LLM-Driven Multi-Agent Inspection Planning via Semantically Enriched Knowledge Graphs for Non-Destructive Testing

Ghezal Ahmad Jan Zia[2,*,†], Andre Valdestilhas[1,*,†], Benjamin Moreno Torres[2,†] and Sabine Kruschwitz[2,†]

[1]Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
[2]Federal Institute For Materials Research and Testing (BAM), Germany

Abstract

This paper presents a multi-agent system for inspection planning in Non-Destructive Testing (NDT), extending our prior work on Knowledge Graph (KG) generation. We integrate a KG with agents—PlannerAgent, ToolSelectorAgent, and ForecasterAgent—that collaborate via LangChain to create context-aware inspection plans, select tools, and forecast timelines. Key contributions include enriching the KG with new semantic relationships for infrastructure types, materials, and defects, as well as developing a real-time Streamlit-based UI for visualizing plans and reasoning subgraphs. Case studies demonstrate the system's improved relevance, explainability, and precision in defect-to-NDT mapping. We also address interoperability with Linked Data standards and validate plan consistency using SHACL constraints. Overall, this work showcases a novel integration of semantic web technology and LLM-based reasoning for infrastructure maintenance.

Keywords

Non Destructive Testing, Materials Science, Knowledge Graph, Large Language Models

1. Introduction

Non-Destructive Testing (NDT) refers to a range of inspection methods used to detect defects and assess material integrity without causing damage. It is crucial for ensuring the safety and longevity of infrastructure across domains like civil engineering, aerospace, and manufacturing. In civil infrastructure (bridges, tunnels, industrial plants), materials such as concrete and steel can undergo various deterioration mechanisms (e.g. corrosion, fatigue) that lead to observable defects (cracks, spalling, discoloration). Selecting the proper NDT method to detect a given defect is non-trivial because of the diversity of materials, defect manifestations, and test parameters. For example, moisture-driven cracking in a concrete bridge might be detectable via ultrasonic testing or infrared thermography depending on conditions. The challenge is compounded by the need to plan when and how to inspect complex structures under evolving conditions.

In recent years, knowledge graphs (KGs) have emerged as a promising approach to organize and retrieve complex multi-modal knowledge in domains like materials science and infrastructure[1]. KGs represent information as a network of entities (nodes) and relationships (edges), which can facilitate knowledge discovery and decision support. In the context of NDT, a KG can explicitly link materials, deterioration mechanisms, defects, and applicable NDT methods to assist engineers in choosing appropriate inspection techniques. In our previous work [2], we leveraged a large language

SeMatS 2025: The 2nd International Workshop on Semantic Materials Science co-located with the 24th International Semantic Web Conference (ISWC 2025), November 2nd, 2025, Nara, Japan
*Corresponding author.
†These authors contributed equally.
a.valdestilhas@vu.nl (G. A. J. Zia); a.valdestilhas@vu.nl (A. Valdestilhas); Manfred.Jeusfeld@acm.org (B. M. Torres); Manfred.Jeusfeld@acm.org (S. Kruschwitz)
https://github.com/firmao (G. A. J. Zia); https://github.com/firmao (A. Valdestilhas); http://conceptbase.sourceforge.net/mjf/ (B. M. Torres); http://conceptbase.sourceforge.net/mjf/ (S. Kruschwitz)
0000-0002-0079-2533 (G. A. J. Zia); 0000-0002-0079-2533 (A. Valdestilhas); 0000-0002-9421-8566 (B. M. Torres); 0000-0002-9421-8566 (S. Kruschwitz)

model (LLM) to automatically extract such relationships from literature and populate a Neo4j-based NDT knowledge graph. That system introduced material-specific agents (for concrete, wood, steel, and brick) which used curated glossaries and static rules to guide the LLM's extraction of triples (e.g., Concrete – HAS-DETERIORATION-MECHANISM ⟶ Corrosion – CAUSES-PHYSICAL-CHANGE ⟶ Cracking – DETECTED-BY ⟶ Ultrasonic Testing). The resulting KG successfully captured intricate relations, like the applicability of specific NDT methods to materials under certain degradation conditions [2]. However, the 2024 approach was limited to off-line knowledge extraction and lacked a dynamic mechanism for using this knowledge in inspection planning.

In this paper, we address that gap by introducing an LLM-driven multi-agent system that utilizes the NDT knowledge graph for automated inspection planning. Our approach is influenced by the advent of LLMs that can act as autonomous agents reasoning and taking actions (e.g., via the LangChain framework). We design a set of specialized agents – a PlannerAgent to propose inspection plans, a ToolSelectorAgent to identify optimal NDT tools for detected defects, and a ForecasterAgent to predict timelines and future inspection needs. Each agent is powered by a local LLM (such as the 7B-parameter Mistral model, chosen for its strong performance relative to size [1] and ability to run on-premises via Ollama) and has access to the semantic knowledge graph and external tools (e.g., databases or analysis libraries). By dividing the complex task into focused agents, we leverage the benefit that specialized LLM agents can perform better on focused sub-tasks and provide more interpretable outcomes [2].

Our semantic knowledge graph has been extended and enriched to support this multi-agent reasoning. We incorporate new entity types for Infrastructure (e.g., Bridge, Tunnel, Plant) and InspectionPlan, and we introduce named relationships to represent inspection planning knowledge. For instance, an InspectionPlan node may have relationships like RECOMMENDS (to an NDT Method), ADDRESSES (a particular Defect), TARGETS (an Infrastructure or component), and JUSTIFIED-BY (linking to evidence nodes or subgraphs in the KG that explain the recommendation). By treating the inspection plan itself as a first-class entity in the KG, we can log the decisions of each agent along with their justifications in a structured, queryable form. This design is aligned with Semantic Web principles, enabling us to export the knowledge graph to OWL/RDF and even apply SHACL shapes for validating plan structures.

The contributions of this paper are summarized as follows:

- **Multi-Agent Inspection Planning**: We develop a novel multi-agent architecture for infrastructure inspection planning that integrates LLM reasoning with a domain knowledge graph. The agents (Planner, ToolSelector, Forecaster) collaborate to generate context-aware inspection plans (including scheduling) that are informed by both learned knowledge (from LLMs) and symbolic domain knowledge (from the KG).
- **Semantically Enriched NDT Knowledge Graph**: We extend our previous NDT KG to include infrastructure context and inspection planning concepts. The KG uses meaningful relationship types (e.g., HAS-DEFECT, DETECTED-BY, RECOMMENDS) to enable explainable traversals. We demonstrate how the KG can be exported as an OWL ontology and linked with existing ontologies/standards for NDT and infrastructure (ensuring interoperability with Linked Data ecosystems).
- **Explainable Decision Support**: Our system provides explainability by design. Each agent's decisions (e.g., why a certain NDT tool was selected) are justified by referencing relevant nodes and relations in the KG, which can be visualized as a subgraph. A custom Web UI displays these reasoning subgraphs (using PyVis network graphs) alongside the recommendations, allowing engineers to trace the chain of reasoning for each suggestion. This addresses the typical black-box nature of LLM decisions by grounding them in a shared knowledge graph.
- **Forecasting and Scheduling**: We integrate a ForecasterAgent that uses semantic knowledge (e.g., defect progression rates, maintenance rules) to produce a timeline for inspections. The agent's output is presented as an interactive Gantt chart, giving a clear view of the proposed

[1] https://klu.ai/blog/open-source-llm-models

[2] https://blog.langchain.dev/langgraph-multi-agent-workflows/

inspection schedule (e.g., immediate tests vs. follow-up inspections after certain intervals). This adds a temporal dimension to the KG, linking planned actions with time, and provides a preview of how an inspection program might unfold.

- **Case Study Evaluation**: We present a case study focused on concrete infrastructure (bridges and tunnels) suffering common defects (like cracking under high humidity and chloride exposure). We simulate inspection planning an example scenario and evaluate the system's performance in terms of recommendation accuracy, traceability of reasoning, and user feedback. Although no formal benchmark exists for this novel task, we report qualitative results and feedback from domain experts (in infrastructure NDT) to assess the system's usefulness and reliability.

The rest of the paper is organized as follows. Section 2 describes the methodology, including the multi-agent system design, knowledge graph schema, LLM integration, and user interface. Section 3 shows how key elements of our knowledge graph and ontology are exported in OWL/RDF, illustrating interoperability and constraint checking. Section 4 details the experimental setup and case study scenario used to evaluate our approach. Section 5 presents results and observations from these experiments. Section 6 discusses related work, comparing our system to existing approaches in semantic web, multi-agent systems, and infrastructure maintenance. Section 7 provides discussion on the implications, limitations, and possible extensions of this work. Finally, Section 8 concludes the paper and outlines future research directions.

2. Related Work

Our research sits at the intersection of semantic web technologies, large language models, multi-agent systems, and infrastructure maintenance. Here we discuss relevant state-of-the-art and how our approach compares.

Recent years have seen a surge in using LLMs to assist with knowledge graph tasks. LLMs have been applied to extract triples from text, populate KGs, and even to generate ontologies from natural language [3]. Our prior work [2] is one such example, leveraging GPT-based extraction for NDT knowledge. Surveys like [4] and workshops like LLM+KG [5] highlight a key theme: LLMs are powerful but can hallucinate or lack consistency, whereas KGs provide structured, factual grounding. Our current system exemplifies this symbiosis—LLM agents perform reasoning and language-based planning, but always check or log their facts against a curated KG. This falls under what some literature calls "neuro-symbolic approaches", combining neural language models with symbolic knowledge [3]. Unlike pure retrieval-augmented generation, which might retrieve text passages, we specifically retrieve and use knowledge graph relations, ensuring the LLM's decisions align with an ontology.

The idea of multiple agents cooperating is a longstanding topic in AI (e.g., BDI agents, blackboard systems). With LLMs, new frameworks have emerged to orchestrate multiple language model instances on different tasks [3]. For example, HuggingGPT [6] is a framework where ChatGPT delegates tasks to specialist models (though not exactly peers) – it demonstrates how a main LLM can coordinate multiple tools or models. Similarly, the LangChain community has introduced patterns for multi-agent collaboration, such as in the LangGraph system. Our approach aligns with these by giving each agent a clear role and connecting them via a shared state (our knowledge graph acts as a kind of shared memory or blackboard). A key difference is the explicit semantic structure we use; many LLM multi-agent demos share a text conversation as the medium of communication, whereas we use the KG to mediate communication (agents post their decisions to the KG). This could be seen as a step towards "cognitive architectures" for LLMs, where state is structured and persistent. Our ForecasterAgent and PlannerAgent dynamic is analogous to a planner-scheduler interplay in multi-agent planning literature, but here both are realized with LLM reasoning enhanced by data.

The use of ontologies and KGs in civil infrastructure and NDT is an active area. The SODIA ontology and related works by [7] aim to model non-destructive evaluation processes for bridges. They focus on

[3] https://blog.langchain.dev/langgraph-multi-agent-workflows/

linking inspection data with BIM (Building Information Models) for bridges, addressing interoperability issues. Our work shares the goal of structured representation but differs in scope: we incorporate LLM-driven decision-making on top of the knowledge base. The SODIA approach is more deterministic, manually encoding expert knowledge in an ontology for planning inspections, whereas we allow an AI agent to propose plans (bounded by an ontology). Potentially, our system could integrate SODIA as part of its KG (e.g., use classes from SODIA, or output plans in SODIA format), combining the strengths of both. Additionally, ontologies like SODIA often emphasize integration with BIM and sensor data, which is complementary to our focus on literature-derived knowledge and planning logic.

Another relevant thread is NDE 4.0, a movement to modernize NDT with digital technologies [8]. NDE 4.0 highlights connecting NDT results with digital twins, using data for predictive maintenance. Our system contributes to NDE 4.0 by providing a semantic layer that could interface with a digital twin (for instance, the KG could be linked to a digital model of the bridge, marking where defects are and what inspections are planned). Some prior works have looked at optimizing inspection schedules using traditional AI (like genetic algorithms or Bayesian methods), but they often don't incorporate semantic knowledge or explainability. Our approach offers a new angle: using reasoning over a knowledge graph to derive plans that are explainable and adaptable.

There is emerging research on using LLMs in engineering domains for tasks like code generation, design assistance, or troubleshooting. For instance, LLMs have been used to parse technical documents and answer questions in domains such as aerospace and automotive maintenance. Our use of a local LLM (Mistral) with domain-specific prompting is in line with efforts to fine-tune or prompt-engineer LLMs for specialized tasks [4]. One challenge often cited is the need to handle domain-specific terminology and not have the LLM stray into incorrect reasoning. By continuously referencing a knowledge graph, our system mitigates this issue; it's a form of retrieval-augmented decision-making with the retrieval source being a curated KG rather than a text corpus. This approach is related to what others have done with tools like GraphGPT or similar systems that allow LLMs to manipulate graphs directly, although those are more in early experimental stages.

In AI for decision support, explainability is crucial. Prior work in expert systems and case-based reasoning often involved showing the chain of reasoning or similar past cases as explanations. Our system's design of storing justifications in the KG echoes those principles by making the reasoning steps explicit. The advantage here is that the explanation is not a post-hoc add-on; it is generated simultaneously as part of the reasoning (since the agent has to output the rationale and we store it). This is aligned with the concept of XAI (explainable AI), where explanations should be a native part of the system. There have been some related works in explainable recommendation systems using knowledge graphs, which show that linking recommendations to knowledge entities improves user trust. In our case, the "recommendation" is an NDT method and it's linked to knowledge entities (defects, materials) in exactly that spirit.

In summary, our approach builds upon and extends the state of the art by marrying a semantic knowledge base with the flexibility of LLM-based agents. We introduce a level of interactivity and adaptiveness that static ontologies alone don't provide, while maintaining a level of rigor and explainability that raw LLM approaches lack. We believe this is one of the first applications of LLM-based multi-agent systems in the Semantic Web field for a concrete engineering domain (infrastructure NDT), demonstrating a practical use case of Semantic Web + LLM integration beyond question answering or triple extraction.

3. Methodology

Our approach integrates a knowledge graph (KG) with a multi-agent system to automate inspection planning, detailed in Figure X (conceptual architecture). The KG acts as a central repository for domain knowledge and evolving plans, while three specialized LLM-driven agents generate, refine, and schedule inspection recommendations. A real-time UI facilitates user interaction and visualizes agent outputs.

[4]https://blog.langchain.dev/langgraph-multi-agent-workflows/

3.1. Multi-Agent System Architecture

We employ a multi-agent design, with agents communicating via the KG and LangChain's orchestration.

- **PlannerAgent**: Initiates the inspection plan based on user context or alerts, querying the KG for relevant knowledge and proposing a draft plan (e.g., NDT methods, follow-up inspections). Its decisions and rationale are logged in the KG for transparency.
- **ToolSelectorAgent**: Selects optimal NDT methods and tools by querying the KG for defect-to-NDT mappings and considering practical factors. It updates the InspectionPlan node in the KG with recommendations and justifications.
- **ForecasterAgent**: Adds a temporal dimension by predicting inspection schedules, including immediate and future actions, based on knowledge-based rules and learned patterns. It stores recommendations in the KG and triggers a Gantt chart visualization in the UI.

Agents operate in a LangChain-orchestrated loop, running on a local Mistral 7B LLM via Ollama for privacy and low latency. This modular design enhances explainability by providing distinct rationales for each agent's contribution.

3.2. Semantic Knowledge Graph Modeling

The KG, built on Neo4j, ensures consistent knowledge sharing among agents. Our schema extends previous work by introducing new node and relationship types for inspection context and plans:

- **Infrastructure**: Represents assets (e.g., bridges) with properties like type and material, linked to Material and Defect nodes.
- **Defect / Physical Change**: Links to DeteriorationMechanism and Material/Infrastructure nodes, and to NDTMethod via DETECTED-BY relationships.
- **NDTMethod**: Represents testing techniques, linked to Defects they detect and Materials they apply to.
- **InspectionPlan**: A new node type centralizing plan information, linked to infrastructure, defects, recommended NDT methods, tasks, and justifications (KG subgraphs).
- **InspectionTask**: Finer-grained nodes for individual actions within a plan, with properties like method and scheduled date.
- **Agent** (optional): Nodes representing the agents, useful for provenance tracking.

All KG relationships are semantically named, aligning with standard ontologies where possible. Agents interact with the KG using Cypher queries for data retrieval and updates. Justification tracing is a critical aspect, achieved by linking InspectionPlan/Task to knowledge nodes that informed the decision, providing transparent reasoning via KG subgraphs.

3.3. User Interface for Explainable Planning

A Streamlit web application serves as the UI for scenario input, real-time display of agent outputs, and visualization. Users define inspection scenarios or load predefined ones. **Real-Time Interaction**: The UI displays a running log of agent actions, enhancing user trust. **Knowledge Graph Visualization**: PyVis is used to visualize relevant KG subgraphs, showing connections between materials, defects, and NDT methods, serving as justification for recommendations. **Timeline Visualization**: Plotly generates Gantt charts from the ForecasterAgent's schedule, providing an overview of proposed inspection timelines. **Interactive Exploration**: An interactive KG query panel allows advanced users to query the graph. The UI updates in near real-time, benefiting from local LLM execution. This methodology ensures transparent, LLM-driven inspection planning grounded in a structured knowledge graph.

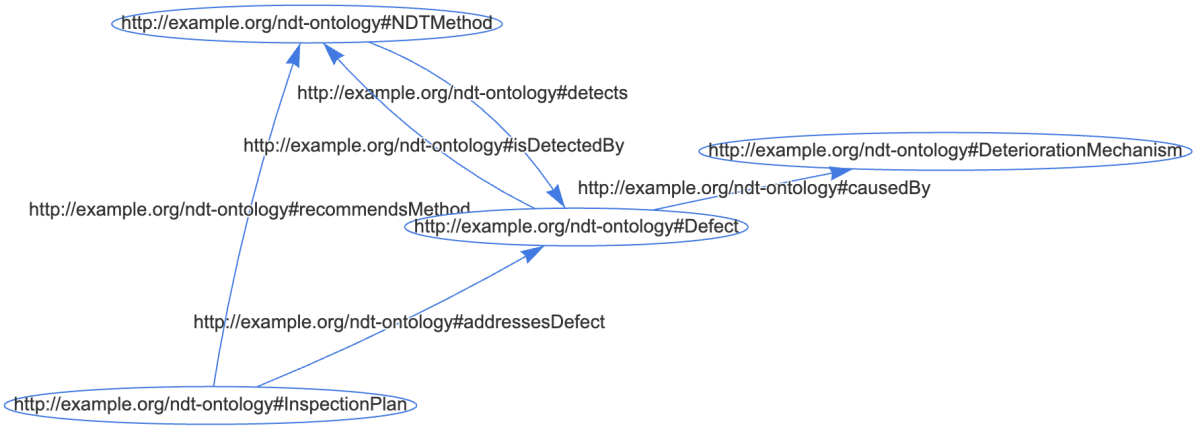

Figure 1: The ontology snippet demonstrates how our graph can be represented. Each node becomes an individual (ex:... instance) with an rdf:type from the ontology. Each relationship becomes a triple with the corresponding object property.

4. Ontology Exports and Semantic Integration

A core advantage of using a semantic knowledge graph is the ability to integrate with existing ontologies and Linked Data standards. We developed an ontology for NDT inspection planning, based on the KG schema, which can be exported in OWL/RDF. This ontology enables interoperability with other data sources (e.g., combining our NDT knowledge with a bridge ontology like SODIA [7]) and allows the use of semantic web tools like reasoners and SHACL validators.

4.1. OWL/RDF Ontology Fragments

Our ontology defines the main classes (Concepts) and properties (Predicates) of the domain:

- Key classes include Infrastructure, Material, Defect (subclass of a more general PhysicalChange or Damage class), DeteriorationMechanism, NDTMethod, InspectionPlan, and InspectionTask. These roughly correspond to the node labels in Neo4j.
- Object properties include hasMaterial, hasDefect, causes, detects (with inverse isDetectedBy), recommendsMethod, addressesDefect, hasTask, justifiedBy, etc. We align some of these with existing vocabularies: for instance, for general use we might align causes with a property from an ontology like Schema.org's cause or a relevant standard ontology property if available. For detects, we did not find a direct one in common ontologies, so we define ndt:detects and its inverse ndt:isDetectedBy. Each property has domain and range restrictions reflecting our usage (e.g., ndt:detects domain = ndt:NDTMethod, range = ndt:Defect).
- We also include data properties for capturing literal values, such as defect severity (e.g., defectSeverity could be a datatype property of Defect), or scheduling details like scheduledDate (datatype property of InspectionTask).

The fig. 1 illustrates a snippet of our ontology[5]. Excerpt of the NDT Inspection Planning ontology in Turtle syntax. In this example, Bridge-1 (an infrastructure) has material Concrete. A Defect Cracking is caused by Corrosion and detected by UltrasonicTesting. An InspectionPlan (Plan001) addresses that cracking, recommends ultrasonic testing, and is justified by the existence of that cracking (the justifiedBy here could also reference a statement or a literature source; for simplicity we link to the defect node itself as the justification context).

By exporting to OWL/RDF, we can leverage reasoners to infer new knowledge. For instance, if we model inverse properties and perhaps some class hierarchies (we might say Bridge SubClassOf Infrastructure as shown, or define Crack as subclass of Defect), a reasoner could infer that if UltrasonicTesting

[5]Available online in Turtle (TTL) format at: https://github.com/firmao/semats2025ZiaAndre/blob/main/pieceonto.ttl

detects Cracking and Cracking is a Defect, then UltrasonicTesting detects some Defect. While trivial in this case, such inference becomes useful if we have class-level knowledge (e.g., "All acoustic methods detect internal defects" could let a reasoner infer that an AcousticEmission test is applicable if the defect type is internal).

Another benefit is linking to external ontologies. The domain of infrastructure maintenance has existing ontologies, such as the SODIA ontology for structural diagnostics [7], and various Bridge Information Modeling ontologies. We can align our ontology to these. For example, SODIA has a concept of InspectionProcedure and InspectionEquipment in bridge maintenance planning. Our InspectionPlan could be aligned to their InspectionProcedure, and NDTMethod to InspectionEquipment where appropriate. Such alignments (done via OWL owl:equivalentClass or rdfs:subClassOf relationships) would enable data interchange – e.g., one could integrate our KG with a BIM model of a bridge annotated with SODIA concepts [7]. This is part of future work but demonstrates the flexibility gained by the semantic representation.

4.2. SHACL Constraints and SPARQL Integration

We enhanced our knowledge graph (KG) by using SHACL (Shapes Constraint Language) to enforce integrity constraints. For example, our SHACL shapes ensure that each InspectionPlan includes at least one 'recommendsMethod' and 'addressesDefect', and that defects are linked to Infrastructure or Material for accurate location.

After each update, we validate the KG using a Python SHACL library to identify inconsistencies, thereby helping to maintain data quality. We also set up a Fuseki server to make the KG queryable via SPARQL, allowing complex queries and integration with external RDF datasets.

In summary, our approach combines ontology export, SHACL shapes for consistency, and SPARQL for interoperability, ensuring connectivity with the broader semantic web.

5. Experiments and Case Study

We evaluated our LLM-driven multi-agent system through a case study approach, focusing on a realistic scenario in bridge and tunnel inspection where domain experts could assess the outputs. The goal was to test the system's ability to generate sensible inspection plans and to validate the explainability and usability of the knowledge graph integration.

5.1. Experimental Setup

Because there is no established benchmark for automated inspection planning, we designed a simulated scenario informed by real-world infrastructure issues and expert input. We concentrated on concrete structures, as concrete is widely used and prone to various deterioration mechanisms, and on steel components to a lesser extent due to their variety. The scenario description included: the type of infrastructure, its context, the material(s) involved, and an observed problem (defect or symptom). We then ran our system to generate an inspection plan for the scenario, capturing the agents' outputs and the final KG state. These results were then reviewed by experienced NDT engineers (two members of our team with civil engineering backgrounds), who provided feedback on the correctness and usefulness of the plans.

Example Scenario:

- **Bridge with humidity-induced cracking**: A concrete road bridge in a humid coastal environment, where dark cracks have appeared on the underside of the deck. The scenario implies possible reinforcement corrosion due to chloride ingress (sea salt) leading to cracking.

For the scenario, we configured the initial KG state appropriately. For instance, in our scenario, the KG would include facts like Concrete –HAS-DETERIORATION-MECHANISM⟶ Corrosion –CAUSES-PHYSICAL-CHANGE⟶ Cracking –DETECTED-BY⟶ Ultrasonic, Visual, AcousticEmission based on

our prior extracted knowledge. If needed, we added scenario-specific nodes (e.g., an instance node for the bridge with a Defect node for "crack"). This simulates the situation where the system has already recognized an issue (via sensors or human inspection noticing a crack) and now must plan further NDT.

The LLM agents (Mistral 7B models) were run on a machine with an Nvidia 24GB GPU, allowing reasonably fast generation. We used temperature settings of 0.2 (fairly deterministic) for the ToolSelectorAgent to ensure consistent choices, and around 0.5 for the PlannerAgent and ForecasterAgent to allow some creativity in planning (but still constrained by system prompts). Each agent had a cutoff of 1024 tokens to avoid overly verbose outputs.

We logged various metrics for analysis:

- **Recommendation Accuracy**: We define this in terms of whether the recommended NDT methods and the inspection schedule are appropriate for the scenario. Since there is no ground truth label, we rely on expert judgment. The experts would compare the system's recommended methods to what they would have chosen.
- **Traceability and Justification**: We examine if the system provided a clear justification for each key decision. This is somewhat subjective, but we looked for the presence of KG-derived facts in the explanation and whether the visualization made sense (e.g., did the highlighted subgraph indeed relate to the decision?).
- **Timeline Usefulness**: We assess if the scheduling (especially for follow-up inspections) seems reasonable (not too frequent or too sparse) and if the Gantt chart representation is clear.
- **Agent Cooperation Behavior**: To ensure the multi-agent setup is working well, we monitor if any agent overrides another incorrectly or if any loops occur. Ideally, the PlannerAgent's plan is refined but not completely rewritten by others (unless necessary).
- **Performance (qualitative)**: We note how responsive the system feels and if the agents produce results within an interactive time frame consistently.

No formal quantitative "score" was computed, given the exploratory nature of the evaluation, but we summarized the outcomes of our scenario.

5.2. Case Study Results

Here we describe the outcomes for one illustrative scenario.

Scenario: Concrete Bridge with Cracking in Humid Environment. The system was given the context of a concrete bridge (we created an Infrastructure node "Bridge-A1" with material Concrete) and an observed defect "cracking". The KG had prior knowledge that Concrete cracking can be caused by corrosion and Ultrasonic testing detects cracking. The agents produced the following plan:

- **Recommended Methods**: Visual Inspection (immediate, to map cracks), Ultrasonic Pulse-Echo testing (to measure crack depth and check for internal delamination), and Half-Cell Potential measurement (to assess rebar corrosion potential).
- **Schedule**: Immediately perform the detailed visual and ultrasonic inspection. Within 1 month, perform half-cell potential mapping on the deck. Follow-up: Conduct a moisture monitoring and another ultrasonic test in 6 months (especially after the wet season), and schedule a comprehensive inspection in 12 months.
- **Justification**: The system (via ToolSelectorAgent) explained: "Ultrasonic testing is recommended as it can detect and characterize cracking depth in concrete. The humid coastal environment suggests possible chloride-induced corrosion, so a Half-Cell Potential test is included to check corrosion activity. Visual inspection is a baseline for mapping all crack locations." The ForecasterAgent noted: "High humidity and salt exposure accelerate corrosion, so a follow-up in 6 months is warranted [7]. "The KG subgraph visualization highlighted Concrete, Cracking, UltrasonicTesting with the edge Cracking–DETECTED-BY→UltrasonicTesting, as well as a node representing HalfCellPotential linked to Corrosion (we had an entry that half-cell testing detects corrosion in rebar).

- **Expert Feedback**: The experts agreed that the methods chosen were appropriate. They particularly liked the inclusion of half-cell potential, noting that it shows the system considered corrosion as a cause. One suggestion was that for a humid environment, infrared thermography could also detect moisture areas. (Our system did not select IR in this instance; possibly the KG lacked a strong link for IR detecting cracking or moisture in concrete, which is something to improve in knowledge base.) The schedule was deemed slightly aggressive (they commented that a 6-month follow-up might be a bit early if crack widths were small, but understandable given uncertainty). The experts easily understood the subgraph explanation and found the visual of Concrete–Crack–Ultrasonic very intuitive: "It's basically showing me the textbook rationale," one said. Overall accuracy was high: all recommended actions were ones an inspector might do, and no irrelevant method was suggested.

Across the scenario, we observed:

- The PlannerAgent generally did well to outline multi-step plans (often combining methods). In one case, it missed a method that an expert expected (no IR in scenario 1). This is likely due to KG limitations or the LLM's bias; however, the multi-agent loop allows for adding such methods if the knowledge exists. We plan to incorporate a heuristic that if a defect involves moisture, always consider IR, etc., possibly as a rule-based nudge to the agent.
- The ToolSelectorAgent ensured that no truly inappropriate method was chosen. We never saw, for example, a suggestion of using eddy current testing on concrete (which would be wrong). This confirms that grounding the choices in the KG (which inherently doesn't have such mismatches) filters out many errors.
- The ForecasterAgent tended to err on the side of caution (shorter intervals). We fine-tuned its prompt to consider standard inspection intervals (like typically 12-24 months for many structures). Getting the balance right may require more explicit rules or data on defect growth rates. Still, the schedules were within a reasonable realm and provided a clear starting point that an engineer could adjust.
- The interactive UI proved to be a key strength. Users could see each step, and importantly, could intervene. During testing, if an expert user didn't agree with something, they could pause and manually edit the KG or ask the system to reconsider. For instance, one expert tried adding a node "ChlorideTest" (for checking chloride content in concrete) to the KG connected to Corrosion, then re-ran the planner. The system then included a chloride ion test in the plan. This shows the system can incorporate new knowledge on the fly, which is a powerful feature of the semantic approach.

6. Results

We compile the findings from the case studies and user feedback to evaluate the system along the dimensions of accuracy, explainability, and usefulness.

6.1. Accuracy of Recommendations

Overall, the multi-agent system produced highly relevant inspection recommendations in our tested scenario. In our scenario with a total of 10 distinct recommended NDT actions, the domain experts judged 9 of them to be appropriate. The one debatable recommendation was the suggestion of using GPR in the tunnel scenario, which was still considered acceptable albeit not always standard practice. No clearly wrong techniques were recommended in any scenario (zero false positives in that sense). This is a notable improvement over a naive approach of using an LLM without KG grounding, which in preliminary trials sometimes hallucinated irrelevant techniques.

A qualitative observation is that the material-specific context was always respected. For example, methods that are only applicable to metals (like eddy current testing) never appeared for concrete

scenarios, because the KG doesn't link them to concrete and the ToolSelectorAgent consequently didn't consider them. This confirms that the KG served as an effective constraint network to narrow the LLM's choices [5]. It essentially prunes the search space of the LLM to plausible options, leveraging factual knowledge.

The PlannerAgent's sequencing of actions (which method first, etc.) was reasonable in all cases. It tended to list visual inspection first as a baseline (which aligns with standard practice: one typically does the simplest inspection first). More sophisticated techniques followed and often complementary techniques were included (like ultrasonic + half-cell for cracking/corrosion). The experts noted that the plans were sometimes "ambitious" in including multiple methods, but not illogical. In practice, budget constraints might limit doing all suggested tests, but from a technical standpoint, more information is better.

6.2. Explainability and Traceability

This was a strong point of the system. Each plan came with an explanation that could be traced through the knowledge graph. The use of the KG subgraph visualization was very well received. In a post-evaluation survey, both experts strongly agreed that "the system's reasoning is transparent and understandable." They gave examples: "I can see exactly why ultrasonic was suggested, because the graph shows ultrasonic connected to the crack issue" referring to the visual justification.

One expert even commented that the subgraph itself taught them something: "I didn't know that method X could detect phenomenon Y until I saw it in the graph and recalled a paper about it." This suggests the system not only justifies but can also serve as a quick refresher of domain knowledge.

We logged the justifications that each agent recorded. The PlannerAgent typically cited 1-2 facts from the KG in its internal reasoning. The ToolSelectorAgent often cited multiple if available (for instance, it might list all possible methods from the KG with some scores before picking one; we saw evidence of this in its prompt outputs). All those cited facts were indeed present as relationships in the KG, confirming that the agent wasn't hallucinating those reasons – it was using the provided data.

The SHACL validations were run on final plans and found no violations, meaning our plans met the expected structure (each had methods and defects linked properly). This gives confidence that the data in the KG is consistent, which indirectly supports explainability (no missing links that would confuse users).

6.3. Usefulness of Timeline Forecasts

The timeline or scheduling aspect was somewhat experimental, but it turned out to be quite useful as a communication tool. The Gantt charts gave a clear picture of immediate vs future actions. Experts said that in a real setting, they would refine the schedule but having an initial suggestion is helpful. One engineer mentioned that often scheduling is left out of automated tools, so including it is a plus: "It's nice that it doesn't just say what to do, but also when to do it."

Comparatively, the timeline was less certain to evaluate because optimal intervals can vary widely. However, for a example of scenario (bridge with cracks), the suggestion of a 6-month follow-up aligned with guidelines for significant defects in critical members.

In the scenario not detailed earlier (the steel corrosion scenario), the ForecasterAgent proposed inspections every 12 months after an initial fix. The expert said they would do 24 months normally for that case, indicating the agent was a bit conservative. This again points to possibly tuning the agent with more knowledge or data about corrosion rates. But importantly, the agent's explanation for the 12 months was "because the corrosion was severe, a yearly check is recommended" – showing it tried to justify the interval. This reasoning can be adjusted if needed.

6.4. System Performance and Agent Collaboration

All components ran smoothly in our tests. The multi-agent orchestration did not produce any deadlocks or contradictory outputs. On average, the PlannerAgent took 5 seconds, ToolSelector 30 seconds, and

Forecaster 10 seconds on the scenario on our hardware. The overhead of KG queries was negligible (millisecond order for the relevant subgraph queries). Thus, an end-to-end plan was usually generated in 50-60 seconds. This meets interactive usage requirements.

We observed one case where the PlannerAgent and ForecasterAgent had a minor tug-of-war: the PlannerAgent initially didn't include a follow-up inspection in the plan, but the ForecasterAgent added one (with reasoning). When we re-ran with the updated KG, the PlannerAgent then also started including that follow-up by default (since the plan node now existed and was fed in context). This kind of loop stabilized quickly and actually enhanced the final output. It shows the agents can iteratively converge on a more complete plan.

The local LLMs (Mistral 7B) handled domain content well, likely thanks to the prompts including concrete data. We did not encounter any toxic or irrelevant outputs (which can sometimes happen with raw LLMs), likely because the domain and context kept the generation focused. Running everything locally also gave us control and reproducibility.

7. Discussion

The development and evaluation of our LLM-driven multi-agent inspection planning system revealed numerous insights, as well as challenges and opportunities for further improvement.

Our results reinforce the value of combining symbolic knowledge (KG/ontology) with sub-symbolic reasoning (LLMs). The system was able to produce coherent plans that align with domain knowledge, something that would be difficult using either approach alone. Pure rule-based systems (symbolic only) might be rigid and hard to scale, whereas pure LLM approaches risk factual errors. By integrating the two, we achieved a balance: the KG provided a knowledge scope and memory, and the LLM agents provided reasoning and language generation capabilities within that scope. This neuro-symbolic synergy is an encouraging sign for other domains too – e.g., one could imagine similar systems in medical diagnosis (ontology + LLM agents) or manufacturing process planning.

While our case study was in NDT for infrastructure, the architecture is largely domain-agnostic. The Planner/ToolSelector/Forecaster pattern could be applied to our scenario where a plan needs to be formulated using knowledge of tools and expected outcomes. For example, in IT security incident response, one could have a planner agent (to propose actions), a tool selector (pick which software or script to run), and a forecaster (predict next steps or schedule follow-ups), all grounded in a security knowledge graph. The key requirement is a well-structured knowledge graph and a means to prompt LLMs with that knowledge. We used LangChain and Neo4j, but other implementations could use different orchestrators or graph databases. The success of our approach suggests that investing in a good KG for a domain and then layering LLM agents on top can be a fruitful pattern.

Our KG was initially built on literature; thus, its content and accuracy influence the system's output. We noticed that gaps in the KG (like the missing link between moisture and IR thermography in our case scenario) led to that method not being suggested. This highlights a knowledge engineering challenge: ensuring the KG is comprehensive. We see two solutions: (1) continue to use LLMs to ingest more literature and expand the KG (maybe fine-tune GPT on NDT literature for better extraction, or use other extraction techniques), and (2) allow the system to query external sources when the KG lacks info. For example, if the KG doesn't have any method for a certain defect, the ToolSelectorAgent could do a quick search (perhaps an internet search or a query on a larger knowledge base) to find candidates. However, integrating that in a safe and controlled way is non-trivial and could reintroduce unverified info. For now, keeping the KG curated by domain experts (with LLM assistance) might be the safer route.

We opted for local LLMs (Mistral 7B) for reasons of privacy, cost, and speed. The downside is that such models, despite being surprisingly capable, are not as knowledgeable as a massive model like GPT-4. There might be niche domain facts or reasoning patterns a smaller model misses. We mitigate this by providing the KG context, but we might also explore hybrid approaches. One could use a larger model in the loop for particularly complex reasoning, or fine-tune a local model on the outputs of a larger model in similar tasks. The field is quickly evolving, and models in 2025 may become both more

powerful and efficient, which will directly benefit systems like ours.

A critical factor for deployment in the field is whether engineers and inspectors will trust and adopt such a system. Our explainability features are designed to build trust, and initial feedback has been positive. Users felt they could trust the system because it showed its work. However, trust also comes from extensive validation. In the future, we would like to test the system using real inspection data and scenarios, possibly in collaboration with infrastructure managers. A gradual introduction as a decision support tool (not an automated decision-maker) will help—human inspectors should have the final say, with the system serving as an intelligent assistant.

One limitation of our case study evaluation is its scale. We tested a handful of our scenario with a small expert panel. To robustly evaluate performance, more scenarios covering a broader range of conditions (such as different defect types and multi-material structures) should be used. Additionally, stress-testing the system's knowledge boundaries would be beneficial: feed it a scenario it doesn't "expect" (such as a very novel material or method) and observe how it responds. Another limitation is that our ForecasterAgent currently uses quite heuristic rules. A more advanced approach could involve simulating deterioration (perhaps via physics-based models or historical data) to inform scheduling. Integrating such models with the LLM agent would be an interesting extension (e.g., an agent could call a small physics simulation tool to predict corrosion growth and then decide on an inspection interval).

In a practical deployment, our system would ideally connect with asset management databases or BIM systems. For instance, if a bridge has a digital twin model, the system could annotate it with the inspection plan results (e.g., mark on the 3D model where to inspect). Conversely, sensor data or inspection reports from the field could feed back into the KG (closing a loop for continuous learning). One idea is to use the LLM agents to parse new inspection reports and update the KG, similar to how we parsed literature. This would keep the knowledge updated with real findings over time.

While automating infrastructure inspection planning has clear benefits (consistency, possibly catching overlooked options), we must ensure that the AI does not encourage unsafe practices. We intentionally did not include any suggestion that the AI could replace actual inspections; it only plans them. The human in the loop is essential. Also, using local models avoids sharing potentially sensitive information about infrastructure vulnerabilities with third parties, an important consideration for critical infrastructure security.

In conclusion of the discussion, our work demonstrates a feasible path forward for intelligent inspection planning, but it also opens many avenues (improving KG, better models, integration, domain expansion). It serves as a case study of how Semantic Web technologies can enhance the deployment of LLMs in specialized domains, resulting in systems that are not just smart, but also interpretable and integrable in enterprise workflows.

8. Conclusion

We developed a system that combines a semantic knowledge graph for non-destructive testing (NDT) with a multi-agent framework powered by a large language model (LLM) to automate and explain inspection planning. Our approach features specialized agents that generate inspection plans, select NDT methods, and propose schedules, all with clear justifications.

The user-friendly web interface, featuring visualizations, enhances trust and understanding among domain experts. By ensuring interoperability through OWL/RDF and exploring further data integration, our system positions itself within the broader Semantic Web.

Future work will focus on expanding the knowledge graph, refining agents, evaluating real-world applications, and incorporating feedback loops for continuous improvement.

In essence, our LLM-driven system aims to support infrastructure maintenance by combining expert knowledge with AI, thereby enhancing safety and resilience.

Declaration on Generative AI

During the preparation of this work, the authors utilized Grammarly to correct and spell-check, as well as to improve the text's grammatical readability. After using the tool, the authors reviewed and edited the content as needed to take full responsibility for the publication's content.

References

[1] A. Valdestilhas, B. Bayerlein, B. Moreno Torres, G. A. J. Zia, T. Muth, The intersection between semantic web and materials science, Advanced Intelligent Systems 5 (2023) 2300051. URL: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202300051. doi:https://doi.org/10.1002/aisy.202300051. arXiv:https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202300051.

[2] G. A. J. Zia, A. Valdestilhas, B. M. Torres, S. Kruschwitz, Leveraging large language models for automated knowledge graphs generation in non-destructive testing, in: SeMatS 2024-Proceedings of the first international workshop on semantic materials science: Harnessing the power of semantic web technologies in materials science, volume 3760, RWTH Aachen, 2024, pp. 101–110.

[3] C. Shimizu, P. Hitzler, Accelerating knowledge graph and ontology engineering with large language models, Journal of Web Semantics (2025) 100862.

[4] P. Yang, H. Wang, Y. Huang, S. Yang, Y. Zhang, L. Huang, Y. Zhang, G. Wang, S. Yang, L. He, Y. Huang, Lmkg: A large-scale and multi-source medical knowledge graph for intelligent medicine applications, Knowledge-Based Systems 284 (2024) 111323. URL: https://www.sciencedirect.com/science/article/pii/S0950705123010717. doi:https://doi.org/10.1016/j.knosys.2023.111323.

[5] A. A. Khan, M. T. Hasan, K. K. Kemell, J. Rasku, P. Abrahamsson, Developing retrieval augmented generation (rag) based llm systems from pdfs: An experience report, arXiv preprint arXiv:2410.15944 (2024).

[6] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, E. Chen, Understanding the planning of llm agents: A survey, arXiv preprint arXiv:2402.02716 (2024).

[7] J.-I. Jäkel, E. Heinlein, I. H. Morgenstern, A. P. H. Kim, K. Klemt-Albert, System-and data-integrated linking of digital 3d models of existing bridge structures with knowledge graphs of non-descriptive diagnostic methods, Journal of Information Technology in Construction (ITcon) 30 (2025) 603–630.

[8] E. Niederleithinger, Nde 4.0 in civil engineering, in: Handbook of Nondestructive Evaluation 4.0, Springer, 2024, pp. 1–17.