relevance metric, for navigation within İçərişəhər?

This involves devising proper cost functions that adequately balance distance against cultural relevance, preparing and properly integrating the necessary data into the graph model, and running the

performance of the adapted A* search algorithm within a complicated urban network, thus producing a basis transformed for any number of other heuristic search methods.

1.2.2 RQ2

What approaches achieve the most effective form of data integration for creating a integrated and accurate view of İçərişəhər's urban terrain out of various geospatial data sets fit for a purpose in navigation?

This question explores the challenges of data integration, aiming to establish consistent data validation mechanisms, including conflict resolution and quality control, to underpin the navigation system.

1.2.3 RQ3

How can the effectiveness of urban exploration routes be evaluated through both computational metrics and user experience measures to ensure meaningful cultural engagement in historical districts?

This research question is looking into what can potentially include various layout options and examination of results using predefined metrics between pure benefits of algorithmic efficacy and reflections towards the art of cultural discovery and usability.

1.3 Significance of the study

A major gap currently addressed in this research is the ever-growing chasm between technical approaches to urban navigation and practical tools for cultural heritage exploration. Specialized tools that will help a user explore historical districts like İçərişəhər extend their significance from pure technical innovation straight to the compound of cultural heritage preservation, tourism development, and urban computing methodologies.

On the side of technical advancement, this work calls upon the necessity for data integration in comprehensive urban representations where data can be hard to come by, exists in a fragmented fashion or introduces some other form of sparsity. The work makes an evolutionary stride in pathfinding by introducing techniques whereby cultural and historic factors can be mixed in with network search; all these imply that there is a possibility of the applicability of this framework in any similar historical district.

In relation to cultural heritage preservation, this research is also seeking to provide digital tools where visitors can be engaged with; these are the very tools through which visitors can plan their own personalized routes with attention to cultural preferences. In this way, exploration is encouraged in light of elements beyond the physical distance; therefore, learning and appreciating the rich heritage of İçərişəhər's old district, with the potential of increased satisfaction and retention by visitors.

The tourism industry benefits from the improvement of navigation tools that transform basic pathfinding to meaningful cultural experiences. With personalized routes that are tailored to individual preferences, ranging from the quest for historical landmarks, to architecture of note or a view of scenic beauty, then the system supports savvy tourism development, balancing interests articulated by different types of visitors alongside those keen on preserving their heritage.

The methodology for evaluating route quality is also developed from this research in terms of both technical and user experience, being the next important step to being relevant in the greater domain of user-centered design in urban computing applications. These evaluation frameworks will act as a guiding star for the next stage of development in cultural heritage technology, ensuring that technical innovations relate to better and greater user understanding of the space.

1.4 Scope and Limitations

The study will target the development of an urban exploration back-end system which could be scalable for the historical district of Baku, İçərişəhər. That is, the study will take up these burningly important issues:

1. **Geographic Focus**: The system will only focus on İçərişəhər and its neighboring areas with a clear relationship to a recognized area of contemporary history and culture.
2. **Data Integration**: Various spatial data are aggregated, integrated, and validated, primarily from OpenStreetMap (OSM) for the road network and the Google Maps Places API for points of interest (POIs). Priority is given to road networks and POIs relevant to urban exploration.
3. **Algorithm Development**: This involves processing the OSM road network data, including splitting road segments at intersections identified efficiently using spatial indexing techniques (leveraging R-trees), and constructing a graph representation suitable for pathfinding. Here, the main focus is on the implementation and validation of the A* search algorithm, adaptable to different goals: the shortest path (minimizes distance) and cultural exploration (maximizes on a self-defined relevance metric conditioned to the nearness of POIs).
4. **System Implementation**: A FastAPI-backed back-end framework is under development, utilizing a hybrid database approach: Neo4j for graph operations and PostgreSQL with PostGIS for spatial data management. The object scope gets constrained to the comprehensive implementation of core back-end-centric facilities, involving content-level shaping, processing, algorithmic ruing, and Application Programming Interfaces (APIs) useful at the user front, and that development of User Interface (UI) is specifically not being considered under this research.
5. **Validation Methodology**: Computational performance and the quality of generated routes will be evaluated through field testing.

The following limitations could influence research methodology and outcomes:

1. **Data Accuracy and Completeness**: External data sources possess varying levels of quality and completeness, potentially impacting system accuracy. This research tried its best to handle this limitation by integrating data from various sources and applying select validation procedures whenever possible. At this point, further authentication from professional levels, based on fieldwork or professionally made map databases, should be given weight. This might include discrepancies between OSM-supplied lengths and actual geometrical lengths calculated from boundary clipping alongside the presence of complex or disjoint geometries like MultiLineStrings demanding sifting during data processing.
2. **Resource Constraints**: External APIs have limitations in terms of service; therefore, in both cache-based implementations and regularly updated databases, the provision of database access and data update becoming critical factors in system development. Specifically, the use of the Google Places API is subject to free tier limitations on the number of requests. While the implemented hexagon grid strategy optimizes coverage within these limits for the study area, scaling the system to larger areas or requiring more frequent updates might incur significant costs or necessitate alternative data sourcing strategies.
3. **Noncommercial Context**: As this is a research project, the use of data in the system is confined within the rules regarding data use; however, some functionalities may be restricted as compared to a commercial setting.
4. **Computational Efficiency**: Performing complex, multi-objective route optimization incurs computational costs, particularly at runtime. This necessitates balancing route optimality with calculation time, influencing the design and selection of optimization algorithms and strategies for route evaluation.
5. **Temporal Limitations**: The project captures a snapshot of İçərişəhər's urban landscape, but cannot automatically adapt to changes in the physical environment without manual updates.

1.5 Motivation

My motivation derives from the sheer thrill of sheer technicality experienced by my passion for graph algorithms and data engineering and the heartwarming sense of boundless cultural appreciation. As a backend developer, I am lured by the intricate issues of urban navigation among the immense historical environment. The İçərişəhər's combination of a medieval urban fabric with rich cultural heritage finds

a propitious ground for the manifestation of sorties into innovative approaches to pathfinding and data integration.

The possibilities of confronting my academic interests and post-rational learnings with the real-world processes of data from several worlds and various advanced algorithms are what make attractive my orientation towards this project. For me, this is the very moment where theory meets practice vis-à-vis a problem of consequence, accretive to whether-or-not the solution benefits people from every shade of place and culture. The resistance is then not just positive computation, but cultural considerations in techno-cultural harmony, where we illuminate the ambiance with some enlightenment to the question about the significance of what we are still finding.

These aesthetic considerations of engineering-biased creativity are motivated by the proudest parts of my heart, having worked amidst the rich history and culture of İçərişəhər. Filling higher-ups of curiosity for comprehensive ways of exploration surpassing pedestrian navigation means including the psychological narratives of historical significance and the dreamscape, an enlightening state of incidental interpretation completely changes visitor experience and steadily increases a sense of heritage appreciation for Baku.

This thesis represents what I love to do: the quest for data discovery, aspiration for associated movement of graph algorithms, and the creation of a software system that solves some meaningful issues. This project is more of interdisciplinarity-heuristics how I can contribute to the advance of technology and keeping watch over the preservation and appreciation of historical urban environments.

2 LITERATURE REVIEW

2.1 Introduction

Urban exploration (U.E.), frequently defined as the study of the usually unseen, out-of-bounds aspects of towns and industrial facilities, is becoming more and more widespread, enthralling enthusiasts and scholars [Robinson, 2015]. It includes a wide variety of activities, from exploring small to large abandoned buildings [Foad et al., 2021] to hidden but functioning areas of urban life [Ai and Kachitvichyanukul, 2009]. Historically, it has somehow had ties with counter-cultures and subcultural values, yet it holds a different lens to viewing a city's history, culture, and societal dynamics [Kreller and Ludwig, 2021; Garrett, 2012]. A more important potential application for it shall be as an instrument for detaching the past from an understanding of rediscovered histories and cultures. [Robinson, 2015].

However, traversing such environments is complicated with urban geography in the historical context. They have street configurations and high densities of varied points of interest (POIs) along with entitlements and restrictions imposed to meet efficient movement with historical or cultural immersion [T.J. Ai, 2009]. Conventional navigation methods based on Dijkstra's algorithm [Dijkstra, 1959] or the A* algorithm [Hart et al., 1968] are essentially pathfinding devices linking two points with the shortest or fastest route. In most, if not all, cases, that would be enough, but these algorithms tend not to be adequate for a navigation system designed for urban exploration, whose bouquet is much richer since factors other than the distance or time come into play: historical importance of landmarks along the route, aesthetic appeal of routes, avoidance from very densely crowning areas, etc. [Cook, 2018].

Current navigation systems usually depend on a small source of information such as data for the road networks and are not designed to easily integrate different geographic data such as historical maps, points of interest databases, and user-generated content [Kreller and Ludwig, 2021]. Thus, the breadth and depth of guidance provision for urban explorers on how to traverse places that are culturally rich and historically significant remain highly limited.

However, the developing scenario where large, well-harnessed datasets become openly accessible brings researchers within reach of their promises in graph database technologies [Vukmirović et al., 2019]. Neo4j is one of the graph database types that is able to efficiently represent complex relationships between entities and queries these relationships, thus it is able to create very promising models of the complex networks of urban spaces and the diverse POIs within them [Hunger, 2017]. The integration

of graph databases with relational databases would bring a far more holistic representation of urban environments, combining the advantages brought from both structured and unstructured data management [Thakare and Vyawahare, 2018].

This thesis seeks the means to bring together traditional pathfinding algorithms and what one might characterize as their unique requirements for city exploration in historical districts. Instead, our novel approach will take advantage of the integration of different sources of geospatial data, including historical maps and POI databases, as well as user-generated content into a single data model that will then be used to create and evaluate advanced pathfinding algorithms to support multiple criteria such as historical importance, aesthetic appeal, and user preferences. Through developing a more well-rounded representation of urban environments through the combination of the strengths of graph-based representation, multi-criteria optimization, and data integration, our research will provide more comprehensive, adaptable, and user-centric navigation and exploration solutions in urban settings with rich historical and cultural heritage. The following literature review will provide a summary of the current state of the art, and will provide a foundation of building blocks for the research to be undertaken.

2.2 Urban Exploration: Concepts and Challenges

Urban exploration (U.E.) goes beyond simply visiting abandoned or hidden sights: it speaks to multiple forms of engagement with the urban environment-adventure, historical inquiry, aesthetic appreciation, and social critique [Robinson, 2015]. Even as it is closely associated with counter-cultural or transgressive movements, U.E. can also be seen as acts of "internal tourism" [Ai, T. J., et al., 2008] or "off-limits tourism" [Brown, 2001] whereby people visit those daringly inaccessible and under-discussed parts of the built environment. Paiva and Manaugh (2008) refer to these areas as "Temporary, Obsolete, Abandoned and Derelict Spaces" (T.O.A.D.S), which differs considerably from the curated, systematic, and often sanitized sensations offered by mainstream tourism.

Motivations behind U.E. are so many: from the simple desire for adventure and thrill from hidden places [Foad et al., 2021] to deeper engagement into the history and culture which abandoned buildings tend to reference [Garrett, 2012]. In some cases, nostalgia might motivate exploration: a desire to expand contact to a reality that is usually covered or erased by urban development. [Robinson, 2015]. This means that others consider exploring the city as social commentary since it interrogates property ownership, access to urban spaces, and the dominant narratives of urban history [Kreller and Ludwig, 2021].

However, urban exploration in historical districts presents a unique set of challenges. These areas often feature:

- **Intricate Road Networks**: Historical districts are usually endowed with complicated and thus local road networks; by this is meant that many streets are too narrow, winding alleys and many dead-end passages fill such areas. Because of that, even navigation with the map and GPS proves laborious [Cook, 2018].
- **Heavy Density of Points of Interest (POI)**: Most of the historical districts have high density of POIs such as historical buildings, monuments, landmarks, or cultural sites. Such dense POIs can overwhelm traditional navigation systems that mainly depend on shortest-path algorithms and produce suboptimal routes that miss major sites or disregard the historical context of the area [T.J. Ai et al., 2008].
- **Seeking Efficiency with Cultural Relevance**: Urban explorers in historical districts want to optimize the efficiency of the routes (such as minimizing the time spent traveling or the distance traveled) with their attempt to garner some of that cultural and historical richness from the area while doing so. This often requires a more nuanced approach to pathfinding that involves factors other than distance or time [Robinson, 2015].
- **Safety and Accessibility**: Most places where urban explorers like to go are in dilapidated states and thus pose safety threats to anyone who enters them [Robinson, 2015].

Existing studies on urban exploration systems have rather happened to touch on some aspects of the activity such as:

- **Visualization and Documentation**: This is where a great number of U.E. websites and online communities have been established with the purpose of documenting and redistributing their experience through photographs, videos, and written accounts [Garrett, 2014]. These prove to be better sources of insight into motivations, practices, and aesthetics of urban explorers.
- **Geographical Analysis**: GIS and spatial analysis methods have been applied to mapping and analyzing the spatial distribution of U.E. sites to reveal patterns and correlate urban development with decay and social factors [Foad et al., 2021].
- **Social and Cultural Implications**: Studies have analyzed the social and cultural meanings of U.E., investigating its place in contesting dominant narratives of urban space, community building among explorers, and awareness raising about neglected or forgotten places [Kreller and Ludwig, 2021].

However, there remains a gap for more comprehensive systems that can help under the distinct challenges posed by urban exploration in historical districts. Such systems should:

- **Integrate diverse geospatial data sources**: Combine historical maps, POI databases, user-generated content, and potentially even real-time data from sensors, to provide a resourceful and dynamic representation of the urban environment.
- **Offer multi-criteria pathfinding algorithms**: Moving beyond shortest-path algorithms to include factors like historical-value, aesthetic appeal, safety, and end user-type preferences.
- **Provide intuitive user interfaces**: User interfaces that allow the user to easily explore, plan, and share their own urban experience.
- **Address ethical considerations**: Respect property rights, ensure user safety, and propagate responsible practices of exploration.

By overcoming these challenges, future urban exploration systems will continue to bring empowerment to individuals by engaging them with the urban environment in freshly meaningful manners; which will then lead to an enriched understanding and appreciation for the histories, cultures, and hidden spaces of our cities. These issues will be explored in more detail in the next sections, combining the contributions that graph databases, relational databases, and advanced pathfinding algorithms can make toward that end.

2.3 Geospatial Data Integration

No urban exploration system can function in a historically rich and complex environment without the ability to integrate the many available geographical data sources. A comprehensive urban spatial model must combine information from multiple sources, each with its own strengths, limitations, and sometimes, biases inbuilt. In this unit, we are designing to underline data gathering of importance against a backdrop of data sources commonly employed by urban exploration systems, situating the algorithms to tie hodgepodge data sources up in a more homogeneous landscape.

2.3.1 The Importance of Data Integration

Urban environments are composed of a complex interrelation of physical, social, cultural, and historical factors. No one dataset can cover the entire richness and multi-dimensionality of these environments.

Basic navigation systems rely on road networks, which offer a simplified representation of the urban space without accounting for the task that scalability can do in urban exploration:

- **Historical Landmarks and Buildings**: Databases of historical sites, monuments, and buildings, often with detailed descriptions, photographs, and historical accounts.
- **Points of Interest (POIs)**: Databases of POIs, including museums, galleries, parks, ruins, abandoned structures, and other sites of cultural or historical significance.

- **User-Generated Content**: Online platforms, forums, and social media where urban explorers share their experiences, discoveries, photographs, and stories.
- **Land Use and Zoning Data**: Information on land use patterns, zoning regulations, and property ownership, which can help identify potential exploration sites and assess their accessibility.
- **Architectural Data**: Detailed information of buildings, such as blueprints, can provide a more detailed view of the building. [Foad et al., 2021]
- **Environmental Data**: Information on factors such as noise levels, air quality, and light pollution, which can influence the exploration experience.
- **Real-Time Data**: Dynamic data sources, such as traffic conditions, weather forecasts, and crowd density, which can impact route planning and safety.

Integrating these diverse data sources provides a more holistic and in-depth understanding of the urban setting, thus paving the way for developing systems according to the specific needs and interests of urban explorers.

2.3.2 Common Data Sources

Various data sources are normally used within urban exploration systems and each carries its own advantages and disadvantages:

- **OpenStreetMap (OSM)**: A collaborative open-source project that provides a freely available map of the world which is created and maintained by a community of mappers [Cook, 2018]. OSM data includes road networks, building footprints, points of interest, land use, and other features. It is easy to recognize its international coverage, high resolution in urban areas, and easy access. However, OSM data could be incomplete or inaccurate in some specific areas, with its quality affected by the data entries of the volunteer mappers. [Ai and Kachitvichyanukul, 2009]
- **Google Maps Places API**: This is a paid service providing access to an extensive set of POIs from businesses, landmarks, and public spaces. Google Maps data appears in a way that is otherwise up-to-date and correct, yet there are restrictions and licensing for using this data.
- **Historical Maps**: Historic maps that have been digitized, usually made available from libraries, archives or museums, present a great opportunity for students or researchers studying the evolution of their cities. These maps locate buildings that have been demolished in the metropolitan area and lines of streets which seem to have been forgotten, which must have been part of the city during their days. This could be a challenge to geo-reference properly since the level of detail of such maps may vary greatly.
- **User-Generated Content**: Online platforms, forums, and social media segments may act as a virtual directory where urban explorers can find and develop descriptions of many spots for urban exploration, complete with pictures, video clips, and stories. User-generated content is a major source of information that cannot be ignored in any way. However, its reliability and quality will clearly vary from one source to the other.
- **Sensor Data**: Real-time geospatial data from sensors, traffic cameras included, can greatly augment the information required for transportation and urban studies. This rich, dynamic information will help shape our understanding of the complexities of the urban world as its variables change in real time.

2.3.3 Data Integration Methods and Technologies

Challenges of integrating such heterogeneous data sources involve:

- **Data Compatibility**: Different data sources may use different formats, coordinate systems, and data models, making it difficult to combine them directly.
- **Data Quality**: Data from different sources may vary in terms of accuracy, completeness, and consistency.

- **Data Semantics**: The meaning and interpretation of data may vary across different sources, requiring careful consideration of data semantics and context.
- **Ethical Considerations**: The use of user-generated content and real-time data raises ethical concerns about privacy, data security, and responsible exploration practices.

Several methods and technologies avoid these challenges and foster data integration:

- **Data Transformation and Standardization**: Data from different sources can be transformed into a common format and coordinate system, using techniques such as data conversion, georeferencing, and data cleaning.
- **Data Fusion and Aggregation**: Data from different sources can be fused or aggregated to create a more complete and comprehensive representation of the urban environment. This can involve techniques such as spatial joins, attribute matching, and data conflation.
- **Ontology-Based Integration**: Ontologies can be used to define the meaning and relationships between different data elements, facilitating semantic interoperability and data integration [Kreller and Ludwig, 2021].
- **Graph Databases**: Some graph databases, such as Neo4j, offer a flexible way to model and query complex relationships between entities, making them well-suited for integrating diverse data sources [Thakare and Vyawahare, 2018; Hunger, 2017].
- **Application Programming Interfaces (APIs)**: APIs allow for the leveraging of external data sources, such as Google Maps, to provide additional information. [T.J. Ai et al., 2008].

Selection of appropriate data sources and methods, and integration of these through sustainable methods and technology, will endow us with a comprehensive and dynamic representation of urban environments that should cater towards built and growing wide spectrum of urban exploration applications. The subsequent sections are going to explore how the same data sources and methods can work to advance in path-searching algorithms crucially for the specific needs of urban explorers in historical districts.

2.3.4 Spatial Indexing for Efficient Processing

The integration and processing of big geospatial datasets like road networks and point of interests commonly call for high-speed spatial queries which can require a great deal of computational power. When it comes to finding intersections between thousands of road segments, the process gets drastically slow if every pair of objects has to be compared directly (brute-force). Standard one-dimensional database indexing structures like B-trees are optimized for ordered, single-value lookups and are thus inappropriate for multi-dimensional spatial sets, in which a query often relates to proximity or overlap in at least two dimensions [Gaede & Günther, 1998].

An interesting solution to this challenge here has been the development of specialized spatial index structures. The most influential and well-accepted of these is the R-tree which was proposed by Guttman (1984). The R-tree is a kind of a balanced tree structure that in actuality operates similarly to the B-tree but its domain of interest is multi-dimensional spatial data. In general, it groups nearby objects together and keeps their MBRs in non-leaf nodes. Leaf nodes contain pointers to actual data objects and their MBRs. Non-leaf nodes contain entries consisting of MBRs that enclose all the MBRs of their child nodes.

This hierarchical structure based on spatial containment allows search algorithms to efficiently prune the search space. While executing a spatial query (e.g., finding objects that intersect a certain area), the algorithm would enter the tree starting at the root. If a node's MBR does not intersect the search area, the entire subtree beneath that node can be virginal and all its contents can be ignored [Guttman, 1984]. This finite state reduction in the number of objects checked considerably fastens the speed of spatial operations involving window queries, nearest neighbor searches, and, most importantly of all for network processing operations, spatial joins and overlays.

R-trees were responsible for a paradigm shift in geospatial software development. Libraries such as GeoPandas typically make use of R-tree implementation (often bare through the 'rtree' library) to

achieve performance boost in spatial operations. Under the current study, the use of geopandas.overlay(), with an underlying R-tree spatial index, allowed for the efficient detection of intersections between road segments, replacing a computationally infeasible brute-force approach and making the detailed graph building process manageable.

2.4 Pathfinding Algorithms and Optimization

A fully integrated and comprehensive representation of the urban environment will necessitate pathfinding algorithms able to efficiently find suitable routes out of so many options. Those paths should be able to accommodate specific interests and needs of the urban explorer at any given time. The current section explains the traditional approach in pathfinding algorithms, developing aspects of multi-objective optimization, and exploring how optimization strategies can be implemented for solving the associated intricacies of urban environments.

2.4.1 Traditional Pathfinding Algorithms

There are several classic algorithms to establish paths in the graph - each with its pros and cons:

- **Dijkstra's Algorithm**: Dijkstra is the classic algorithm used to find the shortest path in the graph between two nodes with non-negative edge weights [Dijkstra, 1959]. It proceeds by traversing the graph step-by-step, marking the nodes that have already been visited and maintaining the other ones in a priority queue based on the calculated distance from the start-up node. The role of ensuring the shortest route is guaranteed by the Dijkstra algorithm. But as the size of the graph increases, with large numbers of nodes and connections involved, it will really degrade in performance.
- **A* Search Algorithm**: This is a graph-based extension of Dijkstra's algorithm that uses a heuristic function to approximate the remaining distance to the goal node [Hart et al., 1968]. This heuristic redirects the search process, attempting to manage nodes that are more likely to be among those on an optimal path. This algorithm significantly outperforms Dijkstra's algorithm, especially when the perfect heuristic is available, but its success relies also on the correctness and admissibility of the heuristic function.
- **Breadth-First Search (BFS) and Depth-First Search (DFS)**: These are basic graph-traversal algorithms helpful for purposefully finding some paths. BFS looks for the shortest path through a non-weighted graph by layer-wise traversing. DFS travels through the graph by reaching for the deepest possible level of the local branch before it fully reverses. Even with an evident advantage of ease of implementation, BFS or DFS may prove ineffective if a graph is large or weighted, because these algorithms may diversify over too many unnecessary nodes. [Cook, 2018]

These conventional algorithms are configured to find the single-criterion "shortest path" across a network based on this criterion (e.g., distance or time). Ready for further exploration in urban situations that call for added criteria, usually historical significance, aesthetic appeal, safety, and others, these algorithms have to be redesigned or adapted to meet the expectations of multi-criteria optimization.

2.4.2 Multi-Criteria Pathfinding and Optimization

Multi-criteria pathfinding implies the search for paths that optimize more than one, probably conflicting, objectives. Many approaches have been proposed to meet this challenge:

- **Weighted Sum Approach**: A simple approach is to sum up multiple objectives into one objective function that takes weighted linear combinations of the three arbitrary criteria [T.J. Ai et al., 2009]. In essence, this allows initially single-object optimization procedure that uses traditional algorithms, for example, Dijkstra's or A*, to search for optimal paths. However, determining reasonable weights can foretell no simple task, and the solution path might not be Pareto-optimal. That is, other paths of equal value of at least one additional criterion and will do no worse on any other could be better.
- **Pareto-Based Approaches**: To choose the optimal paths, one each of which represents different trade-offs between the objectives, the Pareto-based techniques aim to derive sets of Pareto-

optimal paths [Kreller and Ludwig, 2021]. Thus methods such as multi-objective A* (MOA*) and Pareto Archived Evolution Strategy (PAES) have been used to find Pareto-optimal solutions. Established Algorithms can have the task of finding the whole front of Pareto front; it might easily be computationally expensive based on the size of the graph.

- **Constraint-Based Approaches**: Driven by the concept of preference, the constraint-based approaches define constraints on some criteria while optimizing others. For instance, a user might prefer geological constraints, whereas avoiding congestion between History Museum and Disneyland would become the constraint.
- **Interactive Approaches**: In interactive approaches, the user would have the ability to dictate both the criteria and their respective importance. The system can, in turn, give the user a set of optimal paths, from which he or she can be more precise in specifying their requirements. [Cook, 2018]

2.4.3 Optimization Strategies for Complex Urban Environments

There are several concerns when seeking paths in urban environments for pathfinding algorithms:

- **Large-Scale Networks**: Urban networks can possibly be large, with thousands or millions of nodes and edges, making finding optimal paths rather intensive.
- **Dynamic Conditions**: There are dynamic changes in traffic conditions, roads blocked for construction, to name a few, thereby needing real-time adaptation of the pathfinding algorithm.
- **Data Imperfections**: Different data sources such as sources like OpenStreetMap can also have some inaccuracies or inconsistencies. As a result, the efficiency of the pathfinding system is greatly affected.
- **Complex Constraints**: Complex constraints such as avoiding some areas, visiting POIs of specific sorts, and respecting the user's preferences can be seen at any time given the complexity of the urban environment.

Many strategies can be employed to tackle these issues:

- **Hierarchical Approaches**: Dividing the road network into some more levels can create a low and high level, e.g. below major roads and up for local streets, subdivides the graph and reduces the search space to improve performance [Foad et al., 2021].
- **Preprocessing Techniques**: Pre-compute certain information that could be reached using features, distances, reachability between nodes, etc., in order to optimize pathfinding queries.
- **Heuristics and Approximations**: The usage of heuristic functions and approximation algorithms can reduce computational load but can also forego the attainment of optimality.
- **Machine Learning**: The formation of machine learning techniques such as reinforcement learning and deep learning would train previous data sets optimizing pathfinding algorithms practices under concrete assumptions while increasing efficacy and robustness as they adjust subsequently to dynamic conditions [Kreller and Ludwig, 2021].
- **Parallel and Distributed Computing**: This will aid to be able to fast-track route-finding calculations with large networks, following road involvements discussed over the other paragraphs.

Through the combination of optimization strategies, critically efficient and optimum pathfinding algorithms for the most difficult scenario of urban realms will be realized. The following sections will address how these techniques of optimization are embedded into the proposed graph-based approach for urban discovery.

2.5 Graph-Based Urban Representation

The inadequacies of simplified urban representations oftentimes utilized by conventional navigation systems point to the need for a more nimble and all-inclusive treatment of how to represent space. Graph databases, meaningfully, have the necessary apparatus by which to represent complicated urban relationships. At the minimum, the concept currently works to discuss the workings of a graph database

and its benefits to an urban environment, as well as investigating any and all important research on applying graph databases to modeling and analyzing urban structures.

2.5.1 Introduction to Graph Databases

A subset of the larger NoSQL database family, a graph database uses graph structures to represent and store the data associated with its subject. In contrast to relational systems, whose focus has traditionally lain squarely on creating well-known relationships, tables connecting with one another, graph databases deal with the relationships of data points along the graph edge instead. Now the fundamental building blocks in a graph database are found to be as follows:

- **Nodes**: Generally, represent entities or objects-these could be people, places, buildings, or events.
- **Edges (Relationships)**: Construct links between nodes, i.e., represent connectivity or relationships between nodes. Examples of relationships that the edges could depict are such as "located in," "visited," "knows," and "part of".
- **Properties**: Attributes associated with nodes and edges. These are not about the entities and relationships and provide additional information.

This is one of the most critical aspects of a graph database, as it makes them very naturally and intuitively model complex relationships and will work well in an urban setting, where things are heavily interlinked and relationships are quite varied and dynamic over time.

2.5.2 Advantages of Graph Databases for Urban Representation

Graph databases are far superior compared to the traditional relational databases in representation and analysis of urban areas for many reasons:

- **Flexibility**: Graph databases can seamlessly merge any data source-may it be of structured data (e.g., road networks, building footprints), or semi-structured data (e.g., POI databases, user-generated content), and unstructured data (e.g., text descriptions, images).
- **Scalability**: Graph databases should be able to model millions or even billions of nodes and edges, storing and representing them for managing complicated urban landscapes.
- **Performance**: A graph database manages all relationships optimally and therefore can traverse and fetch data quite fast-most of this is beneficial when it is necessary for path finding or for an application where traversing relationships would be imperative.
- **Expressiveness**: Graph databases can model very complex new relationships or patterns that are impossible to model within a relational database context, thus allowing a more complete interpretation of an urban environment.
- **Data Integration**: Graph databases enable the integration of data from various sources to give the complete picture of an urban environment, supporting urban exploration where a user must often rely on many unconnected resources for information when planning trips or locating points of interest.
- **Schema Evolution**: Unlike a relational database, the schema of the graph database is governed by no rules of existence, thus granting a natural evolution and adaptation as adventures draw the conversation into the long run. [T.J. Ai et al. 2008]

2.5.3 Existing Research on Graph-Based Urban Modeling

Many studies have attempted to leverage graph databases in urban modeling and analysis, illustrating the potential for numerous applications:

- **Urban Planning and Transportation**: The earlier use was described in which a graph database was applied for modeling road networks, public transportation systems, and pedestrian walkways to map efficient routing, traffic analysis, and transportation planning [Ai and Kachitvichyanukul, 2009].

- **Social Network Analysis**: The tools for social interaction, community structure studies, and information diffusion could have elicited through the combination of these latest tools with other graph models within an urban setting. [Foad et al., 2021]
- **POI Recommendation**: Now users can use graph databases to recommend different points of POI depending on their preferences-for example, language, country, and city [Robinson, 2015].
- **Urban Morphology Analysis**: Graph databases were proposed as an operative set for complementary analysis, such as structure and form analysis of urban design, and could detect patterns and relationships between buildings, streets, and other urban elements [Thakare and Vyawahare, 2018].
- **Historical GIS**: Graph databases can be used to integrate historical maps and other data sources, providing a dynamic and evolving representation of urban environments over time [Kreller and Ludwig, 2021].

These earnest studies show the flexibility exhibited by graph database research in urban modeling, transcending complex relationships, fine integration, and reasonably wide adoption among diverse applications.

2.5.4 Combining Graph and Relational Databases

While the relationally oriented model provides a predominantly enforced set-based methodology, to arrange and manipulate data with the very structure upon which the data evolved, when it comes to dealing with a relationship, graphs hold the upper hand. Therefore, it is pertinent to combine the strengths of both databases to devise a more competent and comprehensive schema in simulation analysis for every urban setting.

A few approaches have been put forward that promise to mold graph databases into the relational realm:

- **Polyglot Persistence**: Using multiple database systems, each optimized for a specific type of data or query, and combining the results as needed [Thakare and Vyawahare, 2018].
- **Hybrid Databases**: Developing database systems that support both graph and relational data models, allowing users to query and analyze data using either approach.
- **Data Integration Frameworks**: Creating frameworks that facilitate the integration of data from different sources, including graph and relational databases, into a unified representation.

By bringing together the power of graph and relational databases, there is a better platform for urban modeling and solutions. This hybrid flexibility and scalability that could be brought into play in a wider array of applications in support of urban users will make it a full-service platform.

2.6 Relational Databases and Urban Application

Graph databases provide many advantages in terms of the modeling and comparison of relations. An urban data management cornerstone, however, remains relational databases. Their structural nature, maturity, ease in manipulation, and acceptability in urban applications are very useful and could not be obviated. This section discusses the role relational databases play in urban life, endeavors into the strengths and potential limitations of these databases, and instance scenarios toward enabling their interaction with graph-based databases.

2.6.1 The Role of Relational Databases in Urban Settings

Depending upon structured data nature, relational databases have been in use for managing urban data in applications such as:

- **Cadastral Data**: Storing land parcels' information with respect to their property ownership, boundary demarcation, and land-use stipulations.
- **Building Information**: Data about buildings consists of attributes like height, number of floors, completed construction materials, occupancy, and use.

- **Infrastructure Data**: Comprising data representing objects like urban roads, utility networks, and public transportation networks.
- **Demographic and Socioeconomic Data**: Holding population, housing, income, employment, and related socioeconomic data for statistical use.
- **Administrative Data**: Managing data of urban governance, personal licensing, city-licenses, and important service requests.

For managing this kind of data, relational databases are very efficient. They include features like data consistency, integrity, and security, have robust transactional management tools, and efficiently make and process queries. Due to the superior organization of these databases, a long-term effort, creates queries fast and powerful-laden with data suited for a variety of management and operational processes in the field of urban development.

2.6.2 Strengths and Limitations of Relational Databases

The strengths of relational databases in urban applications include the following:

- **Data Integrity and Consistency**: The integrity constraints in relational databases support consistency of the data, whereas the internal relationships established in the relational model enforce data correctness.
- **Structured Data Management**: Relational databases are excellent for the structured management of data, with predefined schemas and established relationships among tables.
- **Mature Technology**: Relational databases are a mature technology, equipped with standardization and proper tools and user knowledge.
- **Widespread Adoption**: Relational databases are highly used in many industries and various organizations, thus making it easier to integrate them with existing systems and to find skilled professionals.
- **Efficient Querying and Reporting**: Relational databases deliver high-capacity queries, which search and retrieve information upon complex user-specific criteria.

However, relational databases have limitations in certain urban applications:

- **Limited Flexibility**: The schema of a relational database is typically rigid, hence requiring complex changes in data structure or integration of new attributes.
- **Difficulty Handling Complex Relationships**: Relational databases do not cope well with representing and querying complex relationships amongst entities, such as those in social network and transportation infrastructure systems.
- **Performance Issues with JOINs**: The queries requiring searching through multiple tables through SQL Join often take time in computation, especially on large datasets.
- **Limited Support for Unstructured Data**: These systems lack sound features for managing unstructured types of data, such as text descriptions, images, or sensor feeds.

Remediation of the above constraints in working with spatial data is centered on the architecture from PostGIS as a circular extension. PostGIS has transformed the PostgreSQL Object-Relational Database into a propelled spatial database by extending support for several geographic object types, including points, lines, and polygons, and by developing a rich library of functions for spatial analytics and typical spatial queries compliant with the Open Geospatial Consortium (OGC) standards. Some important abilities include distance measurement (`ST_Distance`), spatial relationship tests (`ST_Intersects, ST_Contains, ST_DWithin`), operations on geometry (buffering, clipping), and most notably, support for the location-spatial indexing (for instance, through GiST or SP-GuST) with which to greatly speed up spatially-based queries.

2.6.3 Integrating Relational and Graph Databases

With recognition of the strengths and limitations of relational and graph databases, researchers and practitioners have considered a range of methods to integrate them in a bid to benefit from the advantages of both paradigms:

- **Polyglot Persistence**: As previously stated, this paradigm references the practice where different persistence stores are selected for various kinds of data, and queries reach for them accordingly. For example, one might use a relational database management system for storing structured data and a graph database to handle relationships [Thakare and Vyawahare, 2018].
- **Hybrid Databases**: A few databases offer users satisfying performance, allowing both graph and relational data models for accessing and querying data. This affords more flexibility and is good for performing some types of queries.
- **Data Integration Frameworks**: The frameworks make moving forward to pull data from kinds of sources into a consistent form of data that can be queried together irrespective of their native format. One example of this might be for combining data in relational databases with data in graph databases. The user would be able to run queries on all of these sources using just one query language.
- **Extract, Transform, Load (ETL)**: Extract, Transform, Load (ETL): The data may be sent from one database to another employing ETL tools such that the data is available in the Database for which it is best suited [Kreller and Ludwig, 2021].

While PostGIS excels at storing, managing, and querying spatial geometries and their attributes with data integrity, graph databases like Neo4j offer superior performance and expressiveness for analyzing network connectivity and executing pathfinding algorithms. This project adopts a hybrid database architecture, combining the strengths of both PostGIS and Neo4j. This approach aligns with the concept of Polyglot Persistence, where different data storage technologies are employed within a single system, each chosen for its suitability to handle specific types of data or perform particular tasks effectively.

While PostGIS is very well equipped in terms of how it saves, organized, and queries with spatial geometries and their attributes with data integrity, graph database engines, like Neo4j, are much better with relational data because they give superior performance and expressivity for network connectivity analysis or executing pathfinding algorithms. Considering the strengths of both PostGIS and Neo4j, this project prefers to take up a hybrid database approach. This goes well as per the Polyglot Persistence model, in which different database technologies are used in a single system, chosen based on their suitability for the specific type of data or task.

In this setup, PostGIS provides the main authoritative store for the geometric descriptions and descriptive attributes of urban features (roads, junctions, POIs), further capitalizing upon its robust spatial querying capabilities (e.g., the ability to find the nearest road to a coordinate). Neo4j, because of its topological network that had been derived, holds the descriptions of the relationships between these features—which include adjacencies and connectivity. In this regard, Neo4j leverages its graph traversal optimization to perform efficient pathfinding thanks to algorithms like A*. Such a well-considered division of labor actually helps the system to benefit from the core advantages of both database technologies: PostGIS for handling what and where aspects of spatial features, whereas Neo4j for managing 'how connected' and 'how to traverse' factors featured in the network, the overall aim being to deliver a more performant and functionally rich navigation system than could be afforded by either technology alone.

Integrating both relational and graph databases can provide an enhanced and more efficient platform for facilitating urban modeling and analysis. Through such linkage, the benefits of differing methods might be combined, working on both structured and unstructured data, and now supporting a wider variety of applications.

3 RESEARCH APPROACH OR METHODOLOGY

This chapter includes a detailed representation of the methodology applied to develop the critical functionalities of the İçərişəhər urban exploration navigation system. This thesis primarily consists of the data processing pipeline required for the preparation of geospatial data for navigation, along with the backend implementation for multi-criteria pathfinding as well as the containerization strategy for deployment and scalability of the system. Overall, this sets up a strong underpinning for the backend intended to serve various specialized modes of navigation according to Old Baku's unique historical and cultural context.

3.1 Overview of the Approach

In this research approach, various technologies and methodologies play a role in finding solutions to the challenges described in the introduction. The core of this approach includes:

1. **Data Acquisition and Processing**: Use most of the publicly accessed data sources, particularly base road network OpenStreetMap (OSM) and Google Places API for the Points of Interests (POIs), then develop a complete pipeline to clean, process, and enrich this data for navigational use. This involves defining the area of study, performing geometric complexity treatments, accurate road lengths calculation, efficient search for network intersections, filtering POIs for relevance, and coming up with innovative "relevance" metrics for cultural pathfinding.
2. **Hybrid Database System**: A hybrid database system is based on the best features of different database paradigms. PostgreSQL with the PostGIS extension is used for storing and querying detailed geospatial features (roads, intersections, POIs) and their attributes, taking advantage of its mature spatial functions and data integrity features. Neo4j, a native graph database, is employed to store the abstracted network topology (nodes representing roads/intersections, edges representing adjacency) and to perform efficient graph-based pathfinding operations using its Graph Data Science (GDS) library.
3. **Multi-Criteria Pathfinding**: The different paths can be provided through the basic A* search variations. A route from one point to another based purely on distance is usually called the shortest path. This is the “cultural” one that is using the relevance metric based on the derived POIs, while the last is geocaching and finds a route based on the nearest geocache.
4. **Backend API Development**: Building a backend service in the Python FastAPI framework to expose pathfinding features through an internal API linking the data layer (PostGIS, Neo4j) to any potential frontend application consuming such a service.
5. **Containerization for Deployment**: Containerization using docker of each of the system components (PostGIS database, Neo4j database, FastAPI backend), and managing the end-to-end app in Docker Compose to define and manage the multi-container application, delivering consistency in development and in deployment and scalability when necessary.

This establishes the backend and data components of a 3-layer architecture (Data, Backend, Frontend) in which further presentation of data and services to users is to be built using avigation applications.

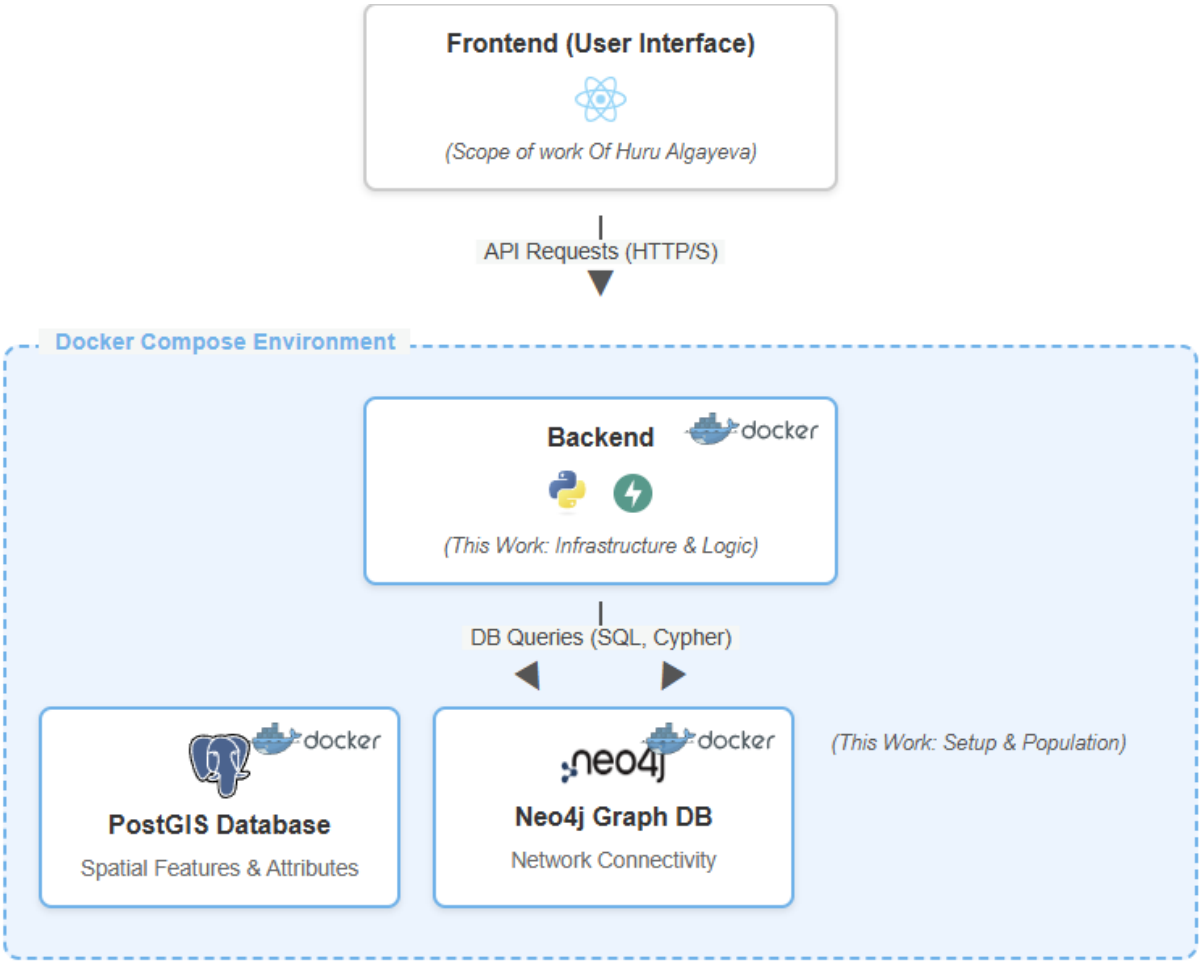

Figure 3.1. High-Level System Architecture Diagram. Diagram shows the system architecture, accentuating the containerization of the components and orchestrated deployment of the backend and data layers.

3.2 Data Acquisition

The foundation of the Navigation System depends on accurate and relevant geospatial data that represents the road network and its points of interest within a study area. This section discusses the definition of a study area, data acquisition processes by the chosen primary sources that have OpenStreetMap (OSM) and the Google Places API.

3.2.1 Study Area Definition

The geographic area this project studies is İçərişəhər (the Old City of Baku) and its surroundings, which should focus on areas important for contemporary history and culture. It was important to set the boundaries since these were critical in ensuring that the most relevant areas fell within the boundary while not processing irrelevant ones.

1. First, a rectangular bounding box was constructed around İçərişəhər arbitrarily. But this turned out to be inappropriate as it included large peripheral areas of little significance and closed off the National Flag Square, one of the major nearby places.
2. A better boundary was created using the `leafmap` Python library in order to interactively draw an initial polygon around İçərişəhər and the adjacently important areas. At first, this polygon was defined in WGS 84 (EPSG:4326) coordinate reference system (CRS).
3. For further refinement, this polygon was projected into the Web Mercator (EPSG:3857) CRS to allow editing directly onto a high-resolution satellite basemap within an individualized

Matplotlib-based polygon editor. With this interactive tool, further adjustments were executed with the tool to the polygon vertices to achieve the accurate delineation of required study areas.

4. After the editing was completed, it saved the refined polygon and re-projected it into the WGS 84 CRS for subsequent data collection steps to access it consistently.

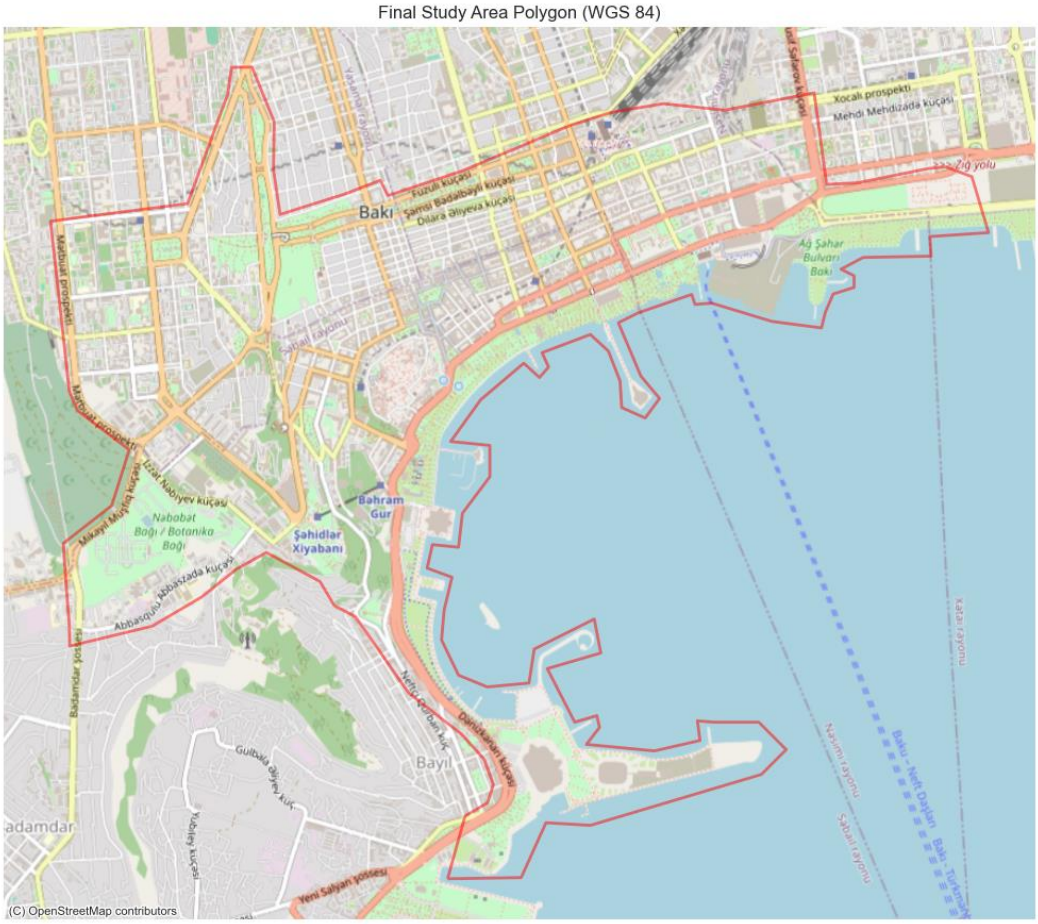

Figure 3.2. Final Study Area Polygon. A map showing red outline of study area polygon, surrounding central parts of the city.

3.2.2 OpenStreetMap (OSM) Data Acquisition

Due to its open access, detailed data, and easy-to-use processing libraries, OSM is an attractive option for determining the road network.

- For network data extraction, the Python library `pyrosm` was utilized. This library efficiently parses OSM data stored in Protocolbuffer Binary Format (.pbf).
- An OSM object was initialized with the path for the .pbf file for Azerbaijan and the bounding box from the refined study area polygon, and the `get_network()` method was called to get the 'walking' network type: it consists of sidewalks, footpaths, and roads that allow for walking, corresponding to urban exploration aims.
- The clipped network data in GeoPandas GeoDataFrame format were then clipped exactly to the refined study area polygon using `geopandas.clip()`.
- In this phase of the project, building footprint data were also obtained from OSM through `pyrosm` for potential use later, but that information was not included in the core network graph.

3.2.3 Google Places API Data Acquisition

Google Places API (there described, with emphasis on the 'Nearby Search' endpoint) was tapped for acquiring a rich data set of Points of Interest (POIs) that are very much needed for the derivation of the

"relevance" metric and to support the geocaching. One of the challenges facing the work was to cover the entire study area while staying within the usage limits set for the free tier for API use (just under 5000 requests per month at the time of undertaking this project).

- A systematic sampling strategy on a hexagonal grid was carefully developed. Hexagons outperform squares and circles to a large extent when it comes to efficiently tiling a plane with shapes closest to the circle (in that they minimize overlap and gaps) with a very regular pattern.
- Under a computed grid covering possible hexagon centers, the study area polygon (in Web Mercator projection) was laid down. The hexagon size (edge length) was tailored to balance the density of coverage versus the total number of API requests. While a smaller hexagon radius increases the chance of not missing POIs within dense areas (since Nearby Search returns maximum 20 results per query), this increases total request count.
- An edge length model of 60 meters appeared to be a good compromise. This is the size through which Nearby Search API calls, centered at individual hexagon centroids, were formed. In total, this setup created 2359 hexagons intersecting with the study area polygon. That meant 2359 API calls and a number far below the limit for free tier, leaving infinite room for reruns or mistakes.
- API calls were made in a batch mode for every centroid of the 2359 hexagons requesting the standard place information fields. After deduplication based on place ID, this yielded a total of 11,109 unique POIs across the defined study area.

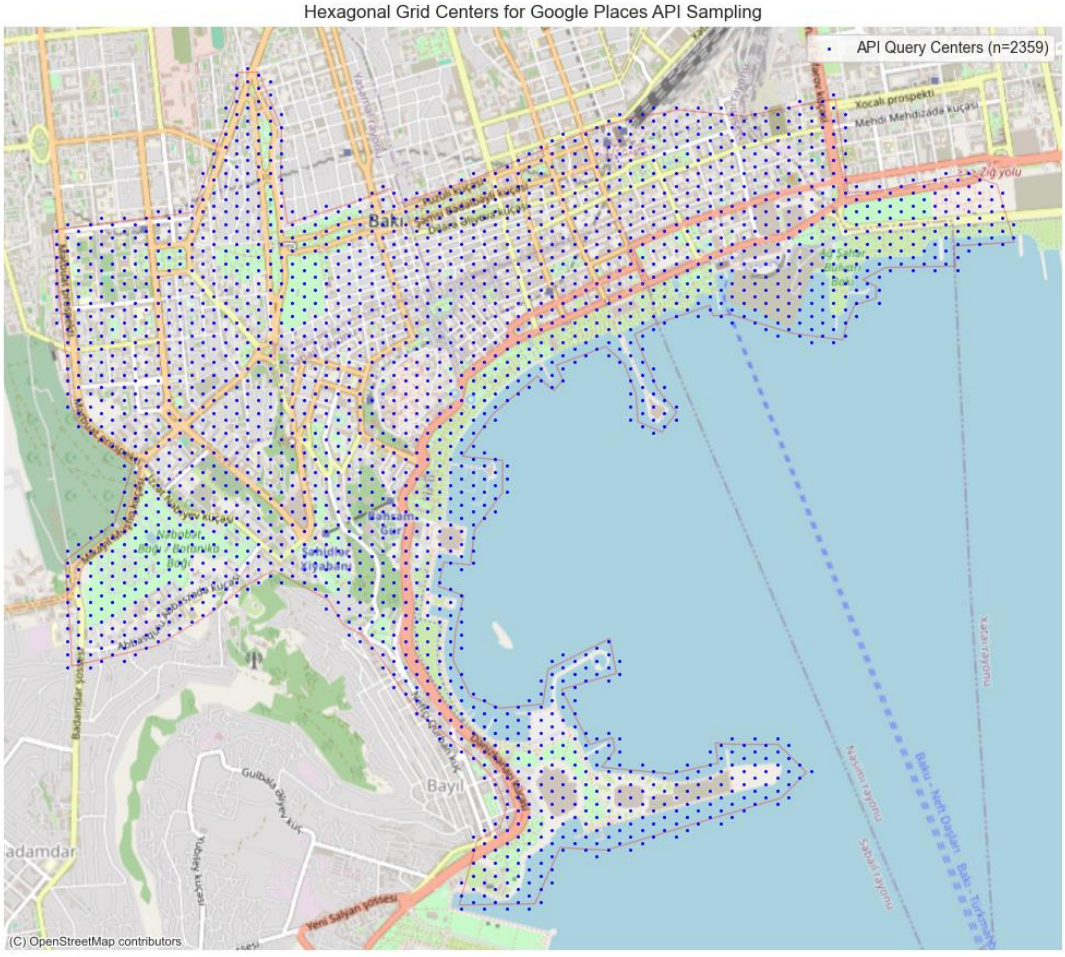

Figure 3.3. Hexagonal Grid for Google Places API Sampling. This diagram shows center of the hexagons in hexagon tiling. These centers are coordinates that are using in Google Places Nearby Search.

3.3 Data Processing Pipeline

Upon finalizing data acquisition for the spelled research area (OSM, Google Places POIs), a robust data processing pipeline was needed to be set up, which included converting it into the proper format suitable for the hybrid database system, and the search algorithms for the possible paths. This pipeline included various components of data cleaning and structuring of the road network. Besides the road network, it also focused on identification of the intersections, filtering and enrichment of POI data, and derivation of 'relevance,' a custom metric.

3.3.1 OSM Road Network Processing

Upon extraction of raw OSM road-network data with `pyrosm` tool, several processing steps were applied to ensure geometric validity, standard attribution, and consistent length measures for graph construction:

1. Initial Filtering: Roads with a distance of zero were removed, as they do not provide valid geometries for routing.
2. Geometry Handling (MultiLineString Processing): OSM data, with more specification after clipping, may sometimes contain MultiLineString geometries which are made up of disjoint segments of interest, thus being roads. Since these were removed during the preliminary stages, OSM roads were retained in this methodology with as much network data as possible
    - An attempt was first made to dissolve all MultiLineStrings into single LineStrings by using `shapely.linemerge()`. This operation would take care of the kind of case where segments ought to be contiguous, and singly stored as distinct parts. (See Figure 4 for the initial geometry distribution).
    - Any geometries still remaining as MultiLineStrings after merging (suggesting that some truly disjoint segments make up one OSM road entry) were exploded using the `geopandas.explode()` function. Exploding unpacks every MultiLineString into multiple rows, each containing a single LineString component; thus, all disjoint segments were treated as separate road segments, with all the proper attributes inherited. This slightly increased the total road segment records from 3437 to 3456 in this dataset (See Figure 4 for data after merge attempt and before explosion)
3. Attribute Consolidation: This involved retention of all essential columns (osm_id, the name, osm_timestamp, length, and geometry); less relevant attributes were consolidated into a single JSON tags field. Redundant tags in OSM were engineered away.
4. Length Recalculation and Analysis: Just perhaps tad more ingeniously true to the old approach, the original length attribute in the OSM was discarded for gross inaccuracy in cases especially allowed by broken geometries often through clipping and processing. Parallel to this, for `geopy.distance.geodesic` upon WGS-84 ellipse, the geometrical lengths of every LineString segment obtained (including products of explode) were recalculated. The new series of comparisons for the original OSM length field against the recalculated geodesic length allowed for second assessment of similarity to the initial analysis (mean error: ~27m; mean percent error: ~22%). Spatial visualizations were prepared to allow a closer look into the spatial distribution of such length calculation errors and single out the faulty segments which were those mostly situated near the study area boundary and because of the complex geometries split. The recalculated geodesic length was used to represent the final length attribute for each segment in the network of roads used for all following processes.

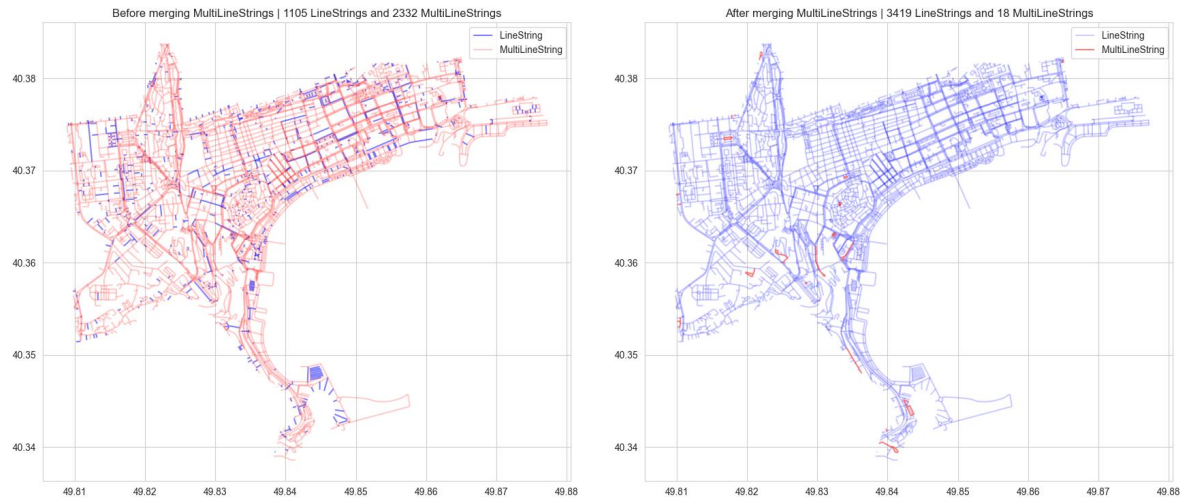

Figure 3.4. Visualization of MultiLineString Processing. LineString geometries in the blue and MultiLineString geometries in the red. First image from the left shows initial distribution of geometries as they came from the OSM. Second image shows geometries after application of shapely.linemerge().

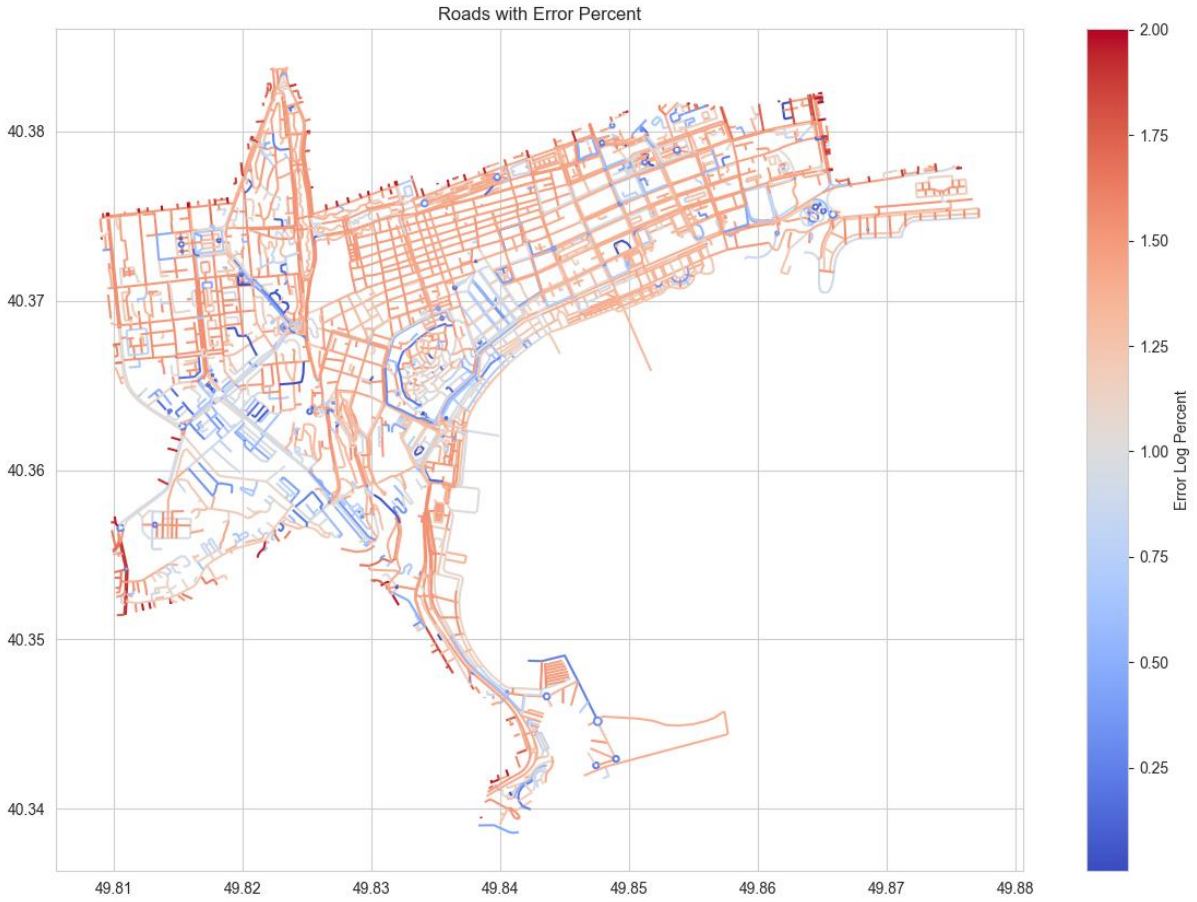

Figure 3.5. Spatial Distribution of Length Calculation Error. Map shows roads colored according to the logarithm of the percent error using visually uniform “coolwarm” colormap. Diagram includes legend color bar.

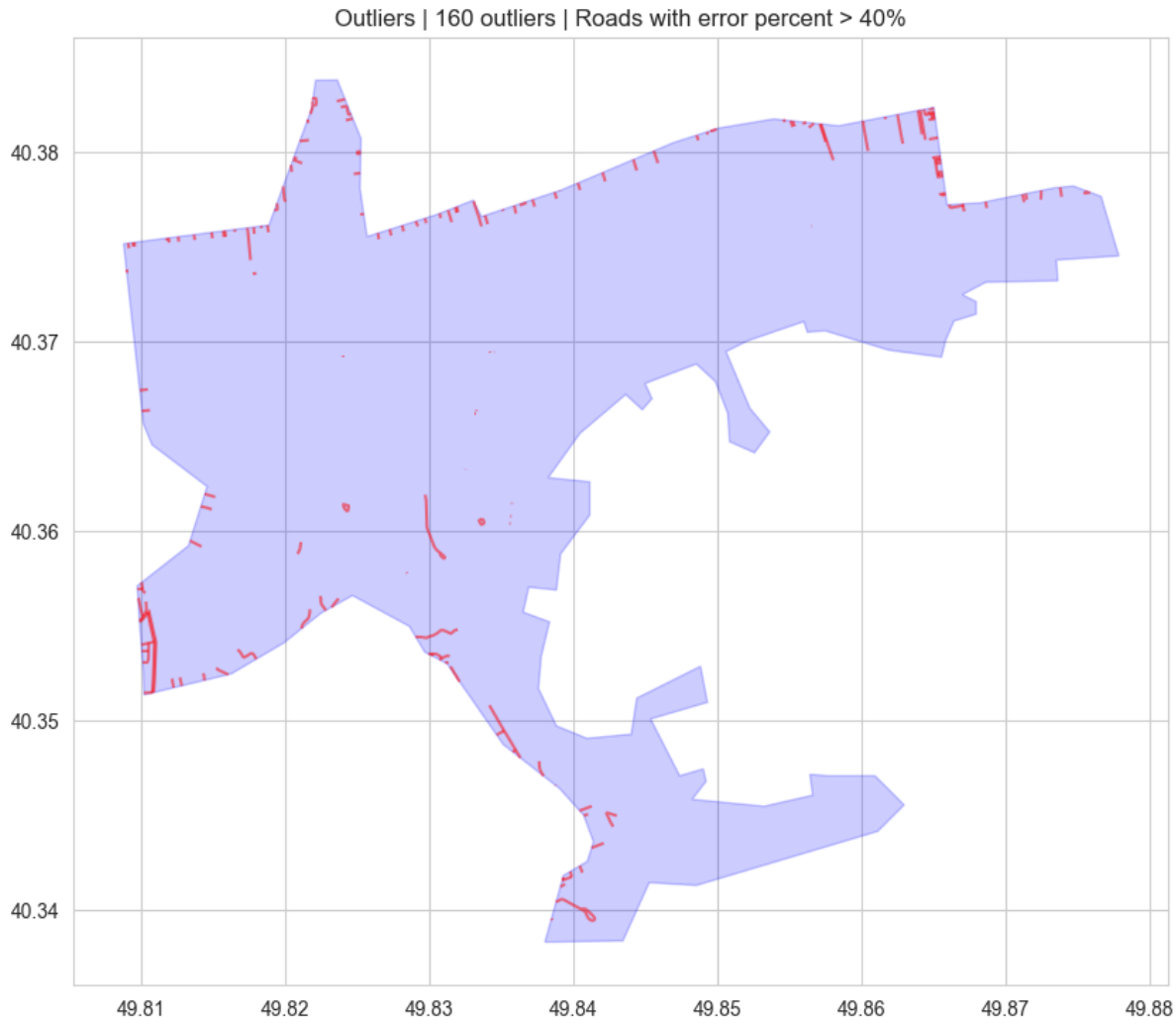

Figure 3.6. Road Segments with High Error (>40% Error). Map shows 160 road segments that have more than 40% difference between provided and calculated lengths. Road segments are displayed on top of the study area polygon.

3.3.2 Intersection Finding and Road Splitting

To have an operational graph where the edges signified connections between distinct road segments at junctions, the continuous LineString geometries were to be split at intersections.

1. **Identification of Intersections**: Initially, an ill-informed brute-force approach comparing each pair of roads for intersections was considered but discarded due to computational unsuitability owing to an $O(n^2)$ complexity for large numbers of line segments. Instead, a highly efficient method that exploited spatial indexing using `geopandas.overlay(..., how='intersection')` was introduced. That function employs an R-tree spatial index to allow rapid identification of road geometries whose bounding boxes overlap them, thus narrowing the number of comparison tests for intersections drastically. An array of intersections was thus returned at locations where road segments crossed or came into contact. Any duplicate redundant points that arose out of different roads intersecting at the exact same site were combined into a single set of actual intersection geometries.
2. **Road Segment Splitting**: For the original road, any and all unique intersections on top of the road were calculated. These were grouped into a shapely.MultiPoint object for each road. Once that was done, the `shapely.split()` function was used to split the LineString geometry of the road associated with the particular MultiPoint object into smaller road segments between intersections or the endpoint and intersection. These road segments inherited all attributes (e.g.,

osm_id, tags, etc.) from the parent road it was derived from, and their lengths were recalculated as detailed earlier through geodesic computation.

3.3.3 Network Graph Construction and Cleaning

The next step after the road splitting method as explained in Section 3.3.2 was to construct the topological network graph representation used by Neo4j for pathfinding. This consisted of formalizing the concepts of nodes and edges based on processed road segments and their intersections, henceforth followed by a cleaning of the graph to ensure connectivity.

1. **Identifying Connectivity**: The previous step was concerned with unique intersection points; however, to build the graph properly, knowledge of precisely which road segments meet at each point was to be determined. `geopandas.overlay(..., how='intersection')` was used again on the newly created splitting road segments (new_roads GeoDataFrame) to give that information reliably. Such spatial join would identify road segment pairs being intersected. These results were filtered (to self-intersections and duplicate pairs where road_id_1 and road_id_2 were swapped), and then exploded to give a definitive list where each row represents a unique intersection point geometry linking two specific road segment IDs (e.g., unique_road_id_1 and unique_road_id_2). Such knowledge allowed linking of road segments through intersection nodes in the graph.
2. **Graph Building**: The networkx Python library is used for constructing the undirected graph. The process loops through the intersection geometries worked out above:
   - For the geometry of each unique intersection point, a corresponding **Intersection Node** is added to the graph ((e.g., named: i_0, i_1, ...), saving the point geometry as a graph attribute.
   - For each pair of road segments (r1_id, r2_id) determined to meet at that intersection point, **Road Nodes** will be added to the graph (e.g. named: r_0, r_1, ...). Each of these nodes will carry all attributes from the new_roads GeoDataFrame row for that segment (this will include: specific geometry of the segment, calculated length, tags, osm_id, etc.). Adding the same road node multiple times is idempotent in networkx.
   - The edges were only drawn between each intersection node and the corresponding road segment nodes meeting at that point. Edges (i_k, r_x) and (i_k, r_y) are created for an intersection i_k connecting roads r_x and r_y.
   - **Edge Weights**: Each edge between an intersection node and a road node acquires a weight attribute equal to half the length of the connected road segment (`road['length'] / 2`). This weight accounts for the cost of traversing from the intersection point to the conceptual midpoint of the road segment (and vice versa). This modelling assumption treats road segments as entities having lengths which are shared equally at their ends. Thus, this in fact generated an initial graph consisting of all processed road segments and intersections. (Initial node count: ~12,492, Initial edge count: ~14,863 based on the provided code output analysis, but with numerical variations due to slight differences in final data).
3. **Network Cleaning via Connected Components**: As stated previously, the clipping process could create small, isolated road networks disconnected from the main İçərişəhər network. To retain analysis on the primary related network only, connectivity analysis of the constructed networkx graph was performed.
   - The `networkx.connected_components()` function identifies all distinct disconnected subgraphs in the overall graph.
   - These components were sorted by size (number of nodes). Within the processed dataset, typically one very large component (>99% nodes, e.g., 12,414 nodes, etc.) represented the main network, while several other much smaller components (6 components ranging from 7 to 30 nodes) represented isolated segments mostly found along the boundary of the study area.
   - The final road_graph for analysis and database ingestion was constructed from the subgraph corresponding only to the largest connected component; all nodes and edges from the smaller disconnected components were discarded.

- The final, cleaned graph consists of 12,414 nodes and 14,785 edges.

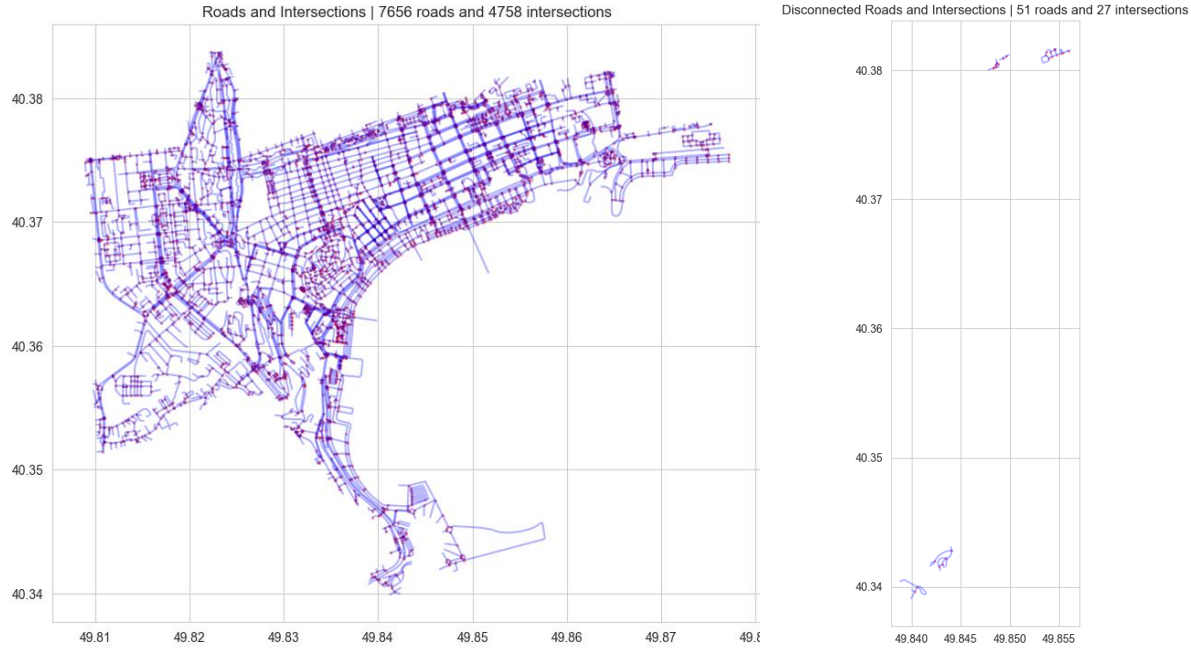

Figure 3.7. Main Road Network and Disconnected Networks. On the left: largest roads subgraph used for navigation. On the right: 6 filtered out smaller disconnected subgraphs.

3.3.4 Google Places POI Filtering:

The initial bulk of the POI pool, consisting of 11,109 entries collected via Google Places API, contained multiple entries irrelevant to urban exploration or tourism (administrative offices, private infrastructure, ATMs, generic names for residential buildings, etc.) and potentially some with very offensive names, containing slurs or being simply inappropriate. Manually checking for processing over 11,000 entries was impractical, thus a Large Language Model (LLM) was used for filtering in an automated way.

1. **Model Selection**: Initial uses of Google's gemini-2.0-flash model produced inconsistencies and unreliable output, often misclassifying relevant POIs, or conversely, failing to identify those irrelevant to the user. Thus, the newer model gemini-2.5-pro was chosen. The slowdown was worth it: far superior in reasoning, the gemini-2.5-pro gave much better results in both accuracy and context-awareness, even appearing to know local contexts, mixed languages, and potentially offensive slang pertinent to Baku.
2. **Process**: The POI's worked in batches (250 at a time). In each case, place name and ID lists were submitted at the same time to the Gemini API, all under a single ongoing conversation context. A carefully written prompt instructed the model to look at each place simply by its name and to classify it as either potentially offensive or irrelevant to general tourism/exploration in the Baku context, with a short justification for every classification. The serenity of maintaining one conversation context seemed to enhance consistency across batches.
3. **Results & Verification**: A large number of POIs have been classified by LLM as offensive or irrelevant. The application of such filters reduced the relevant POI dataset to 5,689. Careful sample manual checking between the filtered and retained sets confirmed the gemini-2.5-pro model was very accurate and reliable for this task.

3.3.5 Calculation of Cultural Relevance Metrics and Smoothing

In order to enable to "cultural" pathfinding mode, a quantitative "relevance" score was allocated to each road segment with reference to the proximity of filtered relevant POIs (from Section 3.3.4). This score tries to capture the "interest" of road segments and their utmost cultural significance. The process

involved calculation of POI-Road binary association, transformation of raw counts, and network smoothing.

1. **POI-Road Association**: Employing `geopandas.sjoin_nearest()` function, the closest processed road segment to each of the 5689 relevance POIs was defined within a projected CRS (EPSG:32639) for accurate calculation of distances.
2. **Initial Aggregation**: The numbers of relevant POIs close to each road segment were calculated using `groupby` method with counting.
3. **Transformation of the Score**: The raw counts generally display an extremely skewed distribution: while many roads will carry zero POIs, very few will carry many. To produce a more even score that can serve as a cost metric and differentiate between those roads populated with few POIs and those with absolutely zero, various transformations were employed:
   - **Reindexing and Offset**: Raw counts were reindexed to match the full list of road segments. Crucially, roads that had no associated POIs were given a fill value of -2 (`raw_counts.reindex(roads.index, fill_value=-2)`).
   - **Base Increment and Scaling**: A value of 2 was added to all counts (`intermediate_counts = raw_counts + 2`). Due to the `fill_value=-2`, roads with originally no POIs have now been accommodated with a value of 0, while the roads carrying one or more POIs have had their counts increased by 2. This was followed by multiplying by 2 (`intermediate_counts = intermediate_counts * 2`). This approach ensures that the base increment primarily affects roads with at least one POI, enhancing their score relative to zero POI roads.
   - **Clipping**: The resulting count was clipped to a maximum value of 50 (intermediate_counts = np.clip(intermediate_counts, 0, 50)) to limit the influence of roads with an exceptionally high number of POIs.
   - **Logarithmic Scaling**: A log base-2 was applied (`relevance_scores = np.log2(intermediate_counts.replace(0, 1))`). Before applying the logarithm, any counts that were 0 (or originating from roads with no nearby POIs) were replaced with 1, ensuring log2(1) = 0 for these roads. Logarithmic scaling design compresses the higher end of the score range, reducing the difference in score between roads with moderate versus high POI counts and relatively increased difference compared to the lower end.

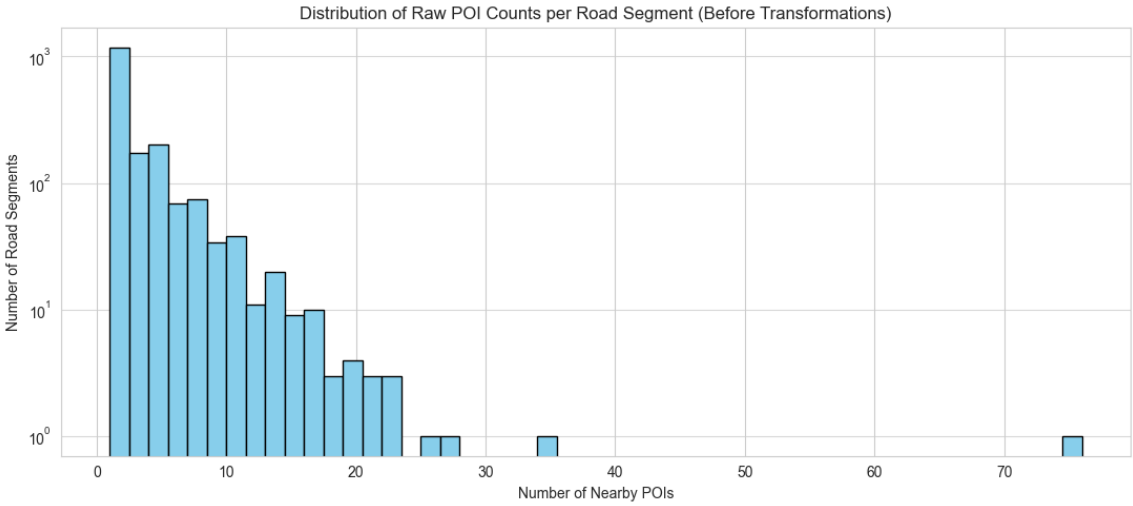

Figure 3.8. Distribution of Raw POI Counts per Road. A histogram showing initial distribution of number of POIs adjacent to each road segment.

4. **Road Network-Based Smoothing**: Another smoothing pass was to use a network-based algorithm, networkx graph from Section 3.3.3 was applied to propagate some "interest" to

neighboring segments, and hence roads near interesting roads are also likely to be of cultural interest.

- The relevance score calculated above was added as an attribute to each Road node in the network graph of networkx. Details of the smoothing process include:
    - Identification of the immediate neighbors in the graph (i.e., Intersection nodes.).
    - Identification of Road nodes that are connected to the immediate neighboring Intersection nodes (thus, Roads which can be reached through only one Intersection).
    - Calculating the mean relevance score of those Roads neighboring the current Road.
    - The current Road's relevance score was updated using a weighted average: new_relevance = 0.85 * current_relevance + 0.15 * mean_neighbor_relevance. In this case, the own road's score matters more with a little bit coming from its neighbors.
- The entire process was iterated twice as regards the whole graph to allow some of the effect to propagate further within the network.

5. **Final Inversion for Minimization**: For the A* algorithm to provide a solution, it minimizes costs. Hence, the smoothed relevance scores had to be inverted so that any highly "interesting" road (value of the raw score) would only incur a very small final cost. All values were determined directly by subtracting each road's smoothed relevance score from the maximum smoothed relevance score for roads (relevance = max(smoothed_relevance) - smoothed_relevance). The final inverted score determines the cost of cultural relevance used for the project.

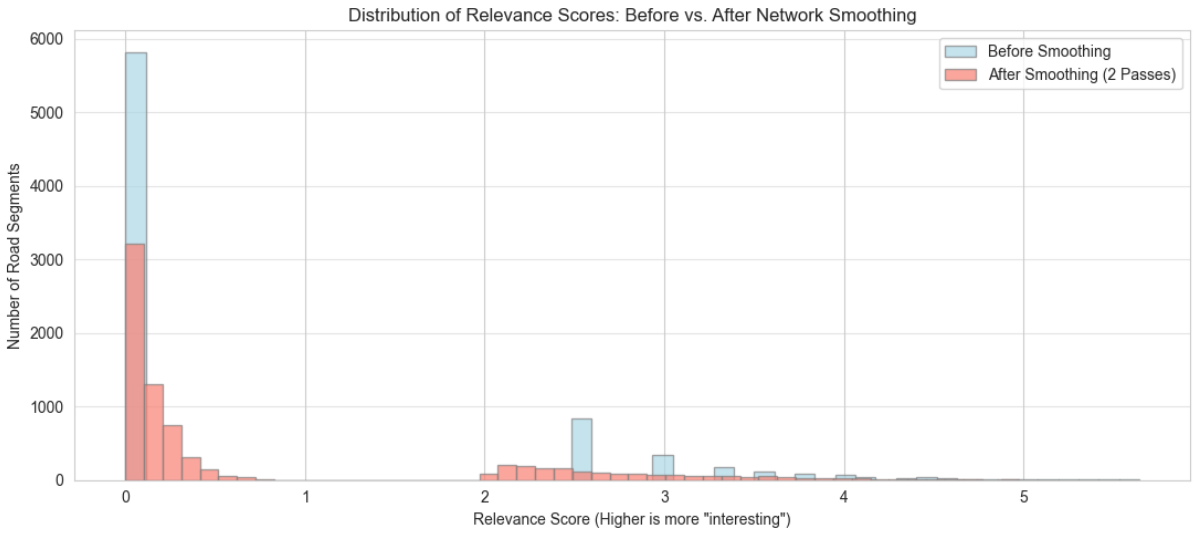

Figure 3.9. Distribution of Relevance Scores Before and After Smoothing. Two histograms plotted one over the other, comparing distribution of the relevance scores before and after the smoothing step.

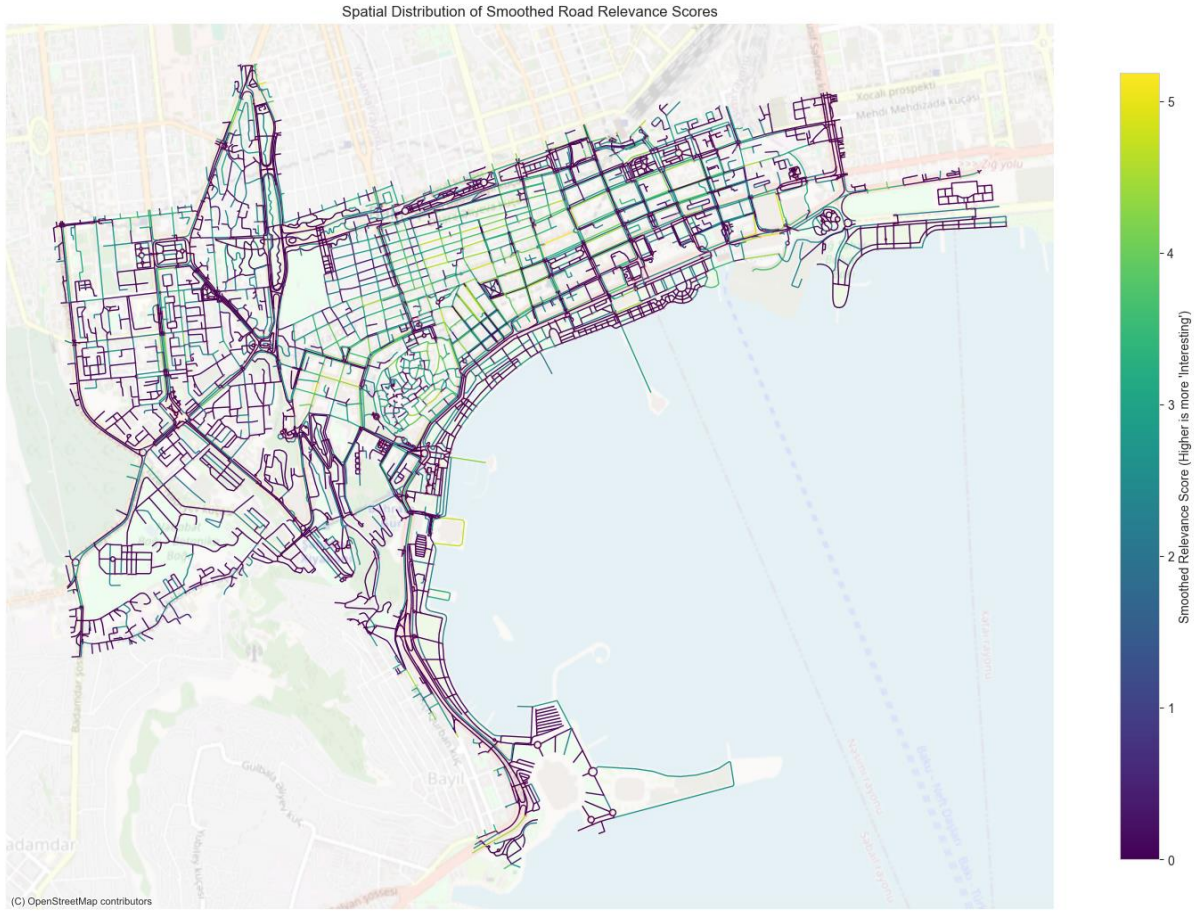

Figure 3.10. Map of Final Smoothed Relevance Scores. Map shows road segments colored according to the final, smoothed, and non-inverted relevance score drawn on the background of the OSM map tiles. The brighter the roads, the more priority it is given during the calculation of the "cultural" path.

3.4 Database Setup and Data Ingestion

The hybrid database approach was, therefore, considered appropriate to meet the distinctive needs of storing detailed geospatial features and querying network topology efficiently for pathfinding, with

PostgreSQL/PostGIS on one hand and Neo4j on the other. This section describes the setup and data ingestion process for both systems.

3.4.1 PostgreSQL/PostGIS Implementation:

PostgreSQL, along with the PostGIS spatial extension, is suitable choice, especially with its capability to handle geometry data types, spatial indexing, and mature relational features.

1. **Schema**: primary table names roads was set up to contain both processed road segments and their intersection points. Important columns include road_id (auto-incrementing primary key), label (unique name used for graph node names, e.g., 'r_123' or 'i_45'), name (commonly used name, when available), geometry (PostGIS geometry type storing LineString for roads and Point for intersections in WGS 84 - SRID 4326), length (calculated geodesic length for road segments), tags (this jsonb field stores consolidated OSM attributes), and finally, osm_id.
2. **Indexing**: To allow efficient spatial querying (e.g., getting the nearest road given a coordinate), a spatial index (`CREATE INDEX idx_roads_geom ON roads USING gist (geometry);`) was created over geometry column using GiST. Another standard index was created on both label and osm_id to improve the lookup times based on these identifiers.
3. **Data ingestion**: The processed road segments (`roads_from_graph`) and intersection points (`intersections_from_graph`), available as GeoPandas GeoDataFrames after processing, were loaded into the roads table using a Python script whereby the SQLAlchemy library (via the helper class PgEngine) was used to connect to the database, and the `GeoDataFrame.to_postgis()` method was used to perform the actual loading. The `if_exists="append"` option was used for adding data without dropping the existing table structure. The dictionary presented in the tags column was serialized into a JSON string prior to loading, such that it would be compatible with the jsonb database field.

3.4.2 Neo4j Implementation:

The native graph database Neo4j was settled on because of its optimization for graph traversal, a necessary property for success in pathfinding.

1. **Graph Model**: The graph model that was put in place in Neo4j consists of:
   - **Nodes**: These are labeled as either :Road or :Intersection. Each node must have a name property that corresponds to the name given in PostGIS and the networkx graph (e.g., "r_123," "i_45"). Nodes contain properties: lat and lon (derived from geometry centroid) necessary for the A* heuristic function in Neo4j GDS. :Road nodes also carry the length property.
   - **Relationships**: One relationship connecting :Intersection nodes to connecting :Road nodes was put in place. Their initial property length represents half the length of the connected road segment (which coincides with the networkx edge weight).
2. **Initial Data Ingestion**: The final cleaned up networkx graph (road_graph in Section 3.3.3) was used as the source for the first data load into Neo4j. A Python script (function `import_networkx_to_neo4j`) connected to the Neo4j database using the official Neo4j Python driver (via `MyNeo4JDriver` helper class). This script iterated through the nodes and edges of the networkx graph:
   - Cypher MERGE queries were executed within a single session to create :Road and :Intersection nodes, setting their name, lat, lon, and (for roads) length properties. MERGE prevents duplicate node creation if the script is run more than once.
   - Cypher MATCH and MERGE queries were executed to create the :CONNECTS_TO relationships between the respective nodes, and the initial length property on the relationship was set.
3. **Relevance Score Update**: The calculated and smoothed relevance scores (Section 3.3.5) were then extended into the graph to enable cultural pathfinding after the initial graph structure and

lengths were loaded. A different Python program (function `update_neo4j_relevance`) was run to adaptively update the graph in batches by:

- The driver connects to Neo4j.
- Batched Cypher queries are executed using UNWIND. For each road in the batch, the query matches the corresponding :Road node by its name and SET its relevance property. It can optionally match the :CONNECTS_TO relationships connected to that road node and also SET their relevance property to the same value. This guarantees the relevance score will exist on both nodes and relationships, thereby serving potential use by different GDS algorithm configurations or direct Cypher queries.

The hybrid implementation provides PostGIS, which stores urban features' rich geometrical and attribute information; whilst Neo4j is a graph database optimized for network topology traversal and pathfinding populated with relevant properties, length, and calculated relevance score.

3.5 Execution of the Pathfinding:

With the processed data loaded into a hybrid database system of PostGIS and Neo4j core pathfinding functionality was implemented to provide three distinct types of routes: Shortest Path, Cultural Path, and Geocaching Path. The A* algorithm was chosen for a number of reasons: it is efficient for finding least-cost paths over graphs with weighted edges; it can incorporate heuristics to guide the search; and optimal resolution of A*'s execution within the library provided by Neo4j graph data science (GDS).

3.5.1 Neo4j GDS Setup:

Whenever graph algorithms are run in Neo4j, creating an in-memory graph projection with GDS is favored to load a graph into memory for faster traversal, which is critical for responsive pathfinding. So before any pathfinding could be done, the following Cypher command, gds.graph.project, was executed to load the relevant parts of the storage graph into memory:

```
CALL gds.graph.project(
  'myGraph',
  {
    Road: { properties: ['lat', 'lon'] },
    Intersection: { properties: ['lat', 'lon'] }
  },
  {
    CONNECTS_TO: {
      properties: ['length', 'relevance'],
      orientation: 'UNDIRECTED'
    }
  }
)
YIELD graphName, nodeCount, relationshipCount;
```

And that created a named graph projection (myGraph) which consists of nodes (:Road and :Intersection) and related latitude/longitude properties (to support the use of the A* heuristic in assessing the distance to the target) and undirected cost relationship, this new: CONNECTS_TO

relationship presents length and relevance attributes calculated earlier. It significantly accelerates the computation of the algorithm when it is run on that in-memory graph projection, compared to being pulled out from the on-disk storage.

3.5.2 Core Function for A Pathfinding*

The primary storage for A* pathfinding function to `find_shortest_path_astar` was developed as a generalized function to interact with the actual A*/Neo4j GDS library implementation. This function satisfies different criteria, as appropriate, to return the path.

1. **Inputs**: This function takes the unique name (label) of the start node, the name of the end node, the name of the GDS graph projection (`'myGraph'`), and the `cost_property` string (either 'length' or 'relevance') that specifies the edge feature to minimize.
2. **Execution**: In brief, the `gds.shortestPath.astar.stream` procedure is implemented and executed according to the A* search, which employs the specified `cost_property` to evaluate path segments and uses the node latitude/longitude properties in calculating the heuristic (i.e., the estimated distance left), in order to guide the pathfinding toward promising paths input and effectively prune the search space. Result from anything executed is what the script specifies at the bottom of the code, either return or raise exception.
3. **Output**: If a path is found by the GDS during the execution, the function outputs to the program a dictionary indicating the `totalCost` of that path (sum of `cost_property` values of edges in the path) and `nodeNames`, which is the list of node identifiers (r_... or i_... labels) in order they occur in the path from source node to destination node. If GDS could not find a path between the two specified nodes, the next GDS would appear as returning a 'null' value from the Neo4j back to the Python processing.

3.5.3 Path Type Implementations:

Using the core A* function and modulating it with specific configurations or combinations obtained the three types of paths required:

1. **Shortest Path**: Primarily, the route minimizes the aggregated geodesic lengths of roads for the shortest road path generated by invoking `find_shortest_path_astar` with `cost_property = 'length'`.
2. **Cultural Path**: In this rich cultural path, the cost function is now inverted and smoothed, as detailed in Section 3.3.5. A lowest inverted relevance value with lesser relevance costs guides the search.
3. **Geocaching Path**: This path deviates the route for visiting particular geocaches which are chosen based on the user's start and destination geocaching search criteria. The procedure simply emphasizes simpler solutions and will not try to find an optimally combined path that would cover each geocache instead.
   - **Pre-calculation**: For each of the geocaches, PostGIS was used to look up the nearest road network node (label) with spatial queries. Such temporary storage would make it easier to search for closer nodes when needed during path calculations.
   - **Runtime Logic**:
     A. Fail safely locate the user's reference path (usually shortest) from start (A) to endpoint (B).
     B. Spot the geocache closest to this reference path geometry with the aid of calculated distances using a projected CRS.
     C. Fetch the nearest road node label (C) pre-computed for the geocache in question.
     D. Run two `find_shortest_path_astar` shortest path calculations for length considering the cost property, one from A's nearest node to the node labeled C and then another from C to B's nearest node.
     E. Combine the resulting node-name sequences: (`A -> C -> B`). This simple approach ensures the cache is visited but requires two pathfinding calls and cannot stake any

guarantee as regards offering the shortest possible path involving any cache, which means this solution represents a trade-off between optimization design and implementation simplicity.

3.5.4 Retrieving Path Geometries for Use:

It is important to note that the set of Neo4j GDS pathfinding procedures work on the graph abstractly but return the list of node identifiers (labels) and the ultimate cost only to illustrate on a map or to provide navigation directions-in detail such as down-to-earth geometry and attributes of the various road segments or junctions. Thus, after attaining the list of ordered node names from Neo4j for a given path, another query needs to be performed by the bundle to access the complete records from the roads table in PostGIS (e.g., via `get_roads_by_labels` helper function) speaking to those labels. These are just some of the spatial records that contain relevant geometries of lines or points and extra attributes (like names and tags of the segments) to follow up in the same order produced by Neo4j to recreate the entire geographically-correct path. This last retrieval step is where utility functions are for these tasks and can be located within the backend API layer (Section 3.6), allowing the reprocessing of quantities and pass them back to the front API.

3.6 Backend API Implementation

A backend Application Programming Interface (API) was implemented to expose pathfinding functionality and orchestrate data processing and accessing the data layers in the system. The backend API is built in Python, using FastAPI. The reason for preferring FastAPI is for its modern features within high performance (especially built on ASGI), automatic generation of interactive documentation (via OpenAPI and swagger UI), and built-in data validation through Pydantic models.

3.6.1 API Structure and Initialization:

The structure of the rear application (main.py) is to orchestrate interactions between processed requests and lower database systems (PostGIS and Neo4j).

1. **Database Connections**: Setting up the application includes instantiation of connection handlers for both PostgreSQL/PostGIS (PgEngine) and Neo4j (MyNeo4JDriver). The connection parameters (host, port, user, password, database name) are passed via environment variables for maximum configurability, particularly within the containerized environment (Section 3.7).
2. **Neo4j GDS Graph Projection**: Our application will be required to verify the existence of a relevant Neo4j GDS graph projection (myGraph) at the beginning of its life cycle probably by executing `gds.graph.project` (perhaps dropping an existing projection first). This guarantees that the in-memory graph for pathfinding is in a state whereby it can accept queries.
3. **Geocaching Data Preparation**: The geocaching data is first loaded from an appropriate JSON file, GeocachingData.json, then processed into a GeoDataFrame, and most importantly, each geocache has its closest road network node label (closest_road) pre-computed using the find_closest_road function. Pre-computing such information directly enhances the runtime performance of geocaching path generation.
4. **CORS Configuration**: The application configuration allows for the CORS middleware (`fastapi.middleware.cors.CORSMiddleware`) to accept requests from specific origin(s) of web-based frontend applications. Example: http://localhost:8001 during development session.

3.6.2 Core Logic and Helper Functions:

The following are the helper functions (described in Sections 3.4 and 3.5 and, for the most part, in the appendix/code) employed by our application to manage database interactions and path calculations:

- `find_closest_road()`: This queries PostGIS to find the nearest road node label to a particular coordinate point.

- `get_roads_by_labels()`: Queries PostGIS to retrieve full road/intersection records based on a list of node labels.
- `find_shortest_path_astar()`: Executes the A* pathfinding query against the Neo4j GDS projection, returning the node sequence and cost.
- `get_all_paths()`: Orchestrates the calculation of all three path types (Shortest, Cultural, Geocaching) and identifies the closest geocache, calling the necessary helper functions.

3.6.3 API Endpoints (/get_routes):

The Existence of this API enables a POST method that will return the wanted data to our application. Such an endpoint will be accessed, either directly or through a frontend.

1. **Request**: A JSON payload containing the (lon, lat) coordinates of the start and target gets into the endpoint. This structure is extensive to facilitate validation using Pydantic in a model (StartTarget).
2. **Processing Workflow**: Provided the request is valid, the end-point shall:
   - Convert input coordinates to `shapely.Point` objects.
   - Invoke `get_all_paths` with start and end points and the initialized database drivers and geocache data. This function functions by:
     - Finding the nearest road nodes in PostGIS with regard to the start/end points.
     - Executing A* queries against Neo4j GDS for the shortest and cultural paths.
     - Geocaching path logic - find closest cache and run the two shortest-path queries from A to cache and cache to B (`A -> Cache -> B`)
     - Return detailed node data (including all properties) from PostGIS for all calculated path sequences.
     - Reorder a result from PostGIS to follow path sequences from Neo4j.
     - Finally, return results for all three paths with closest geocache information.
   - Finally, serialize these results from `get_all_paths` into a final JSON response format, including serializing geometry objects into WKT format for JSON compatibility and compaction.
3. **Response**: A simple JSON object is sent back with
   - `shortest_path`: A list of dictionaries, each representing a road/intersection segment in the path (label, name, tags, and geometry as WKT).
   - `cultural_path`: Similar list for the cultural path.
   - `cache_path`: Similar list for the geocaching path.
   - `closest_cache`: A dictionary containing details about the geocache that is closest to the user's route (ID, name, description, task, position as WKT, etc.).

3.6.4 Root Endpoint(/):

A conducive endpoint (/) infrastructure is graciously streamlined into simplicity. It serves to provide primary health checks and diagnostics by furnishing any basic information necessary to its very extent of services in terms of configuring the hosts for database connections.

This FastAPI backend encapsulates complicated logic that communicates with the hybrid database systems, execute our pathfinding algorithm, preparing results in structured consistency suitable for a front-end.

3.7 System Containerization and Orchestration

A containerization strategy ensures the navigational system's portability, reproducibility, and ease of deployment on different environments using Docker and Docker Compose. This encompasses the backend API and database systems (PostGIS and Neo4j) while encapsulating each component and its dependencies into isolated containers, which are easy to install and prevent hard to resolve conflicts with software versions or system libraries.

3.7.1 Dockerizing the FastAPI Backend

The FastAPI backend application was containerized by building the Dockerfile for the FastAPI backend application. The Dockerfile contained types of code that would define the necessary steps and procedures for constructing a standalone runtime containing the application code, Python runtime, essential systems, and Python libraries.

1. **Base Image**: A lightweight official Python image based on Debian Bookworm (`python:3.12-slim-bookworm`) was selected for the environment to keep image size to a minimum while running over a recent OS environment.
2. **System Dependencies**: To date in Docker build are such system libraries that take care of potential complications for the Python dependencies of the application. These include the following: `build-essential`, useful only for the compilation of packages; `libpq-dev` for the `psycopg2` PostgreSQL adapter; `gdal-bin`, `libgdal-dev`, `libproj-dev`, and `libgeos-dev` libraries to deal with GDAL services of geopandas and Shapely.
3. **Python Dependencies**: A Python `requirements.txt` file is copied into the image, listing all the packages needed by the application. The pip install then runs once the file is copied and installs the requirements while passing the `--no-cache-dir` option flag for minimal image growth.
4. **Application Code and Data**: The real application source code (`main.py`) and the data files, be it the data directory containing the geocaching information, were all copied into the /app working environment on that container.
5. **Port Exposure**: Hence exposing port 8000 being the default port for the uvicorn server for running FastAPI.
6. **Run Command**: `CMD` as it specifies the actual command that should hence be invoked once the container is instantiated from the image: `uvicorn main:app --host 0.0.0.0 --port 8000`. This way, the Uvicorn ASGI server runs on the FastAPI application (i.e., the app instance within main.py) by opening port 8000 for listens on all network interfaces.

3.7.2 Orchestration with Docker Compose

Contrary to a Dockerfile that delineates how to build an image, Docker Compose allows for defining and managing the multiple-container application environment. This comprises the services of the application and their database dependencies of PostGIS and Neo4j. The docker-compose.yml file orchestrates the setup and networking of these services.

1. **Service Definitions**: `postgis`, `neo4j`, `app` are essential services being defined.
2. **`postgis` Service**: Considering `postgis/postgis:latest` as the base image Defines environmental variables: `POSTGRES_USER`, `POSTGRES_PASSWORD`, and `POSTGRES_DB`.
   - Maps host port 5432 to the container port 5432 to the potential to be able to connect from the outside or further debugging the database setup.
   - Sets up a named volume named `postgis_data_ugr` that mounts to `/var/lib/postgresql/data` to store the PostgreSQL data which may persist across containers.
   - Also includes health check logic to ensure some signals representing when the database is up and instruction dependent services like load balancers to start.
3. **`neo4j` Service**: Uses a `neo4j:enterprise` image (requires acceptance of the license agreement for development/research use).
   - Sets the `NEO4J_AUTH` variable for the purpose of authentication (`neo4j/neo4j_password`).
   - Configured `NEO4J_PLUGINS` for automatically including the graph-data-science and apoc libraries required by the application.
   - Almost too obvious to state, the setting `NEO4J_ACCEPT_LICENSE_AGREEMENT` is `yes`.
   - Maps host ports 7474 (Neo4j Browser) and 7687 (Bolt driver connection) to the respective container ports.

- Uses named volumes (`neo4j_data_ugr`, `neo4_logs_ugr`, `neo4_conf_ugr`) for persistent storage of data, logs, and configuration.
- Includes a health check to verify the Neo4j server is serving replies.

4. **`app` Service (FastAPI Backend)**:
   - Specifies "`build: .`" , the Command allows Docker Compose to build the image from within the Dockerfile found in this current working directory.
   - Maps host port 8000 to container port 8000.
   - Defines important environment variables that overwrite defaults or configure the app within the Compose network environment. For instance, `POSTGRES_HOST` is set to `postgis` while `NEO4J_URI` is set to `bolt://neo4j:7687`, making use of the service names set up inside the Compose file. Those service names would be resolved through Docker Compose internal DNS into respective container addresses. Database credentials for the rest should then be passed through an environment variable.
   - Include a `depends_on` clause with condition: `service_healthy` for both postgis and neo4j. This will enable the backend app container to only start once the database containers pass the health check, making sure no connection errors occur at startup.
5. **Named Volumes**: These named volumes exist for use with the database services and enable storage of data between containers in the event of container stop-then-removal scenarios.

This containerization architecture helps develop and test seamlessly and even simplifies potential deployment: with one command, "`docker-compose up`", the whole backend system is brought up including its complex database dependencies and respective bespoke configurations such as Neo4j plugins.

4 RESEARCH RESULTS AND ANALYSIS OF RESULTS

This section shall report the results obtained at the implementation of the methodology discussed in Chapter 3. The scope of this section encompasses results from the data processing pipeline, generation characteristics of navigation paths, performance evaluation of the backend system and successful containerization of the application components.

4.1 Data Processing Results

The data acquisition and processing pipeline have converted raw geospatial data into formats that are now usable by the hybrid database system and subsequent pathfinding operations.

4.1.1 Study Area and Network Processing

After interactive refinement, a final polygon (shown in Figure 3.2) defining the study area was created, including İçərişəhər and the most important surrounding areas.

- This initial extraction of OSM 'walking' network within this polygon contained 3,437 road features. The first shape and composition is shown in Figure 3.4 (left panel).
- On applying `shapely.linemerge` and then `geopandas.explode` to sort out MultiLineString geometries (refer Section 3.3.1 and Figure 3.4, right panel), the dataset became 3,456 different LineString road features.
- The intersection finding and road-splitting process undertaken on these segments (Section 3.3.2). As figure 3.4 represents, the graph initially constructed from these split segments and their intersection points consisted of 12,492 nodes (7,707 road nodes and 4,785 intersection nodes) and 14,863 edges.
- The network cleaning through connected components analysis revealed one large primary network, while 6 smaller disconnected components representing the cleared-out smaller components were filtered out from the final network graph. The final network graph for pathfinding consisted of 12,414 nodes (7,656 road nodes and 4,758 intersection nodes) and 14,785 edges (:CONNECTS_TO relationships). This main network component is visualized in Figure 3.7 (left panel).

- Thus, during the cleaning process, 78 nodes (51 "road" nodes and 27 "intersection" nodes) and 78 edges were discarded, part of the 6 smaller disconnected components. These discarded smaller components are depicted by Figure 3.7 (right panel) .
- The discrepancy with which lengths were calculated (Section 3.3.1) resulted to considerable, illustrated wrongly spatially as Figure 3.5; the segments with high error (discrepancy >40%) are concentrated at boundaries or due to complex geometry splits (Figure 3.6). This established the need for recalculating of the geodesic length.

4.1.2 Filtering of Point of Interest (POI)

- The sampling design employed to create the hexagonal grid (fig. 3.3) resulted in a total of 11,109 unique POIs, as indicated by the Google Places API.
- Automated filtering based on the gemini-2.5-pro LLM delineated and listed 5,170 POIs as not relevant or offensive (43 counted among them as possibly offensive), leaving a body of 5,689 POIs used in calculations of relevance.

4.1.3 Calculation of Relevance Metric

- The nearest road segments initially assigned to the POIs-the 5,689 relevant ones-have shown very uneven distributions; a fair number of roads have zero associated POIs as seen in Figure 3.8.
- Thus, the transformation and network-based smoothing process (ref. Section 3.3.5) establishes this final relevance score. Figure 3.9 compares the cases of the distribution of relevance scores before and after both smoothing passes to show just how extreme values have been considerably reduced, with only a slight broadening of the distribution.
- Descriptive statistics that concern relevance scores (the last inversion step for A* notwithstanding) are listed in Table 4.1.

Table 4.1: Statistical Characteristics of the Descriptive Road Relevance Scores (Before Inversion)

Statistic	Original Score (After Log Transform)	Smoothed Score (2 Passes)
Count	7656.0	7656.0
Mean	0.7522	0.7505
Std Dev	1.3831	1.1869
Min	0.0	0.0
25% (Q1)	0.0	0.0024
50% (Median)	0.0	0.1251
75% (Q3)	0.0	0.5419
Max	5.6439	5.1832

- As seen in Figure 3.10, this is how the last spatial distribution of remnant smoothed relevancy scores - where greater values indicate higher "interestingness," prior to inversion - appears in space. The cost property for cultural pathfinding was built on these smoothed and inverted scores.

4.2 Pathfinding Results

This section assesses the performance and functionality of all three pathfinding modes developed, namely: Shortest Path, Cultural Path, and Geocaching Path. Each mode is evaluated from the point of view of path length, number of nodes, and qualitative visual differences by comparing the routes generated through three pairs of randomly selected start (A) and end (B) points over different areas of the İçərişəhər study area.

4.2.1 Example Path Set 1

These routes were generated between a start at the western end of the study area and an end toward the southeast, passing importantly through almost all the central region.

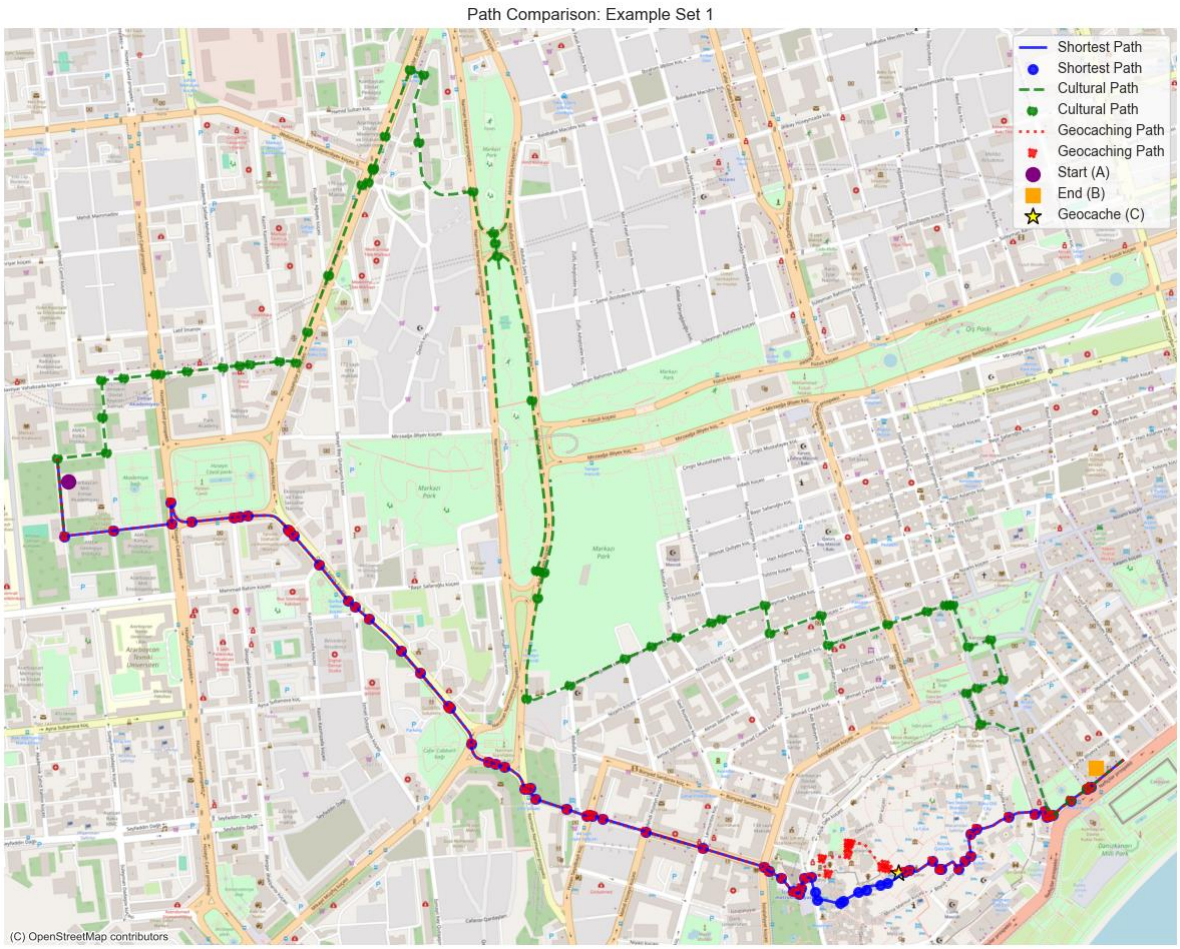

Figure 4.1. Path Comparison for Example Set 1. This map, generated from system outputs, displays Example Set 1 with Shortest Path (blue), Cultural Path (green), and the combined Geocaching Path (red). Marked are the start (A, purple), end (B, orange), and selected closest geocache (C, yellow star).

Table 4.2. Path Characteristics for Example Set 1

Metric	Shortest Path	Cultural Path	Geocaching Path
Total Length (meters)	3343.3	5121.5	3427.4
Node Count	139	119	142

As seen in Figure 4.1, these distinct routes presented by different algorithms depend on Shortest Path (blue), which takes a relatively direct route. Cultural Path (green) deviates significantly, taking a much longer route (5121.5 m against 3343.3 m approximately 53% longer) apparently towards a northward turn before heading toward the destination which suggests it prioritizes areas being calculated as having a higher relevance to the areas.

Interestingly, the Cultural Path uses fewer nodes (119 against 139), probably as an indication of using much of the long and major road segments within the culturally relevant areas which it traverses. The Geocaching Path (red) is here pretty close a replica of Shortest Path, though including the nearest geocache (C) leads only to a minor extension in length (3427.4 m against 3343.3 m, about 2.5% longer) and similarly with the node number being hardly different (142 against 139)

4.2.2 Example Path Set 2

The second example involved the longest distance from near the northwestern part of the study area to the south-eastern coast (Baku Bay).

Figure 4.2. Path Comparison for Example Set 2. This map depicts the Shortest Path (blue), Cultural Path (green), and Geocaching Path (red) for Example Set 2 plus the points of start (A), end (B), and selected cache (C).

Table 4.3. Path Characteristics for Example Set 2

Metric	Shortest Path	Cultural Path	Geocaching Path
Total Length (meters)	6377.3	7078.2	6664.6
Node Count	189	151	212

In this case therefore for the longest route (Figure 4.2), the Shortest Path (blue) again takes direct route through the most direct available network. For the huge part of the distance, the Cultural Path (green) has a distinctly different southern route along the coastal boulevard area and then heads north for the terminus. The result is a longer path (7078.2 m against 6377.3 m, about 11% longer), but again uses fewer nodes (151 against 189), seemingly preferring more major or scenic routes in the relevant zones.

The Geocaching Path (red) thus integrates the chosen cache (C), located south of the shortest path which resulted in a moderate increase in length (6664.6 m against 6377.3, about 4.5 percent longer) and has a higher node count (212) reflective of the detour for the cache point.

4.2.3 Example Path Set 3

This third sample has start and end points more centrally located within the studied area, traversing the denser network of the Old City and its immediate surroundings.

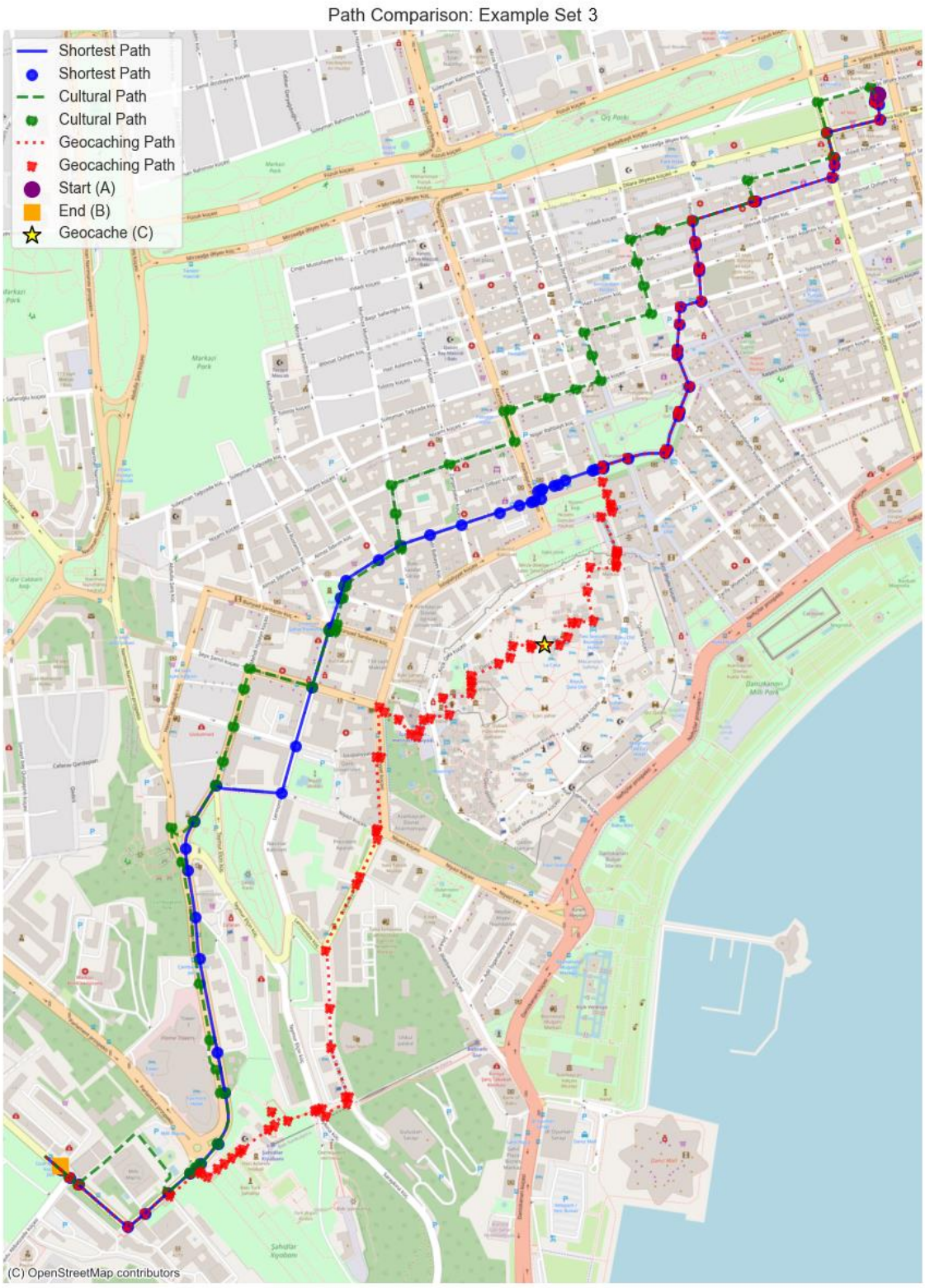

Figure 4.3. Path Comparison for Example Set 3. This map shows Example Set 3, Shortest Path (blue), Cultural Path (green), and Geocaching Path (red), as well as start-point (A), the endpoint (B), and selected geocache (C) points.

Table 4.4. Path Characteristics for Example Set 3

Metric	Shortest Path	Cultural Path	Geocaching Path
Total Length (meters)	3304.1	3607.3	3662.7
Node Count	151	107	214

The Difference was less in the overall direction for example (Figure 4.3) in this central area between Shortest Path (blue) and Cultural Path (green) but still between the streets chosen. The Cultural Path is a bit longer (3607.3 m against 3304.1 m about 9 percent longer) and uses significantly fewer nodes (107 against 151), again suggesting more attention to potentially more significant or direct routes through culturally relevant areas, as opposed to smaller or less relevant segments. The Geocaching Path (red) detours to include the cache (C) producing a length similar to that of Cultural Path but longer than Shortest Path (3662.7 m against 3304.1 m about 11% longer) and the highest node number (214) representing the added segments of the A->C and C->B routes.

4.2.4 A Summary of Path Characteristics:

The analysis of these three examples thus illustrates the capacity of the system to produce qualitatively distinct routes within given optimization criteria.

- Its **Shortest Path** invariably constitutes the minimum distance towards any other destination, hence providing a base for comparison.
- The **Cultural Path** optimizes relevancy as computed, and thus successfully diverts from the shortest route, as well as likely towards more integrated-use pathways through POI-dense or probably more scenic value (such as the boulevard in Example 2). Another interesting comment: while sometimes having longer distances, the Cultural Path uses fewer nodes (segments/intersections) than does the Shortest Path. This might indicate more likely following richer or stretched major routes - within the relevant zones - at some less relevant areas rather than many smaller turns. Still, a deeper set of studies is needed to take this up.
- Thus, **Geocaching Path**, as best, captures that tiny detour to the nearest relevant geocache, producing a path just a bit to moderately longer than the shortest and a node count reflecting that split into two segments.

The visible differences as depicted in Figures 4.1, 4.2, and 4.3 coupled with the quantitative figures in Tables 4.2, 4.3, and 4.4 confirm that the implemented A* variations and the derived relevance metric successfully produce routing options that can be distinctly tailored to different user intentions (efficiency, cultural exploration, or geocaching).

4.3 Performance Evaluation

Another key step is to evaluate the performance of the back-end system and assess the extent of its responsiveness to users, under stress. The detailed benchmarking at various load conditions was out of the scope of the current thesis although several observation performance timings were recorded on the local development environment. These results were indicative of the responsiveness of the system.

The vital performance criterion is the backend API call that requests path computations from the start to the endpoints. This involves various steps: querying the network edge nodes near the start/end coordinates (done by PostGIS queries), running the A* network-graph path-finding algorithms (Neo4j GDS queries) over the Neo4j's graph representation, fetching the geometries of the paths (another PostGIS query) and finally formatting responses.

- Neo4j GDS A* Performance: The core pathfinding calculations done through Neo4j Graph Data Science library, among other optimizations, with A* implementation performed very efficiently. Individual path queries (for shortest or cultural paths through usual start/end nodes within the study area) for the most part would range from 20 on average to 40 milliseconds for more distant points.

This brief time of computing repeated itself for pathfinding on length and custom-weighted cost properties because it seemed that the cost functions had little impact on the needed runtime performance of the GDS query in this graph structure.

- Overall, API Endpoint Response Time: The entire round trip starting from a foreordained request into `/get_routes` and ending with return of all three types of paths (i.e., Shortest, Cultural, Geocaching) and closest Geocache information, averaged around 200 milliseconds. This involves:
    - Initial PostGIS queries for the finding of the nearest start and end nodes.
    - Four, separate A* pathfinding calls to Neo4j GDS.
    - PostGIS, requires for the geometries of all nodes on the final paths.
    - Data processing and serialization into a JSON response in the FastAPI application.

The observed round-trip response of about 200ms is in the acceptable range of interactive application response times, confirming the feature of handling very quick and near-real-time route generation for end-users. The 20 to 40ms time recorded by the A* implementation of Neo4j GDS has played a pivotal role in achieving these results. The hybrid database approach appears capable of serving this application requirement in the logged environment where our tests had proved it adequate for the task.

It must be highlighted that these timings are instantaneous and valid only for a development environment without any concurrent user load. In production, performance with concurrent upsurges of requests may vary. But the results give a strong comfort proposition about the suitability of the architecture and algorithm seeds for the future urban exploration navigation system. Possible future optimizations (e.g. cache strategies, more efficient queries) must be tested if heavier load scenarios form a great demand.

4.4 Deployment of the System

After fine-tuning the procedure, the complete architecture where Doker was used with respect to the backend application and Docker Compose with respect to the multi-container environment was implemented successfully. The `docker-compose up` command was executed with the FastAPI backend image built using the Dockerfile and, most importantly, the launching of the three containers, including app, postgis, and neo4j, through the `docker-compose up`. All requisite configurations were in sync with the specific instructions with respect to the `docker-compose.yml` file, including the database connection environment variables, storage of persistent data, port forwarding, and the installation of Neo4j plugins (GDS and APOC).

Services operating within the Docker Compose network are crucial since the backend to FastAPI could communicate with the PostGIS container through the service name, PostGIS, and could connect to the Neo4j container through the Bolt URI, which was `bolt://neo4j:7687`, as instructed for the production environment by environment variables. Health checks in the docker-compose file helped to ensure that the database services were ready for running before the backend application started, which therefore prevented connection errors during initialization. The API endpoints of the backend, expected to work properly, including the `/get_routes`, were tested and worked fine in the containerized setup when both databases were queried and respective route results retrieved.

The same could be proved to be a positive confirmation for containerization in this context. The fact shows that the entire backend infrastructure can be deployed and relied upon in isolation under a controlled reproducible environment with a single command hence removing many humps that could slow down a developer attempting to work with the system or scientist speedy development. This stands out as an assurance to the ease of introspection of the system on testing or use-case point of view.

4.5 Discussion

The findings in this chapter demonstrate successful development and deployment of core backend services for the İçərişəhər urban exploration navigation system. This discussion interprets the results, outlines the main challenges faced, and presents an evaluation of the outcomes in the context of the initial research questions.

4.5.1 Interpretation of Results:

Results have been quite encouraging in respect of all of the target outcomes. Data processing pipeline (Section 4.1) effectively channeled raw OSM and Google Places data into structured formats which can be recognized by the hybrid database system. Specifically, the pipeline tackled practical issues like geometry inconsistencies in OSM (Figure 3.4) and length discrepancies too (Figures 3.5, 3.6); and used the LLM (`gemini-2.5-pro`) to filter the relevant POIs, halving the initial 11,109 POIs (resulting in 5,689 actual entries). And later, the calculation and smoothing of the relevance metric (Table 4.1, Figures 3.9, 3.10) led to spatially distributed values, aimed at finding cultural interestingness values (Fig. 3.10) also achieved.

The pathfinding setup (Section 4.2) generatively came out with three conceivable route types using various A* algorithm variations as per Neo4j GDS setup. Actual visual comparisons (Figures 4.1, 4.2, 4.3) and quantitative efforts (Tables 4.2, 4.3, 4.4) attest to a picture, whereby Cultural Path is markedly configured by the relevance and, as such, almost never does it follow the Shortest Path but rather opts for longer routes with potential engaging routes. Cultural Path inclines toward the smallest number of graph nodes (segments) - an indicative sign that it almost favors major or continuous routes within related areas; however, more investigation is required in this respect. Geocaching Path also managed to accommodate the temporary visit nearest relevant geocache well, showing the system's flexibility.

Observational performance (Section 4.3) evaluation showed a responsive system, with API response times generally falling around 200ms and core A* pathfinding happening within 20-40ms. These time signals that the hybrid architecture and the algorithms chosen go a long way in supporting a responsive user experience. This was confirmed by the successful containerization of the system using Docker and Docker Compose (Section 4.4); its first indication revealed flexibility rather than rigidity.

4.5.2 Challenges Faced:

Implementation came with many challenges. Through and through, the inherent variability and the potential for inaccuracies with crowdsourced OSM data meant that much attention had to be paid to cleaning and validation. The Google Places pose a constraint against the effectively unlimited use of their API, thus revealing the need to develop the spatial sampling strategy with hexagonal grid units; POI filtering then delivered a practical challenge on the difficulty employing large and unfiltered external data. While LLM-based filtering with `gemini-2.5-pro` proved to be a success, model choice and prompt engineering were essential here. Developing the relevance metric stood as inherently heuristic; the specific transformations put in place and smoothing parameters resulted from iterative refinement for an even-handed, spatially realistic distribution, albeit with another possible set of formulae leading to a different likelihood. Finally, the hybrid database system management needs to be watchful of data consistency between PostGIS (holding geometries) and Neo4j (holding topology and pathfinding weights), something that has been regulated through introduced workflows.

4.5.3 Relevance to the Research Questions:

The findings provide direct responses to the research queries posed right to the introduction in Section 1.2. The research has successfully developed and demonstrated, in brief, a complete backend system that can support multi-criteria urban navigation within İçərişəhər by overcoming challenges with data integration while leveraging a hybrid database design. This research shall solidify the chosen approach and provide a solid platform on which to build the project in the future, serving as the basis for user-focused evaluations.

4.5.3.1 RQ1 (Extending A)*

The research has shown how A* can be extended effectively in Neo4j GDS by including different cost properties. Specifically, working with relevance property, which we had designed from the POI data and network smoothing, made A* work to develop "cultural" paths that otherwise are out of alignment with criteria of length. Eventually, the backend had an extended graphing mechanism via a well-done payload construct upon this model. Validation of performance of data pipelines (Section 4.3) confirmed

the effective computational plan in meeting the project goals. The approach, depending on some conventions, supplies the choice for configurable cost properties shaping up around other heuristic search methods or cost functions.

4.5.3.2 RQ2 (Data Integration)

The hybrid database approach has proved itself as effective for aggregating into navigation various scattered geospatial data. PostGIS managed the storage and spatial queries over the specific boundaries of road and intersection geometries supplied by `get_roads_by_labels`, whereas Neo4j performs the handling the network topology for the pathfinding queries based on relationships referred to in `find_shortest_path_astar`. The purposeful integration of OSM data (road) and Google Places data (POIs), however, generated the relevance metric that serves to establish relevance links between POI data back to the graph of the road network. The produced integrated view facilitated discovery of different route types (Section 4.2), proving their fitness in terms of multicriteria navigation.

4.5.3.3 RQ3 (Evaluating Routes):

Within the constraints of this thesis (the backend operations), evaluation of urban exploration routes takes place primarily through computational metrics and qualitative visual analysis. The disparity in path lengths and number of nodes, as shown in Tables 4.2 to 4.4, support quantitative analyses. Map outputs (Figures 4.1 to 4.3) allow for a qualitative evaluation of the diversion and alignment of these paths to areas of interest, as indicated by the relevance map (Figure 3.10). Even though these successes suggest that the system is capable of producing a variety of potential 'engaging' action paths, a broader critique grounded in cultural engagement would require getting insights from user experience studies, which fall outside the scope of this thesis but are presented in the work of Hürü Algayeva.

5 SUMMARY AND FUTURE WORK

This chapter finds the thesis in conclusion: the research carried out is summarized and the main contributions stated for a more focused discussion under the research questions and sets forth areas of improvement that are currently under development.

5.1 Summary of Work

This thesis has addressed the provision of enhanced navigation experiences within the historically complex urban environment of the İçərişəhər. Standard navigation systems fail in capturing the cultural richness and specific requirements of such areas. Hence, this research aimed to work on the development of a backend infrastructure, data-processing pipeline, and deployment process of such a system that should be able to expend upon the multi-criteria routings; specifically, the shortest distance, cultural relevance, and integration with geocaching.

In order to suit all of the above, a method combining several stages was implemented. For these, we collected data about the road network from OpenStreetMap and Points of Interest from Google Places API in a narrowly stepped study area around İçərişəhər. Then with the use of Geopandas, Shapely, and Pyrosm Python libraries for data processing, the task of data cleaning, whereas handling geometrical complexities (from MultiLineStrings), measuring correct segment lengths, efficient collision-checks using spatial indexing, and network graph generation, was carried out effectively. The Google Places POIs were filtered for relevance using the gemini-2.5-pro LLM while formulating a "relevance" customized index refined in proximity of POIs to the road segments.

The hybrid databases were implemented with PostgreSQL together with PostGIS for storing detailed geometries and attributes and with Neo4j for storing the graph network topologies and supporting efficient graph traversals through the network. The pathfinding was then executed using the A* algorithm in the Graph Data Science library within the Neo4j setup with relevance defined as a cultural cost property and length as distance cost property to generate "Shortest" and "Cultural" paths. A "Geocaching" path included, where the main segments of the two shortest paths are combined for a chosen nearby geocache. The development of a RESTful backend API using FastAPI orchestrating the

database queries, implementing the pathfinding logic, and marshalling to routing service completed the overall deployment to production. Their entire system (backend, PostGIS, Neo4j) implemented with default configurations within Docker and Docker Compose for portability and ease of deployment.

The results have proven the applicability of this methodology successfully. The data-processing pipeline provided clean network data (12,414 nodes, 14,785 edges in the main component) and a set of POIs (5,689) filtered by relevance to this work. The system successfully produced visually clear and numerically diversified routes for the three modes (Shortest, Cultural, Geocaching) between various points in the study area. Observational performance tests concluded with results of adequate responsiveness on hands on practical usage (approx. 200ms end-to-end API response time), with containerization very powerful for reliability in denture and performance.

5.2 Contributions

This research contributed to helping distinct urban navigation systems on a couple of levels, particularly within the care of cultural heritage exploration. These give a solid starting point towards developing better navigation systems keyed to the old city environment of İçərişəhər, establishing methodologies and functions.

1. **Tailored Geospatial Data Processing Pipeline**: A data processing pipeline was designed and implemented for integrating and cleaning OSM roads and Google Places POIs for the context of İçərişəhər. This included the robust handling of inconsistencies in the OSM geometry (i.e., MultiLineString processing via `explode()`), calculation of exact geodesic length, efficient intersection matching through spatial indexing (`geopandas.overlay()`), and network graph cleaning.
2. **Novel Application of LLM for POI Filtering**: Utilizing a massive language model (gemini-2.5-pro), a study demonstrated the utility of context-aware, automatic filtering of a large POI dataset based on a tourism and urban-exploration context with very high precision within the cultural and linguistic contexts indicative of Baku.
3. **Custom Relevance Metric for Cultural Pathfinding**: Quantitative POI-proximal "relevance" metric implementation was performed, incorporating transformations (offsetting, scaling, clipping, scaling, and logarithmic scaling) and network-based smoothing of the "interestingness" of road segments for cultural exploration.
4. **Implementation of a Hybrid Spatial-Graph Database System**: A real scenario has been tackled in implementing a hybrid database architecture supporting PostGIS storage and querying for the spatial features, as well as network topology and graph-based pathfinding over Neo4j as many other complex geography-based network data sets are sure to be managed in the future.
5. **Multi-Criteria A* Pathfinding Implementation**: The emphasis is laid on real-time, functional pathfinding; for instance, our implementational output of the Neo4j GDS-based A* algorithm provided opportunities to point to multiple trustable features for a single pathfinding task, meaning that these multi-criteria would be the four alternatives (approximate shortest paths) for defining needed route: standard shortest path (by length), cultural path (by relevance), and geocaching path (including a detour).
6. **Reproducible Containerized Backend System**: We built an entirely containerized FastAPI backend system (Docker and Docker Compose) packed with the application incorporating PostGIS, Neo4j (still with GDS and optionally APOC), plus all dependencies to prepare firm deployment and a possibility for reproducibility.

5.3 Research Questions Revisited

The research conducted and the results obtained provide answers to the research questions initially posed in Section 1.2:

5.3.1 RQ1

How can the A* search algorithm be effectively extended to incorporate qualitative criteria, such as historical importance or scenic value represented by a derived relevance metric, for navigation within İçərişəhər?

This question was addressed through the development and application of the "relevance" metric (Section 3.3.5). This metric, derived from filtered POI data and refined through transformations and network smoothing, served as the qualitative criterion representing cultural significance or "interestingness". The A* algorithm, implemented via Neo4j GDS (Section 3.5), was effectively extended by configuring it to use this relevance score as its cost property instead of the standard length. The results (Section 4.2) demonstrated that this approach successfully generated "Cultural Paths" that were distinct from the shortest paths, favoring routes through areas with higher calculated relevance. The data processing pipeline ensured the necessary graph structure and attributes were available, and observational performance analysis (Section 4.3) indicated the approach was computationally efficient. The use of a configurable cost property provides a foundation adaptable to future refinements or alternative qualitative metrics.

5.3.2 RQ2

What approaches achieve the most effective form of data integration for creating an integrated and accurate view of İçərişəhər's urban terrain out of various geospatial data sets fit for a purpose in navigation?

This research demonstrated the effectiveness of a hybrid database approach combined with a tailored data processing pipeline. The pipeline integrated data from OSM and Google Places, addressing data quality issues through cleaning, geometric processing (including intersection finding using spatial indexes), and LLM-based filtering (Sections 3.3.1-3.3.4). The hybrid architecture (Section 3.4) leveraged PostGIS for storing accurate spatial geometries and attributes, enabling efficient spatial queries (like finding nearest nodes), while Neo4j stored the network topology optimized for pathfinding algorithms. This integrated view, populated through the processing pipeline, proved fit for the purpose of multi-criteria navigation, as evidenced by the successful generation of distinct path types (Section 4.2).

5.3.3 RQ3

How can the effectiveness of urban exploration routes be evaluated through both computational metrics and user experience measures to ensure meaningful cultural engagement in historical districts?

Within the scope of this backend-focused thesis, the effectiveness of the generated routes was primarily evaluated using computational metrics (path length, node count) and qualitative visual analysis (Section 4.2). The results showed measurable differences between the path types, indicating the system's ability to produce varied routes based on the optimization criteria. The visual comparison confirmed that 'Cultural Paths' often traversed different areas than 'Shortest Paths'. However, assessing whether these routes ensure meaningful cultural engagement requires user experience measures (e.g., user studies, surveys, feedback on perceived route quality and enjoyability), which were outside the scope of this work. This research provides the technical foundation and initial computational evaluation, and further validation through user-centered studies was conducted in the work of the Huru Algayeva, showing positive results.

5.4 Limitations

This research led to a successful back-end system, but there are limitations that should be acknowledged:

1. **Data Quality Dependency**: The quality of the paths generated depends squarely on the quality of the input data sources, i.e. OSM and Google Places API. Despite the cleaning software,

inaccuracies may remain with OSM geometries, tags, or length information. On a similar note, Google Places could have gaps or contain inaccuracies.

2. **API and Economic Constraints**: The use of Google Places API for POI data causes a restriction due to the network restrictions on the free usage tier. Depending on the extent of the system to cover greater areas or involve more frequent updates of POIs, the cost could soar to a put trade-off; alternative POI data source or strategies could necessitate considering.
3. **Subjectivity of Relevance Metrics**: Note that the relevance metric used in the cultural path exploration method is based on a number of specific heuristics (proximity of POIs, transforms, and smoothing). Though designed to discern "interestingness," it is subjective and simplified. Different methods, weights, or inputs (e.g., types of buildings, or historical importance tags) can produce very different cultural paths.
4. **Simplified Pathfinding Objectives**: The form of cultural path-finding presented with a straightforward, combined cost metric (inverted relevance) for the application of the A-star algorithm makes it unattainable for true multi-objective search (in contrast to Pareto-optimal paths). Even the rather simplistic geocaching path-finder loses track of coming up with the absolute shortest path in possible proximity of a cache.
5. **Static Data Representation**: It is the current moment of a core urban-base data structure consisting of a captured time snapshot of the urban environment. There is no provision here for the auto-capturing process for any changes in the world, such as in relation to the closed displacement of temporary roads or construction, variations in POI status, or other possibly dynamic events.
6. **Performance Evaluation Scope**: Performance evaluation was performed by using times of observation of the development environment. Therefore, the absence of validated benchmarking scenarios for production mode under real-world loads calls for caution with this benchmark's performance evaluations.

5.5 Future Work

The system developed in this thesis serves as a solid and secure basis, but there are several possibilities for future enhancement or research that can be built on its weaknesses and limitations. With such improvements, the İçərişəhər urban exploration navigation system beyond the limitations will become a step closer to being a comprehensive and interactive tool for cultural heritage discovery

5.5.1 Data Enrichment and Integration:

1. **Integrating Building Data**: The integration of OSM building footprints, already available, could be taken up future lines of inquiry. This could mean linking the POIs to specific buildings. Such measures could comprise some new metrics along these lines of building density, building age, or building style into the calculation of routes.
2. **Leverage More POI Properties**: Expanding the metric of relevance or pathfinding constraints with further detailed properties about POIs could raise questions. For example, opening hours, more specific category beyond the primary types or tag elements for historical importance that might have come from the OSM or any other data sources, depending upon their availability.
3. Alternative POI Sources: Check the alternative POI data sources, some of which might be open (e.g., visit Wikidata, national heritage register) to weaken dependence on Google Places API and eventually improve coverage and richness of attributes.
4. **Historical Map Integration**: Research how historical maps of İçərişəhər may be georeferenced and incorporated to give a sense of time to influence pathfinding according to historical routes.

5.5.2 Relevance Metric and Pathfinding Algorithms:

1. **Refine Relevance Metric**: Try alternative ways of computing what makes an asset more relevant. For example, by using distance decay rather than simply joining to the nearest road, one might

take into consideration the popularity of the POI or include ratings, some also trying to use machine learning to learn weights of relevance.
2. User Preference: Allow customization of the cultural pathfinding; e.g., a user may prefer the weightage of POI classes i.e., preferring museums over restaurants, or he might specify the characteristics of the route he/she prefers: avoiding hills, preferring pedestrian-only streets.
3. **Multi-Objective Optimization**: Implement a variety of pathfinding algorithms capable of finding a range of Pareto-optimal solutions which will simultaneously balance distance and with relevance (or any other criteria), hence producing a set of compromises to select from instead of selecting a "cultural path".
4. **Improving Geocaching Logic**: Improve the geocaching path algorithm for an optimized route in a way that it starts considering the caches present in the vicinity of the shortest cultural path instead of reckoning only the one pin at a time, or build on the constraints in the pathfinding search for a cache visit, if the algorithm supports this.

5.5.3 Graph Model Enhancements:

1. **Richer Graph Schema**: Consider adopting the richer graph schema models, whereby POIs or significant buildings are represented as separate nodes connected to the network of roads. This aids in performing more complex queries of relationships and potentially more nuanced pathfinding.
2. **Typed Relationships**: Different relationship types (such as `:PEDESTRIAN_ACCESS`, `:SCENIC_ROUTE_SEGMENT`) are suggested for embedding more semantic information on network connections.

5.5.4 System and Technical Improvements:

1. **Real-Time Data**: Study the inclusion of real-time data sources into the system; these sources might include temporary road signages, event information, or even dynamic queue-density information to check up on route validation and supplement user experience.
2. **Performance Benchmarking and Optimization**: Carry out some dedicated performance testing under simulated loads to fully identify bottlenecks restricting the system performance. Work on optimization tips like storing previously requested queries or pre-computation of marked GPS locations.
3. **Data Update Strategy**: Design the strategy and pipeline for real-time updating of network data and POIs related to the system.

References

Ai, T. J., & Kachitvichyanukul, V. (2009). A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research, 36(5), 1693–1702.

Cook, S. (2018). An Approach to Pathfinding for Real-World Situations (Doctoral dissertation, University of Leeds).

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.

Foad, D., Ghifari, A., Kusuma, M. B., Hanafiah, N., & Gunawan, E. (2021). A Systematic Literature Review of A* Pathfinding. Procedia Computer Science, 179, 507–514.

Gaede, V., & Günther, O. (1998). Multidimensional Access Methods. ACM Computing Surveys (CSUR), 30(2), 170–231. https://doi.org/10.1145/280277.280279

Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, 47–57. https://doi.org/10.1145/602259.602266

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.

Hunger, Michael. (2016). "From Relational to Graph: A Developer's Guide." DZone Refcardz. (Note: Text cited as 2017, but list entry is 2016)

Kim, J., Hong, S., Jeong, S., Park, S., & Yu, K. (2024). SGIR-tree: Integrating r-tree spatial indexing as subgraphs in graph database management systems. ISPRS Int. J. Geo-Inf., 13(2), 346.

Kreller, I., & Ludwig, B. (2021). User Preferences and the Shortest Path. KI-Künstliche Intelligenz, 35(1), 85-93.

Robinson, P. (2015). Conceptualizing Urban Exploration as Beyond Tourism and as Anti-Tourism. Advances in Hospitality and Tourism Research (AHTR), 3(2), 141-164.

Vyawahare, H.R., Karde, P.P., Thakare, V.M. (2018). A Hybrid Database Approach Using Graph and Relational Database. In 2018 IEEE International Conference on Research in Intelligent and Computing in Engineering (RICE). (Note: Text cited as Thakare and Vyawahare, 2018)

Vukmirović, S., Čapko, Z., & Babić, A. (2019). The exhaustive search algorithm in the transport network optimization on the example of urban agglomeration Rijeka. In MIPRO 2019 - Proceedings of the 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (pp. 1138–1143).