Reducing Schema Dependency in Natural Language Interfaces for Infrastructure Digital Twins through Automated Reader and Parser Generation

Peihang Luo[1], Linjun Lu[1], Ya Wen[1], Erika Parn[2], Lavindra de Silva[1], Borja García de Soto[2], and Ioannis Brilakis[1]

[1]Department of Engineering, University of Cambridge, United Kingdom
[2]S.M.A.R.T. Construction Research Group, Division of Engineering, New York University Abu Dhabi (NYUAD), Experimental Research Building, Saadiyat Island, P.O. Box 129188, Abu Dhabi, United Arab Emirates
pl452@cam.ac.uk, ll718@cam.ac.uk, yw710@cam.ac.uk, eap9920@nyu.edu, lpd25@cam.ac.uk, garcia.de.soto@nyu.edu, ib340@cam.ac.uk

Abstract –

Natural language interfaces (NLIs) have been explored as a way to lower the entry barrier to interacting with infrastructure digital twins (DTs) by reducing the need for extensive prior training. However, most existing NLIs rely on manually configured parsing logic based on fixed and predefined data schemas, making them difficult to scale across different DT deployments. In practice, infrastructure DTs often include datasets with varying schemas, even when using the same file format, resulting in significant manual effort to adapt interfaces to new datasets. This paper presents an early-stage exploration of an approach to reducing schema dependency in NLIs for infrastructure DTs through automated reader and parser generation. The proposed approach inspects the input dataset, infers key dataset characteristics, and generates dataset-specific readers and parsers on demand to support basic natural language queries. Generated components are reusable, reducing repeated configuration effort for subsequent interactions with the same dataset. A proof-of-concept prototype is implemented and demonstrated using a road defect dataset. The results demonstrate the feasibility of on-demand reader and parser generation for schema-agnostic natural language interaction, highlighting its potential to improve the scalability of NLIs for infrastructure DTs.

Keywords –

Natural Language Interface (NLI); Natural Language Processing (NLP); Large Language Model (LLM); Digital Twin (DT); Human-Computer Interaction (HCI); Operations and Maintenance (O&M); Infrastructure Management

1 Introduction

Infrastructure systems are critical to the normal functioning of our society. However, the effective management of these systems requires timely access to relevant and accurate information to support condition inspection, performance monitoring, and informed intervention decisions. In practice, a substantial proportion of infrastructure lifecycle costs, typically in the range of 65–80%, is incurred during the use phase, which covers the costs of operations and maintenance (O&M) activities [1]. Limited or inefficient access to asset condition and performance information can lead to delayed interventions, suboptimal decision-making, duplicated work, and increased reliance on corrective or reactive maintenance. Such approaches are significantly more costly than timely and data-informed preventive maintenance, as deferred minor repairs tend to escalate into major interventions at multiple times the original cost [2]. In addition to higher lifecycle costs, inadequate information availability in the O&M phase is also linked to increased safety risks, excessive material and carbon waste, and disruptions to essential services [3]. These challenges directly affect infrastructure productivity, the ability to meet sustainability targets, and the reliability of essential services, highlighting the importance of improving how operational data are accessed and used in infrastructure management.

To support O&M activities, infrastructure asset owners and operators have increasingly adopted digital twin (DT) systems to integrate heterogeneous datasets, including geometric models, geospatial information, sensor data, and inspection records, into unified digital representations of physical assets [4]. These systems are typically accessed through dashboards, graphical user

interfaces, or scripting-based tools to support monitoring, analysis, and decision-making across the asset lifecycle. While such data-rich systems offer significant potential, their value is not always fully realized in practice, as access to information often remains uneven across stakeholders, particularly in the presence of skills shortages and increasing data complexity [5]. In response, natural language interfaces (NLIs) have recently been explored as a way to lower entry barriers, reduce training requirements, and improve usability by allowing users to query DT data using natural language [6].

However, NLIs are typically costly and time-consuming to configure and maintain. Most existing approaches rely on stable data schemas and manually configured, dataset-specific logic to process user queries. This limits the reusability of NLIs across different DT deployments. As a result, NLIs often require repeated manual adaptation for each dataset and use case, limiting their scalability and contributing to high configuration and maintenance effort. This paper explores an early-stage approach in which dataset-specific readers, schema profiles, and parsers are generated automatically based on inspection of the dataset. Rather than predefining schema links and manually configuring query processing logics, the proposed method infers dataset characteristics and generates components on demand to support basic natural language queries. Generated components are treated as reusable artefacts, allowing subsequent queries on the same dataset to be handled without unnecessary repeated generations. The key contributions of this paper include: (i) an early-stage approach to automated, dataset-specific reader and parser generation for NLIs, based on explicit separation of data access, schema understanding, and query-specific processing, and (ii) a proof-of-concept prototype demonstrating the feasibility of this approach using a road defect dataset.

2 Background and Related Work

2.1 Data Structures and Schemas in Infrastructure Digital Twins

Infrastructure DTs are typically constructed by integrating multiple data sources that capture different aspects of a physical asset and its operation. Common data representations used in infrastructure DTs include IFC-based building information modelling (BIM) data [7], geospatial datasets (such as shapefiles and geo-databases) [8], tabular and relational databases for asset inventories and inspection records [9], and time-series data from sensors and monitoring systems [10]. Together, these datasets provide complementary views of asset geometry, location, condition, and performance.

In practice, the structure and schema of DT data vary widely across infrastructure domains, asset types, and organizational contexts. Different DT implementations often adopt different schema designs to reflect domain-specific requirements, data availability, and existing information management practices. Even when the same data format is used, such as a shapefile or a tabular dataset, the underlying semantics, including what entities are represented, how attributes are defined, and how relationships are encoded, can differ substantially depending on the intended purpose of the dataset.

As a result, infrastructure DTs rarely conform to a single, universal schema. Instead, schema variability is expected and necessary for real-world deployments, allowing DTs to accommodate diverse asset types, operational needs, and organizational constraints. This variability reflects sensible engineering choices, but it has important implications for how data can be accessed and interacted with across different DT implementations.

2.2 Natural Language Interfaces for Infrastructure Digital Twins

To improve the usability of infrastructure DTs, recent studies have explored NLIs as a means of interacting with DT data. Prior work has demonstrated the use of NLIs to query IFC-based BIM data [11], as well as to access geospatial infrastructure databases that store asset inventories, inspection records, and condition data [10]. In these systems, users can express their needs using natural language rather than using specialized query languages or software-specific workflows.

Across different application contexts, NLIs have the potential to reduce training effort and improve usability by allowing non-expert users to interact with the required information by directly expressing their intents in natural language. Reported benefits include faster access to data, improved task performance, and broader participation of stakeholders who may not be familiar with the interface or underlying data structures, demonstrating that such interfaces can lower entry barriers and support more intuitive interaction with complex DT datasets [6].

From an implementation perspective, most existing NLIs rely on explicit knowledge of the underlying data structure to process user queries. This typically involves manually specified mappings or data access logic that connect user queries to underlying datasets. While these pre-configurations enable reliable query execution within a given DT, they also tightly couple the interface to the structure of the underlying dataset. Some recent LLM-based approaches have also explored automatic code or function generation for data access and query execution. However, these systems typically generate task-specific logic on demand in a one-step and single-use manner, without an explicit pipeline to guide generation or mechanisms to support reuse, which can lead to repeated regeneration of similar logic, higher computational cost, and reduced efficiency over repeated interactions.

2.3 Limitations, Research Gap, and Objective

The variability of data structures in infrastructure DTs and the design of existing NLIs highlight a mismatch between how DT data are organized and how NLIs are currently implemented. While infrastructure DTs adopt diverse data schemas by design, most existing NLIs are implemented in a schema-specific manner, assuming a known and stable data structure. As a result, NLI configurations developed for one DT rarely transfer directly to another, requiring repeated manual reconfiguration and limiting scalability in practice.

While prior work has focused primarily on improving interaction quality and usability within individual DTs using NLIs, little attention has been paid to reducing their schema dependency. The objective of this study is to investigate the feasibility of automatically generating dataset-specific readers and parsers through inspection of the underlying dataset, with the aim of reducing schema dependency and improving the scalability and reusability of NLIs across different DT implementations.

3 Proposed Solution

3.1 Scope of the Proposed Solution

This paper proposes an early-stage approach to reducing schema dependency in NLIs for infrastructure DTs by automatically generating dataset-specific readers and parsers on demand based on the underlying dataset and the user's natural language query. The proposed solution currently targets single-dataset interactions and aims to support single-step queries that involve basic operations such as summarization, simple filtering, and basic aggregation. The focus is mainly on demonstrating the feasibility of automated, on-demand reader and parser generation and the reuse of generated artefacts across multiple queries on the same dataset.

3.2 System Overview

Figure 1 provides an overview of the proposed system architecture to support schema-agnostic natural language interaction with infrastructure DTs. The system adopts a modular design in which dataset access, inspection, and query-specific processing are handled by different components coordinated by an LLM-based orchestrator.

The architecture explicitly separates three roles: data access, schema understanding, and query-specific logic. Readers handle format-specific access logic, the schema profiler summarizes dataset structure and attributes, and parsers encode query-specific processing logic.

This explicit separation of roles allows different components to be generated and reused at different levels, allowing the system to avoid repeated generation when possible. Readers are associated with data formats and

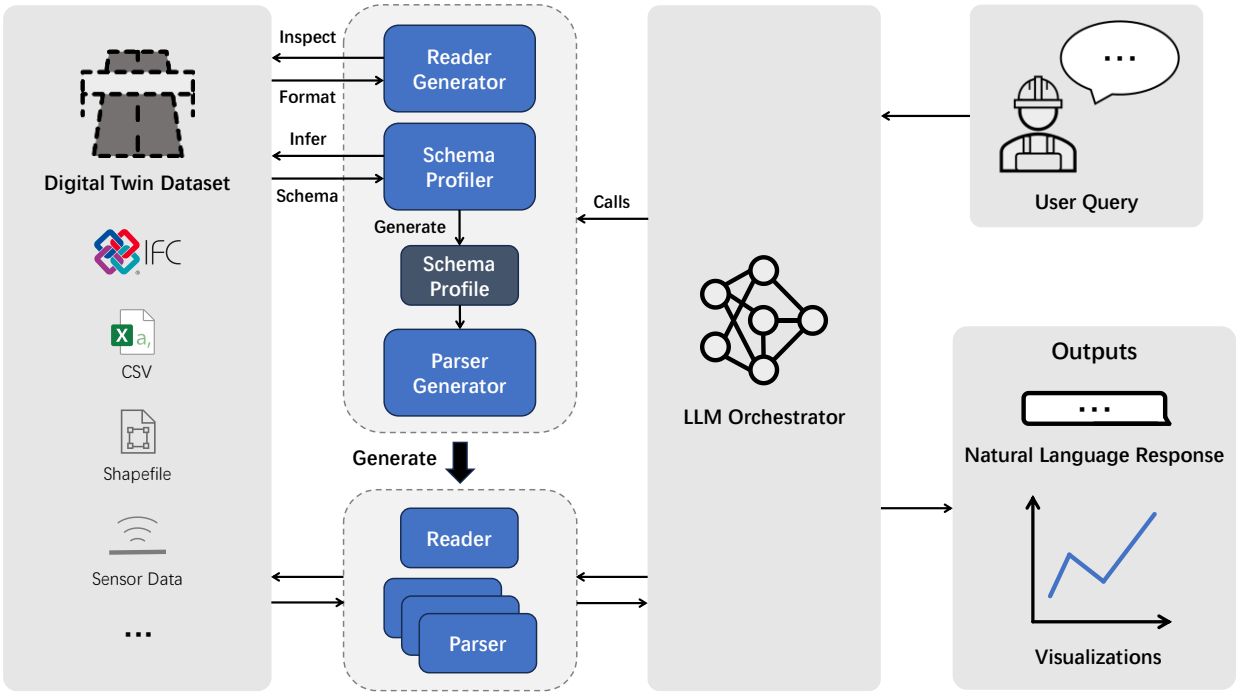

Figure 1. Proposed system architecture for automated, dataset-specific reader and parser generation to support schema-agnostic natural language interaction with infrastructure DTs. The LLM orchestrator coordinates three core components: a reader generator, a schema profiler, and a parser generator, which together enable the generation and reuse of dataset-specific components for query execution.

can be reused across files sharing the same format, schema profiles can be reused across queries on the same dataset, and parsers can be reused for queries sharing the same data processing logic. The automated generation of dataset-specific components enables the system to access and process data without relying on predefined logics. These generated components are then used by the LLM orchestrator to execute the query and produce the corresponding response to the user.

To better understand the workflow, consider the query "Count defects by severity." When this query is issued, the system first checks if any existing components can be reused. If no compatible components are found, a dataset-specific reader is generated. Using this reader, the schema profiler then inspects the dataset and produces a schema profile. Using this profile together with the user query, the system generates a query-specific parser that groups the records by severity and counts how many fall into each category. The parser calls the reader to access the data, executes the aggregation logic, and returns the results, which are then formatted into a natural-language response. For later queries on the same dataset, the reader and schema profile are reused, and a new parser is generated only if the task logic changes. The detailed pipeline is described in Section 3.3.

3.3 Key Steps in the Processing Pipeline

Figure 2 illustrates the pipeline executed when a user issues a query. Starting from the user query, the pipeline determines whether existing components can be reused and, where necessary, generates new components before executing the query and returning a response. The following subsections describe each step in more detail.

3.3.1 Inspection of Generated Components

The pipeline begins by inspecting existing components to determine whether they can be reused for the current query. This aims to reduce unnecessary regeneration and can potentially speed up subsequent interactions with the same dataset. To enable this, each generated reader and parser is accompanied by a structured component description that specifies its purpose, required inputs, and expected outputs.

When a new query is issued, the system inspects the available component descriptions to assess compatibility with the current dataset and query. If compatible components are identified, they are reused directly. Otherwise, the pipeline proceeds to generate new readers, schema profiles, or parsers in the subsequent steps.

3.3.2 Reader Generation

This step is triggered when no existing reader is found to be compatible with the dataset being queried. The process begins with an inspection of the queried dataset to identify its file format and basic access characteristics. This inspection extracts information required for data access, including the dataset type (e.g., tabular, geospatial, or time-series), the file format used, and any structural metadata (e.g., column headers in tabular data or attribute fields in geospatial files). The purpose of this inspection is not to infer full semantic meaning, but to determine how the data should be loaded and represented internally so that subsequent processing steps can operate independently of the original file format, isolating all format-specific handling from downstream processing.

Based on the extracted information, the system generates a dataset-specific reader. It encapsulates all logic required to load the dataset and expose the data through a consistent internal representation that can be consumed by downstream components. Alongside the generated reader, the system produces a description that explicitly documents the reader's functionality and interface. This description includes the supported data format and the structure of the internal representation exposed to downstream components, including the expected inputs and outputs. By making this information explicit, the system can reason about reader compatibility and reuse without inspecting the reader's internal code.

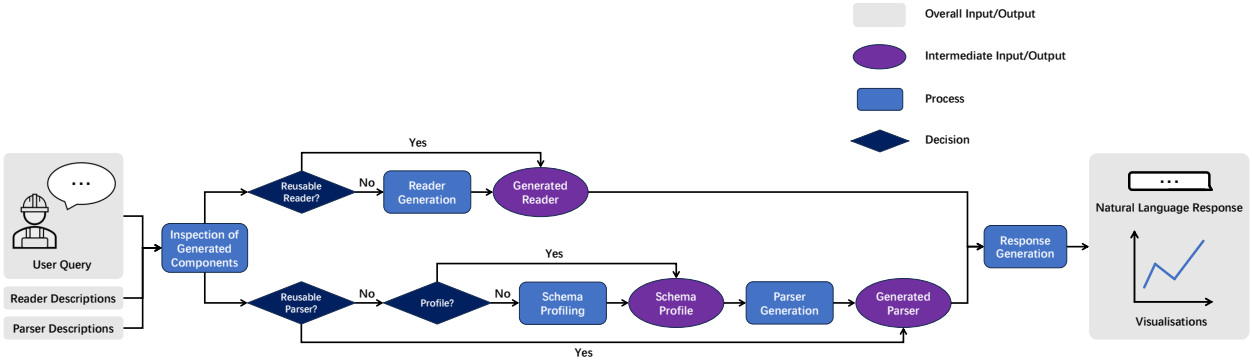

Figure 2. Proposed pipeline for schema-agnostic natural language interaction with infrastructure DTs. The pipeline starts from a user query, inspects existing component descriptions for reuse, and conditionally generates readers, schema profiles, and parsers before executing the query and returning a response.

Readers are designed to be independent of query-specific logic. As a result, a reader generated for a particular data format can be reused across multiple queries on the same dataset, and potentially across different datasets that share the same format. This contrasts with parsers, which encode query-specific processing logic and therefore have a more limited reuse scope. The separation of reader generation from parser generation is a key design choice that reduces redundant component generation.

3.3.3 Schema Profiling

Schema profiling is responsible for capturing the characteristics of a dataset in a compact and LLM-friendly form that supports reliable and efficient parser generation. In this way, schema profiling decouples structural understanding of the dataset from both data access and query-specific processing logics.

Using the reader's access interface, the system examines entities and attributes present in the dataset. The goal of this inspection is to identify the elements that might be referenced in queries and how they can be accessed from the dataset. For example, if the schema profile identifies a categorical field with a defined set of values and an implicit ordering, this information can be used by the parser generator to construct both aggregation logic and filtering conditions. The output of this step is a schema profile that summarizes the dataset structure in a compact, LLM-friendly format. The schema profile explicitly lists all entities, attributes, and their basic properties, and may include basic relational information where relevant. Importantly, the profile focuses only on the information required to interpret and construct query logic. This reduction in complexity helps the parser generator focus on the data processing logic instead of being distracted by the raw data and any low-level details. The schema profile can be generated once per dataset and reused across multiple queries.

3.3.4 Parser Generation

Parser generation is responsible for translating the user query into executable data processing logic. To generate a parser, the system provides the schema profile as a contextual input alongside the user's natural language query. The schema profile guides the interpretation of the query by explicitly specifying which entities, attributes, and basic relationships are available in the dataset. This allows the system to resolve references in the query and disambiguate terms. The parser generator then constructs executable processing logic that corresponds to the query intent. All data access operations are delegated to the reader. This separation allows the same reader to support multiple parsers with different task logics, and enables parsers to focus only on query semantics and processing behavior.

Alongside the generated parser, the system produces a parser description that explicitly documents the parser's functionality and interface. This description captures the task logic supported by the parser, the types of operations performed, and the expected inputs (if any) and outputs. By making this information explicit, the system can treat parsers as reusable artefacts rather than opaque code fragments, enabling the system to assess compatibility with new queries in an extensible manner. Because parsers encode query-specific logic, their reuse scope is more limited than that of readers and schema profiles. However, parsers can still be reused when subsequent queries share the same data processing logic.

3.3.5 Response Generation

In the final step of the pipeline, the generated reader and parser are called together to execute the user's query. They are executed within a controlled environment with basic error handling during loading and execution. The output produced by the parser is then post-processed to construct a response suitable for presentation to the user, and, where appropriate, may also generate simple tabular or visual representations to support interpretation.

Once the response has been generated, the interaction cycle is complete. All generated components remain available for subsequent queries on the same dataset. This allows the system to avoid unnecessary regeneration.

4 Research Methodology

4.1 Dataset

The proposed approach is demonstrated using a road defect dataset stored as a shapefile. The dataset is adapted from the CAMHighways dataset [8]. The dataset contains 21 defect records with five fields, including categorical attributes (defect type and synthetically populated severity level), numerical attributes (centroid coordinates), and geometries. It includes 12 defect types and 3 severity categories, reflecting variability in attribute values. Further details are shown in Table 2. The shapefile format is representative of infrastructure datasets commonly integrated into DT systems for road asset management.

4.2 Prototype Implementation

A proof-of-concept prototype is implemented in Python to demonstrate the proposed reader and parser generation pipeline. In the current implementation, **GPT-5-mini** is used via the OpenAI API due to its reliability in generating executable Python code and structured textual outputs, while also having a relatively low cost and short response time. However, the prototype is designed to work with other cloud-based models and

local models, allowing flexibility in deployment. Generated artefacts are stored as Python and JSON files so that they can be reused across multiple queries. The prototype is initialized without any prior knowledge of the schema or structure of the testing dataset. All components are generated on demand through inspection of the input dataset, without relying on any predefined schema mappings or dataset-specific configuration.

Reader generation begins with file-level inspection to identify the dataset format. The reader generator then prompts the LLM to produce a Python function that:

1. loads the dataset using an appropriate library (e.g., `geopandas.read_file()`),
2. returns a structured in-memory representation (e.g., a GeoDataFrame), and
3. exposes a consistent access interface for downstream processing.

Readers are stored as Python modules (e.g., `reader_<dataset_id>.py`) containing a callable function. This allows generated parsers to import and use readers directly. Alongside the reader, a reader description is generated in structured text form, capturing the supported file format and the returned data structure. The generation is guided by a structured prompt template that specifies the dataset path, detected file format, and required output interface, instructing the LLM to produce a Python function that loads the dataset and returns a structured representation. Similar structured prompts are used for schema profiling and parser generation.

Schema profiling is performed by executing the generated reader to access the dataset and inspecting the data programmatically. This includes attribute names, data types, and basic value statistics (e.g., ranges, missing values). The schema profiler then prompts the LLM to summarize the extracted information into a compact, machine-interpretable schema profile, which is stored as a JSON file (e.g., `profile_<dataset_id>.json`). This JSON profile is designed to be used directly as structured context during parser generation.

Parser generation takes the user's natural language query, the schema profile JSON, and the reader description as inputs. The LLM is then prompted to generate a Python script that:

1. imports the generated reader,
2. calls the reader to access the dataset,
3. applies query-specific processing logic (e.g., filtering, aggregation), and
4. returns structured results.

Parsers are also stored as Python files (e.g. `parser_<query_id>.py`). Each parser is accompanied by a parser description capturing the supported query intent, the operations performed, and the expected inputs (if any) and outputs.

4.3 Experimental Procedure

The proposed solution is evaluated through three demonstration queries designed to exercise different parts of the pipeline and observe component reuse:

1. "Summarize this dataset."
2. "Count defects by severity."
3. "Show defects with severity greater than or equal to major."

The first query focuses on dataset summarization, and is expected to trigger reader generation, schema profiling, and parser generation. The second query focuses on aggregation and counting, and is expected to reuse the reader and schema profile to generate a new parser. The third query focuses on conditional filtering, which is also expected to reuse the reader and schema profile to generate a different parser. Across these queries, reuse can be verified by observing that the reader and schema profile generated during Q1 are reused during Q2 and Q3, while only query-specific parsers are regenerated.

5 Discussion

5.1 Results & Findings Summary

Across the three demonstration queries (Q1–Q3), the system generated and used dataset-specific components as described in Section 3. For the first query (Q1), the pipeline generated (i) a reader for accessing the shapefile dataset, (ii) a schema profile summarizing all attributes and types, and (iii) a query-specific parser. For subsequent queries (Q2 & Q3), the previously generated reader and schema profile were reused, while new parsers were generated to implement different task logics. Table 1 summarizes which components were generated versus reused for each query, illustrating the intended reuse scope: readers and schema profiles can be reused across queries for the same dataset, whereas parsers are more query-specific.

Table 1. Generation versus reuse across queries

Query	Reader	Schema profile	Parser
1	Generated	Generated	Generated
2	Reused	Reused	Generated
3	Reused	Reused	Generated

The pipeline executed successfully for all three queries. For Q1, the system generated a schema profile that correctly identified all attributes, including defect type, severity, and centroid coordinates, together with geometry presence and coordinate reference system (CRS). An excerpt of the generated schema profile is shown in Table 2.

Table 2. Generated schema profile excerpt

Field	Type	Example value(s)
type	string	bleeding, crack_block, patch
severity	string	Minor, Major, Critical
centroid_x	float	601290.592
centroid_y	float	261755.202
geometry	geometry	(polygon)

For Q2, the generated parser performed a grouping and counting operation over the severity field and produced a severity distribution (shown in Table 3).

Table 3. Output for Q2: Defect counts by severity

Severity	Count
Major	9
Minor	8
Critical	5

For Q3, the generated parser applied a conditional filter, returned both a natural language response and a list of all matching defects (Table 4 shows the first 5 entries).

Table 4. Output excerpt for Q3: Filter severity ≥ Major

Type	Severity	Centroid X	Centroid Y
crack_alligator	Major	600616.860	261585.626
crack_block	Major	599663.538	261501.259
crack_corner	Major	598875.798	261998.663
crack_corner	Critical	600218.031	261552.017
crack_longitudinal	Critical	600056.473	261526.859

Table 5 shows the generation time of each component. Supporting these queries using conventional NLIs would require dataset-specific configuration, including but not limited to manual schema linking and implementation of query logics. In the proposed approach, these steps are performed automatically, reducing manual configuration efforts to set up NLIs for different DT datasets.

Table 5. Component generation time

Component	Generation time (s)
Reader	17.9
Schema profile	10.3
Parser (Q1)	55.5
Parser (Q2)	31.3
Parser (Q3)	21.3

In the demonstrated queries, the generated components successfully executed the required tasks under the tested conditions. This shows that the proposed approach is feasible for an infrastructure dataset stored in a common geospatial format, and that component generation is feasible within interactive timescales for the demonstrated queries. In addition, the reuse behavior observed in Q2 and Q3 supports the intended design rationale: while task-specific parsers might need to be regenerated across queries, readers and schema profiles can be reused for queries over the same dataset, avoiding repeated regeneration of format-handling components and dataset structure information, and enabling faster subsequent interactions. While the evaluation is done on a single test dataset, the results show that the system can adapt to an unseen dataset by automatically generating the required components from a blank initial state. This shows the potential of the approach to operate across different data formats without prior schema knowledge.

5.2 Contributions

This paper provides early evidence that automated, dataset-specific component generation can reduce schema dependency in NLIs for infrastructure DTs. The main contributions of this study are: (i) an early-stage approach to automated, dataset-specific reader and parser generation for natural language interaction with infrastructure DTs. The approach reduces schema dependency by explicitly separating data access, schema understanding, and query-specific processing logic, with generated artefacts documented via explicit descriptions to support reuse across queries; and (ii) a proof-of-concept prototype demonstrating the feasibility of the approach on a road defect shapefile dataset.

6 Conclusions

This paper presented an early-stage approach to reducing schema dependency in NLIs for infrastructure DTs through automated, dataset-specific reader and parser generation. Instead of relying on manually configured parsing logics, the proposed method inspects an input dataset and generates reusable readers and query-specific parsers on demand. A proof-of-concept prototype was implemented and demonstrated using a road defect dataset, showing that basic summarization, aggregation, and filtering queries can be supported while reusing dataset-level artefacts across queries.

Practically, this approach offers a more scalable and maintainable solution for NLIs in infrastructure DTs. By separating format-specific data access, schema profiling, and task-specific query logic, the proposed approach reduces the manual effort when configuring NLIs for different infrastructure DTs. This has the potential to lower the cost and time required to deploy NLIs for DTs across different infrastructure assets and organizations, supporting broader access to DTs for practitioners with

varying technical backgrounds. At a societal level, improving access to infrastructure data during the O&M phase can contribute to more timely interventions, better-informed decision-making, and more efficient use of resources, which are critical for maintaining safe, reliable, and sustainable infrastructure systems.

This study only presents an early-stage exploration. The prototype was only tested on a single dataset and supports only relatively simple queries. Further evaluation is required to more comprehensively assess its generalizability across a wider range of data formats and schemas. A further limitation is that the robustness and correctness of generated components have not been systematically evaluated. Specifically, their behavior under edge cases (e.g., missing values, unexpected data types, or variations in query phrasing) and validation of generated code remain to be addressed. Future work will aim to incorporate validation and recovery mechanisms and evaluate performance across a wider range of datasets and query conditions. Future work will also aim to extend the approach to more interaction patterns typical of large-scale infrastructure DTs. This could include multi-dataset interactions, more systematic management of generated artefacts, and support for more complex tasks.

7 Acknowledgments

The authors gratefully acknowledge EPSRC, Bentley Systems, and the University of Cambridge for funding this research through the EPSRC Centre for Doctoral Training in Future Infrastructure and Built Environment: Resilience in a Changing World (EPSRC grant reference number EP/S02302X/1). This research is supported by the Digital Roads Prosperity Partnership (DR). DR is supported by the EPSRC grant number [EP/V056441/1], Costain, National Highways, the University of Cambridge, Department for Transport and Didimi. This research was partially supported by the Center for Sand Hazards and Opportunities for Resilience, Energy, and Sustainability (SHORES), funded by Tamkeen under the NYUAD Research Institute Award CG013.

References

[1] United Nations Department of Economic and Social Affairs. *Managing Infrastructure Assets for Sustainable Development: A Handbook for Local and National Governments*. United Nations, New York, 2021. doi: 10.18356/9789210051880.

[2] W. R. de Sitter. Costs of service life optimization "The Law of Fives". In *CEB-RILEM Workshop on Durability of Concrete Structures*, pages 131–134, Copenhagen, Denmark, 1984. ISBN 87-87336-18-9.

[3] A. Louhghalam, M. Akbarian, and F.-J. Ulm. Carbon management of infrastructure performance: Integrated big data analytics and pavement-vehicle-interactions. *Journal of Cleaner Production*, 142: 956–964, 2017. doi: 10.1016/j.jclepro.2016.06.198.

[4] Q. Lu, A.K. Parlikad, P. Woodall, G. Don Ranasinghe, X. Xie, Z. Liang, E. Konstantinou, J. Heaton, and J. Schooling. Developing a Digital Twin at Building and City Levels: Case Study of West Cambridge Campus. *Journal of Management in Engineering*, 36(3): 05020004, 2020. doi: 10.1061/(ASCE)ME.1943-5479.0000763.

[5] M. Yin, V.K. Reja, R. Wei, I. Brilakis, B. Sheil, F. Perrotta, A. Marie d'Avigneau, and L. Lu. Exploring the value of digital twins for information management in highway asset maintenance. *Developments in the Built Environment*, 21: 100614, 2025. doi: 10.1016/j.dibe.2025.100614.

[6] P. Luo, X. Wen, Y. Jiang, E. Pärn, and I. Brilakis. ChatTwin: Bridging the usability gap to digital twin adoption in infrastructure operations and maintenance with a Natural Language Interface. *Automation in Construction*, 181: 106655, 2026. doi: 10.1016/j.autcon.2025.106655.

[7] F. Mok, P. Luo, W. Chen, and I. Brilakis. Enriching and maintaining road digital twins with condition information. In *Proceedings of the 30th International Workshop on Intelligent Computing in Engineering*, pages 338–347, London, UK, 2023. doi: 10.17863/CAM.96810.

[8] A. Marie d'Avigneau, L. Potseluyko, N.R. Anvo, H.M. Taha, V.K. Reja, D. Davletshina, P. Lam, L. de Silva, A. Al-Tabbaa, and I. Brilakis. CAMHighways: The Cambridge Highways dataset. *Advanced Engineering Informatics*, 64: 103036, 2025. doi: 10.1016/j.aei.2024.103036.

[9] P. Luo, E. Parn, S. Kookalani, and I. Brilakis. TailorAlert: Large Language Model–Based Personalized Alert Generation System for Road Infrastructure Management with Digital Twins. *Journal of Computing in Civil Engineering*, 40(2): 04025154, 2026. doi: 10.1061/JCCEE5.CPENG-7025.

[10] P. Luo, E. Parn, and I. Brilakis. ChatTwin: Enabling Natural Language Interactions with Infrastructure Digital Twins. In *Proceedings of the 2024 European Conference on Computing in Construction*, Chania, Crete, Greece, 2024. doi: 10.35490/EC3.2024.259.

[11] S. Iranmanesh, H. Saadany, and E. Vakaj. LLM-assisted Graph-RAG Information Extraction from IFC Data. In *Proceedings of the 2025 European Conference on Computing in Construction*, Porto, Portugal, 2025. doi: 10.35490/EC3.2025.366.