Navigating through Work Items in Issue Tracking Systems via Natural Language Queries

Delina Ly*§, Sruthi Radhakrishnan ‡, Fatma Başak Aydemir §, and Fabiano Dalpiaz §
*Application Lifecycle Management, VX Company, The Netherlands, dly@vxcompany.com
§Department of Information and Computing Sciences, Utrecht University, The Netherlands, {f.b.aydemir, f.dalpiaz}@uu.nl
‡ itemis, Germany, sradhakrishnan@itemis.com

***Abstract*—Issue Tracking Systems (ITSs) serve multiple purposes in software development: they represent requirements and bugs, facilitate communication between team members, and support project management by assigning tasks. As software systems and projects increase in complexity, the number of work items and their relationships grows considerably, forming a complex network that is difficult and time-consuming for team members to navigate using traditional ITSs. Motivated by the interest of two companies in exploring the use of Artificial Intelligence in software development, we introduce the GraphRAG Dialogue Insights (GDI) framework. GDI uses knowledge graphs, Retrieval-Augmented Generation (RAG), and Large Language Models (LLMs) to enable users to query work items in natural language (NL). In addition to a NL response, GDI returns supporting information to help users understand how the LLMs queried the knowledge graph. We empirically validate GDI through two proof-of-concept implementations, each addressing a use case: i) onboarding and ii) trace link recovery. Initial user-centered validations indicate that practitioners find the system intuitive, useful, and accurate in generating responses. Moreover, they express willingness to adopt the system and identify additional potential use cases, including impact analysis, troubleshooting, and issue resolution. However, further refinement is required to deploy the system in practice.**

***Index Terms*—Issue Tracking Systems, Large Language Models, Prompt Engineering, GraphRAG, Onboarding, Trace Link Recovery**

I. Introduction

Software development is a collaborative activity where effective communication and knowledge sharing are crucial for project success [1]. Knowledge about processes, systems, and projects is often dispersed across several teams, individuals, and tools. As software companies transition from co-located work setups to hybrid and fully remote work arrangements, team members face increased challenges in seeking this knowledge and guidance from their colleagues to complete tasks [2].

Software development teams often use Issue Tracking Systems (ITSs) to capture this knowledge. ITSs include lightweight representations of requirements (often as user stories), reported bugs, and tasks that represent responsibility assignments to team members. While each team may work differently, ITSs offer a shared, structured knowledge base.

ITSs are a valuable and ever-evolving source of knowledge, as team members can read, update, expand, and act on the captured information. In this paper, we use the term *work items* to refer to any trackable unit of work in the software development process, including epics, features, user stories, bugs, and tasks. Detailed work items that are adequately linked to one another are helpful in supporting software development teams. However, as systems and projects increase in complexity, navigating work items using traditional ITSs becomes challenging and time-consuming, as these systems have limited functionality for searching and acquiring knowledge, making it difficult to answer complex questions required for tasks.

This research is driven by the needs of two software development companies that are interested in seizing the technological opportunity provided by Large Language Models (LLMs) to support software development teams in the assisted exploration of ITSs. LLMs have powerful natural language (NL) capabilities, which have been effectively applied to various software engineering tasks, such as onboarding [3], [4] and trace link recovery (TLR) [5]–[8]. However, due to input length constraints, LLMs are limited in processing data from software systems and projects. Information retrieval techniques, such as Retrieval-Augmented Generation (RAG), can enhance their processing capabilities [9]. GraphRAG extends RAG by incorporating knowledge graphs, which structure information as a network of entities and the relationships between these entities. These graphs have been used to model software repositories, enabling complex queries and providing insights [10].

In this paper, we propose the GraphRAG Dialogue Insights (GDI) framework, which uses knowledge graphs, RAG, and LLMs to enable users to query work items in NL and trace how the LLM integrates the user question with the retrieved information to generate a response. We validate GDI through two proof-of-concept implementations at two software companies, each addressing a specific use case that requires navigating through work items: *i.* onboarding and *ii.* TLR.

The first use case, *onboarding*, refers to integrating a new team member into an existing project or system and is validated at itemis, where the second author is employed. itemis is a German software company founded in 2003 that specializes in IT software development and consulting, with around 200 employees. Building on their prior application of GraphRAG for onboarding in the aerospace sector [11], itemis aims to explore whether GDI could benefit their clients using ITSs.

The second use case, TLR, is validated at VX Company, a Dutch software organization founded in 1988, where the first author is employed. VX Company develops and maintains customized software solutions and has around 200 employees. By generating or repairing trace links between work items and

other software artifacts [12], TLR may support tasks such as bug localization [13] and change impact analysis [14]. VX Company is interested in exploring whether GDI can support their software development teams in their daily work. To address both use cases, we define our main research question:

MRQ. *How is GDI perceived by users for the tasks of i. onboarding ii. trace link recovery?*

This research question is refined into five research questions to validate GDI based on user-centered criteria. Each research question comes in two variants, *i.* one for the onboarding use case, and *ii.* one for TLR use case.

RQ1. *How do users perceive the usability of GDI compared to ITSs?* This question assesses users' perception of how easily and effectively GDI supports them in achieving their goals.

RQ2. *How do users perceive the accuracy of the responses generated by GDI?* This question addresses users' subjective evaluations of the correctness of returned information.

RQ3. *How do users perceive the adequacy of the supporting information provided by GDI?* This question assesses users' subjective evaluations of whether the generated query and the retrieved context adequately support their understanding of how the LLM queried the graph based on the question.

RQ4. *How do users perceive GDI's usefulness?* This question assesses the users' perception of the value provided by GDI.

RQ5. *How likely are users to adopt GDI in their work?* This question evaluates users' willingness to use GDI to find information in ITS to support their tasks.

This paper makes the following contributions:

- We introduce GDI, a GraphRAG-based framework developed to better address the diverse information needs of software development teams. We provide the source code of GDI, the original and improved open-source knowledge graphs [15], along with the logs, an informed consent form, a user-centered validation session protocol, and the surveys and interviews questions in a replication package [16]. Our framework enables local setups that address security concerns critical to practitioners.
- We empirically validate GDI through two proof-of-concept implementations, each addressing a use case interesting for the software companies: *i.* onboarding for itemis and *ii.* TLR for VX Company.

The remainder of the paper is structured as follows. In Section II, we discuss the technical background for our framework. In Section III, we present GDI detailing its architecture, user interaction, knowledge graph construction process, and implementation. In Sections IV andV, we report on the user-centered validations for the two use cases. In Section VI, we list the lessons learned and discuss the threats to validity and limitations. In Section VII, we contrast our work with existing studies. Finally, in Section VIII, we conclude the paper.

II. Background

We present the necessary technical background for our framework: prompt engineering (Section A), RAG (Section B) and GraphRAG (Section C).

A. Prompt engineering

Prompt engineering is a continuous process of carefully designing and refining input queries or *"prompts"*. These prompts can include instructions, context, input data, and response guidelines to assist LLMs in generating relevant and accurate responses aligned with specific goals [17]. Effective prompts can improve LLM performance [5], enable new use cases, and save valuable resources [17]. Additionally, prompt engineering provides a framework for documenting prompt patterns that facilitate problem solving and knowledge transfer among developers and users [18]. Zero-shot prompting leverages carefully crafted prompts to enable LLMs to perform tasks beyond their explicit training, while few-shot prompting improves task understanding by integrating input data, such as examples of user queries, into the prompts [19]. Moreover, persona-based prompting instructs LLMs to adopt a specific perspective or *"persona"*, guiding them in determining which details to focus on and what types of outputs to generate [20].

B. RAG

RAG is a technique that improves the performance of LLMs on knowledge-intensive tasks [9]. RAG consists of two main components: a *retriever* and a *generator*. To facilitate information retrieval, data from external sources is segmented into smaller chunks and transformed into numeric vectors that represent their semantic meaning using embedding models. These vectors are stored in indexes, allowing the retriever to efficiently search and retrieve the most relevant chunks based on their semantic and syntactic similarity to the user's question. Subsequently, the generator, generally an LLM, combines the retrieved data with the user's question to generate a more accurate and contextually relevant response.

However, RAG has its limitations [21]: it relies on diverse parsers to chunk and embed external data, complicating the data preparation process. Additionally, information may be lost during the chunking and embedding processes, which can potentially reduce response accuracy. Moreover, retrieving information from unstructured text may not fully capture semantic relationships, which can affect the quality of the retrieved information.

C. Graph + RAG = GraphRAG

The limitations of RAG can be addressed using GraphRAG [22], [23], which leverages *knowledge graphs* in three different ways [24]: (1) as a *content source* consolidating dispersed information and uncovering indirect relationships between entities, (2) as a *semantic source* preserving the meanings and relationships of entities to mitigate information loss during chunking and embedding, and (3) as a *structured retriever* retrieving information based on the user's question and the structure of the knowledge graph which inherently captures semantic similarity. The retriever can identify relevant information that is not explicitly mentioned in the initial set of retrieved chunks using the graph structure, leading to more comprehensive and contextually relevant responses [25]. The graph structure also enhances retrieval efficiency and facilitates

comprehensive cross-referencing and traceability analysis [26]. Thus, GraphRAG can be advantageous in software engineering, where artifacts are often interconnected.

The knowledge graph also provides visual insights that can help developers identify patterns and anomalies, interpret results, and optimize performance. In addition, the knowledge graph can be queried using query languages to trace the retriever's traversals through the graph, adding to the transparency, trustworthiness, and interpretability of GraphRAG.

III. GDI: GraphRAG Dialogue Insights

We present GDI and describe its architecture (Section A), user interaction (Section B), our knowledge graph construction process (Section C), and implementation details (Section D).

A. Architecture of GDI

GDI uses GraphRAG to navigate through work items to answer questions from the users. Since it is based on GraphRAG, a knowledge graph, and an LLM are essential components of a GDI instance, together with the GDI Core component that facilitates interaction among the components in GDI. Fig. 1 presents a simplified view of the GDI architecture with a locally hosted LLM. The system can be easily configured to access remotely hosted LLMs via their APIs.

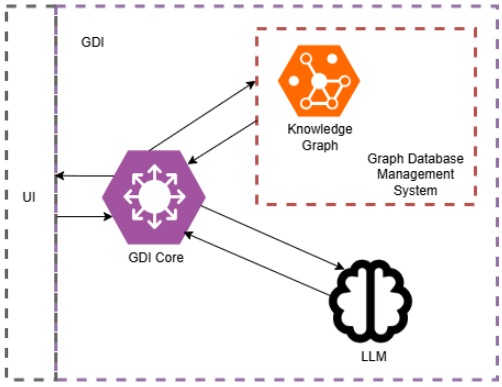

Fig. 1. Architecture diagram of a GDI instance using a local LLM.

The GDI Core component creates prompts and facilitates the interaction between the LLM and the knowledge graph. This component relies on prompt templates to generate prompts at run-time. There are two prompt templates in the system. The first template is used to create the query generation prompt, which incorporates the schema of the knowledge graph and the user's question. The second template is used to create the NL response prompt and includes the selected user persona, the user's question, and the context retrieved from the knowledge graph. Fig. 2 presents an example of the latter template.

B. User Interaction with GDI

Fig. 3 shows the interaction between the user and GDI components, excluding the user interface (UI) component due to space constraints. The user specifies their persona, such as

```
QA_GENERATION_TEMPLATE_SE = """
You are a software engineer responsible for
- Writing, testing, and maintaining high-quality
 code.
 - Contributing to designing scalable and
 maintainable software solutions.
 - Identifying and fixing software defects.
     Based on the question and Cypher response,
     write a natural language answer:
      # - Question: {question}
      # - Cypher response: {context}
Make sure all the following RULES are
considered before a generating response:
# DO NOT BE TOO TECHNICAL.
# DO NOT INCLUDE SUGGESTIONS, QUESTIONS AND
APOLOGIES. """
```

Fig. 2. An example of a prompt template used by the LLM at run-time to generate the NL response prompt.

software engineer and their LLM of choice for the interactions. The user initiates the interaction by asking a question to GDI (M1), for example, *"I found a bug in AuthService, which uses Node.js. Which team members are familiar with this technology and should be contacted?"*. The GDI Core component constructs a prompt using the prompt template that includes the knowledge graph schema and the user's question and sends it to LLM (M2). Based on this prompt, the LLM generates a query in the query language used for the knowledge graph (M3). The GDI Core component queries the knowledge graph with this query (M4). The knowledge graph sends the structured data corresponding to the query as the retrieved context (M5). GDI Core constructs a prompt using this context, the persona selected by the user, and the user's question, and sends it to the LLM (M6). GDI Core receives the NL response (M7). Finally, the output is presented to the user (M8).

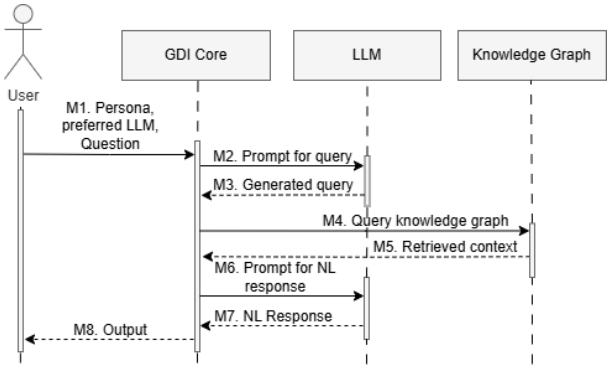

Fig. 3. Sequence diagram outlining the interactions between the user and GDI components.

The browser-based interface is presented in Fig. 4. On the left side (area A), the user can select a persona based on the stakeholder role, which corresponds to the options in the first drop-down menu, such as software engineer or product owner. If multiple LLMs are integrated into the system, the user can indicate their preference through the second drop-down menu. At the top of the interface (area B), we provide example questions and guidelines to the user. The user can access their

conversation history under the “Chat History” item. There is an input field for the user to enter their questions at the center of the interface (area C). Below this field (area D), there are the UI elements to present the output of the GDI instance, including *1.* NL response to the question, *2.* the generated query used to query the knowledge graph, *3.* the retrieved context as a result of the query. The user can save the chat history by clicking on the “Save Session” button (area E).

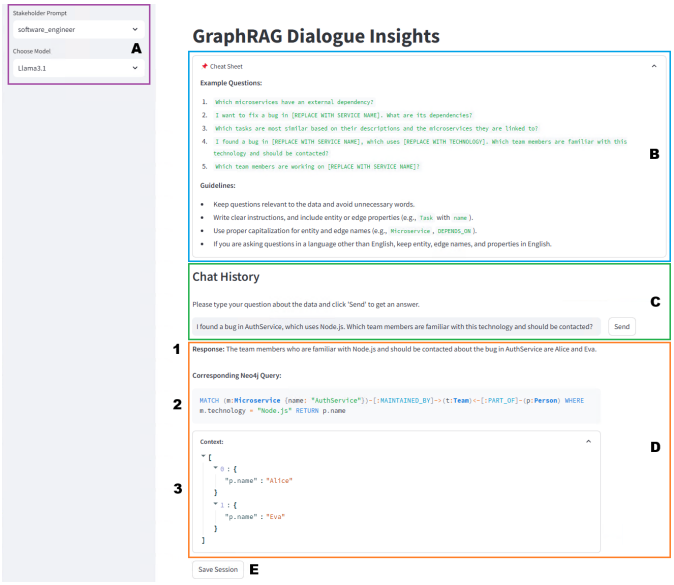

Fig. 4. GDI User Interface.

C. Knowledge graph construction

To build the knowledge graph, the first step is to collect work item-related data, such as epics, features, user stories, and their relationships, from the ITSs by exporting the data in a tabular format, such as CSV. Tabular data can easily be loaded into a graph database management system to implement the graph. However, it is important to preprocess the data by removing sensitive information such as email addresses, and filtering out embedded code fragments due to the GDPR and privacy concerns. Additionally, handling empty fields by assigning a default value is crucial to ensure consistency, as graph database management systems typically do not support null values.

After preprocessing, the next step is to build the schema for the graph. In our schema, we add the work items as nodes and the relationships among them as edges. We also specify the properties of the values, such as the date or the author. Fig. 5 presents an excerpt from our open-source knowledge graph, where the nodes denote a team (blue), a task (orange), and a microservice (purple), and the edges represent the relationships. We store the data in CSV format, where nodes and edges are organized in separate folders under version control using Git to track changes and maintain snapshots of the knowledge graph. For smaller knowledge graphs with fewer than 100 nodes and edges, we directly ingest the data into the graph database management system using queries. For further information on the knowledge graph construction process, we refer to the book chapter of Fensel et al. [27].

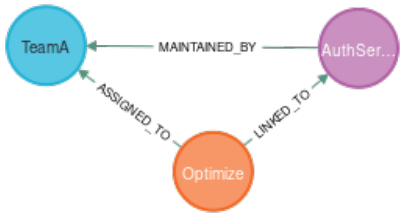

Fig. 5. Snapshot of the open-source knowledge graph.

D. Implementation of GDI

GDI is developed in Python. We created one graph schema for each use case. However, in large software systems, where the same type of work item may have different attributes across different projects, aligning the metadata corresponding to that work item type is necessary. We use a local instance of Neo4j as the graph database management system and connect to it through the Neo4jGraph Python client. Cypher is the query language for Neo4j. Fig III presents the query corresponding to the example user question provided earlier in Section B.

```
MATCH (m:Microservice {name: "AuthService"})
[:MAINTAINED_BY]->(t:Team)<-[:PART_OF]-
(p:Person) WHERE m.technology = "Node.js"
RETURN p.name
```

Fig. 6. Example of a generated Cypher query by the LLM.

We use Streamlit [28] to accelerate the development of GDI and LangChain [29] to streamline the LLM integration. Multiple LLMs can be plugged into the system either by hosting them locally or accessing them through their APIs. We use Ollama to host a local instance of Llama 3.1 due to sensitive and confidential data. In our implementation, we integrate Meta’s Llama3.1 (8B, the temperature is set to 0 to maximize determinism for both use cases) [30] and OpenAI’s GPT-4 [31] into our system. The LLMs can be customized according to preferences, constraints, and available resources. Ollama and Neo4j can also be installed via a Docker setup. The technologies mentioned, such as Neo4j, Cypher, Streamlit in this section, are modular and can be interchanged.

IV. Validation of GDI for Onboarding

We present the dataset for the knowledge graph, the research methods used, and the results from the onboarding use case.

Dataset: We needed a dataset that the participants would not already be familiar with. As such, we selected an open-source synthesized Azure DevOps dataset provided by Neo4j [15] using purposive convenience sampling due to its small size (33 nodes and 52 edges). The dataset represents a microservices architecture within the retail domain, including services (e.g., *AuthService, OrderService*), infrastructural

components (e.g., *Database, ExternalAPI*), task management (*e.g., BugFix, Optimize*), team structures and members (see Fig. 5). Since the dataset is publicly available, participants can ask questions and then manually verify the responses.

Cognitive walkthrough: We conducted a group cognitive walkthrough (CW) [32] lasting 30 minutes to systematically validate GDI by reasoning through user actions and identifying potential usability issues. Since there are no end users for the dataset, we involved practitioners through purposive convenience sampling. The colleagues of the second author with work experience in ITSs and GraphRAG systems were invited via Slack and email (see Table I). The invitation detailed the study's objective, the purpose of the CW, and a short description of the user-centered validation session. We obtained the consent of the participants to record the CW.

TABLE I
PARTICIPANT ROLES AND EXPERIENCE FOR ONBOARDING USE CASE.

ID	Role	Years of Experience
1	Front-end Developer	3
2	Artificial Intelligence Engineer	3
3	Artificial Intelligence Engineer	4
4	Systems Engineer	15
5	Technical Manager	20

The goal of the CW session was to systematically validate GDI by reasoning through user actions and identifying potential usability issues. We followed the streamlined CW method, an adaptation designed to address social constraints in software companies, such as time limitations, lengthy discussions, and defensiveness [33]. The walkthrough consisted of five steps:

1) *Define inputs:* We selected the software engineer prompt template with Llama 3.1 (8B) to reflect the participants' roles. We also defined user queries related to onboarding (UQ1-UQ5) and user action sequences (see Fig.3).
2) *Convene the CW session:* GDI ran locally on a Mac M1 Pro with 32GB unified memory (200GB/s bandwidth), a 14-core GPU, and a 16-core Neural Engine. The CW session was held via Zoom, a communication platform, with screen sharing enabled, as the participants were working in a fully remote arrangement.
3) *Walk through the user queries:* Five participants validated each question by answering two questions: *"1) Will the user know what to do? 2) If the user takes the correct action, will they recognize that they are making progress towards their goal?"* If a plausible story was told for both questions, then we proceeded to the following user question. Else, we documented usability issues and identified missing knowledge required for the user to proceed.
4) *Document findings:* We documented design gaps and ideas, learnability challenges, and issues in task analysis. These findings are discussed in this section.
5) *Revise the UI:* After the CW and user-centered validation sessions, we refined our implementation of GDI to resolve the identified usability issues, which are discussed under *GDI Improvements* in this section.

The considered user queries were as follows:

$\mathbf{UQ}_1$:What are the dependencies of [SERVICE NAME]?
$\mathbf{UQ}_2$:Is there a relationship between [SERVICE NAME] and [SERVICE NAME]?
$\mathbf{UQ}_3$:Which microservices depend on [SERVICE NAME]?
$\mathbf{UQ}_4$:Which tasks are assigned to Team [TEAM NAME], and what services do they affect?
$\mathbf{UQ}_5$:Which services are maintained by Team [TEAM NAME]?

Protocol: After the CW session, participants were invited to 15-minute individual user-centered validation session to gain hands-on experience with GDI. Participants received guidance with predefined user queries and were encouraged to formulate their own questions. The logs of these sessions were saved for analysis and shared with the participants along with the CW session recording for reference during the survey. After these sessions, participants were invited to complete a 15-minute survey (refer to our replication package for the logs and survey questions [16]). We used a 5-point Likert scale due to ease of understanding for the questions related to perceived usability, perceived accuracy, and likelihood of adopting.

RQ1.a: To validate the perceived usability of GDI, participants were asked: *"How easy was it to interact with the system compared to acquiring the required information from other tools (e.g., Azure DevOps, Jira, GitHub Issues)?"*

Finding 1: The majority of participants found GDI easier to use than traditional issue tracking systems, with only one participant reporting difficulty (see Fig. 7).

P4 reported having difficulty interacting with GDI, particularly when formulating user questions. P4 expressed a preference for directly using a graph language due to its guaranteed precision and recall, and found the system inefficient for query generation. In contrast, the majority of the participants indicated that the system was easy or very easy to use compared to other tools. Fig. 7 presents the results.

We asked participants for possible improvements. P1 recommended enhancing the UI to make it more accessible for non-technical users. P5 suggested enhancing the prompting of the system. P2 provided feedback on the knowledge graph, suggesting the need for more detailed, role-specific information. P4 noted that the system (due to its knowledge graph) is not able to distinguish between internal and external dependencies. P3 highlighted that if alignment on how to structure REST-APIs is incorporated into the work items of ITSs, GDI can be used to support technical compatibility between software development teams, and contribute to seamless integration.

RQ2.a: To validate the perceived accuracy, participants were asked: *"How accurate were the responses provided by the system?"*, followed with *"If you found the responses inaccurate, misleading, or unnecessary, please provide examples."*

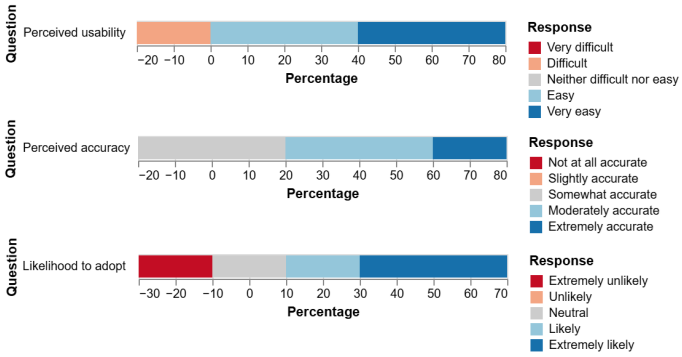

Fig. 7. Overview of perceived usability, perceived accuracy, and likelihood to adopt GDI by participants (onboarding use case).

Finding 2: For the onboarding use case, participants rated GDI's responses as follows: two participants found them somewhat accurate, two participants found them moderately accurate, and one participant found them extremely accurate.

Participants rated the accuracy of GDI as presented in Fig. 7. P3 highlighted issues related to the system's reliance on linguistic nuances, such as results changing with capitalization. P3 elaborated *"Some responses simply weren't true. Example: Which tasks are unassigned? There are no unassigned tasks."* The tasks were assigned to a team, which GDI did not infer.

P4 found the responses somewhat accurate: *"Which Microservices has External Dependency? Artificial Intelligence (AI) could not answer this question at all, as it does not understand the word "External." I.e., when using an AI, I expect flexibility in the formulations I can use as prompts so that I do not have to learn a special language to communicate with the AI."* This indicates a limitation in GDI's ability to handle flexible language inputs, something that users expect from intelligent systems. P1 rated the system responses' moderately accurate, noting that *"Queries summarizing numbers were not 100% successful."* This suggests that GDI has limitations in providing reliable results when dealing with numerical queries.

P3 highlighted two key areas for improvement: *i.* ensuring the system interprets queries correctly regardless of capitalization or syntactic features and *ii.* reducing hallucinations through domain-specific fine-tuning. P3 highly advised refining the model with Azure DevOps-related data via *"continued pretraining on domain-specific data and supervised fine-tuning, specifically for the task of query generation."* P3 also recommended implementing a validation script to review queries before execution and warning users if a generated query might fail, thereby helping them refine their questions.

RQ3.a: To validate the adequacy of the supporting information, participants were asked: *"Did the response provide sufficient supporting information to justify the response?"* If they answered no, they were prompted with *"Please explain what information was missing in the responses."*

Finding 3: Three participants indicated that the response provided sufficient supporting information to justify the answers, whereas two participants indicated that it did not.

The participants who rated GDI as somewhat accurate in Finding 2 reported that the response lacked sufficient supporting information to justify the responses. P3 pointed out *"The reasoning behind why it provided a certain response was missing."* However, the participant did not elaborate further. This suggests that P3 may have expected more detailed explanations and found that the generated query and the retrieved context were insufficient to justify the answers provided by GDI.

Similarly, P4 found that the Cypher query did not provide insight into the raw data of the underlying model. While P4 could explore the graph by running the Cypher query in Neo4j, P4 preferred a more intuitive approach, stating *"I would rather like to see a graphically presented subgraph of the underlying graph model that encompasses all the nodes and edges which are relevant for the answer."* P4 suggested integrating the visualization of the subgraph directly into GDI's UI and making the elements clickable to enable navigation to related information for enhanced usability. The other participants were able to justify how GDI generated its responses based on the generated query and the retrieved context.

RQ4.a: To validate the perceived usefulness, the participants were asked to *"Which tasks would you use this system for?"* with pre-determined answer options, and *"Are there any other tasks that you would use this system for? Please specify."*

Finding 4: Participants indicated multiple uses for GDI: onboarding and knowledge sharing, performing impact analysis, understanding context, troubleshooting and resolving work items, and optimizing resource allocation.

P3 indicated an interest in using GDI for TLR. P1 and P3 showed interest in using the system for tracking the progress and status of work across different sprints or iterations. These tasks might be less interesting as they were less relevant to their roles: four out of five participants were hands-on engineers (see Table II for an overview). However, they expressed interest in using the system to find the right contact person for issues related to specific technologies or services from other teams (P2), identifying incorrect string values in continuous integration (CI) artifacts and image specifications (P3), and team management (P5).

RQ5.a: To validate the likelihood of adopting GDI, participants were asked *"How likely are you to use this system to find answers related to the questions?*

Finding 5: Participants' likelihood of adopting the system varied, with one rating it as extremely unlikely, one as neutral, one as likely, and two as extremely likely.

Most participants expressed a neutral to strong willingness to adopt the system in their work, as presented in Fig. 7.

In contrast, P4 expressed a strong reluctance to use GDI due to proficiency in graph query languages. However, many practitioners are not proficient in these query languages, and our system enables them to benefit from data in ITSs.

GDI Improvements: We refined GDI to resolve the usability issues identified in the CW and user-centered validation sessions. We incorporated a cheat sheet that outlines user questions and guidelines to help users. We improved the UI and log mechanisms by displaying the generated query and the retrieved context separately through structured formatting to enhance readability. We also introduced buttons that allow users to copy the query and its retrieved context easily.

Based on participants' feedback, we implemented modifications to the knowledge graph. We restructured the team organization by linking tasks to individual members rather than to teams (P2). We changed the *database* and *ExternalAPI* microservices to dependencies of other microservices (P4).

The participants asked the questions PQ1–PQ5 listed below during their interaction with GDI. We used these questions as input for few-shot prompting and incorporated them into the query generation prompt template. When rerunning these queries, we observed that the previously encountered issues, including errors, had been successfully mitigated.

- $\mathbf{PQ}_1$: Which team members are working on RecommendationService?
- $\mathbf{PQ}_2$: I noticed a bug in AuthService that has something to do with Node.js, who from the Team should I contact that is familiar with that technology?
- $\mathbf{PQ}_3$: What are the most similar Tasks?
- $\mathbf{PQ}_4$: Which Microservices has External Dependency?
- $\mathbf{PQ}_5$: What are all the biggest teams?

We used the output of the onboarding use case, i.e., an improved version of GDI for the TLR use case.

V. Validation of GDI for trace link recovery

We present the dataset for the knowledge graph, the research methods used, and the results from the TLR use case.

Dataset: We built a dataset for the TLR study using the work items of VX Company, the employer of the first author. The work items and their relationships are extracted from a proprietary Azure DevOps repository called *Non-profit organization care* (NPO), which we cannot share due to confidentiality agreements. We selected NPO based on purposive convenience sampling due to its moderate size (830 nodes and 387 edges) and our access to domain experts who can validate the content of generated responses by GDI.

NPO provides non-profit organizations in South Africa with a web-based solution that enhances administration, streamlines daily operations, and ultimately improves services for beneficiaries. The system is built using a Model-View-Controller architecture, implemented in Groovy and Grails, and operates on a Tomcat server with a MariaDB database. We employ a hybrid approach to enhance performance by combining *graph indexing*, which preserves the complete graph structure for efficient graph traversal and relationship queries, with *vector indexing*, which converts data into vector representations for fast similarity-based retrieval. We define a schema for the proprietary dataset as a procedural step, as we do not propose it as a standard for other software repositories.

User-centered validation: We conducted user-centered validation sessions to give participants hands-on experience with GDI before they validated the system in the follow-up interview. We used purposive convenience sampling to select participants with work experience in ITSs and NPO (see P6–P10 in Table II) from VX Company. The participants, who are colleagues of the first author, were invited by e-mail, including the objective of the study and an overview of the session.

TABLE II
Participant Roles and Experience for TLR Use Case.

ID	Role	Years of Experience
6	Full Stack Developer	2
7	DevOps Engineer	5.5
8	Product Owner (Client)	14
9	Service Coordinator	33
10	Full Stack Developer	34

Protocol: The sessions were conducted via Microsoft Teams, which is an online communication platform used by software development teams of VX. Participants were presented with an informed consent form. If we obtained consent from the participants, we started the recording and screen sharing as participants were located in different places. The session started with an explanation of the study, its objectives, and a system demonstration, lasting 15 minutes.

If participants had no questions, we proceeded with the session, lasting 30 minutes. We used various stakeholder persona prompt templates with Llama3.1 to represent the different roles of participants. GDI ran locally on a desktop with Intel Core i7 CPU, Nvidia GeForce RTX 4080 Super (16 VRAM), and DDR5 16GB Memory.

We defined five user questions (UQ6–UQ10) with varying complexity related to TLR. The questions and guidelines were provided as a cheat sheet to help participants, who were encouraged to modify the questions to suit their needs.

- $\mathbf{UQ}_6$: Is there a relationship between Bug [ID] and User_Story [ID]?
- $\mathbf{UQ}_7$: What are the tasks of User_Story [ID]?
- $\mathbf{UQ}_8$: Which Incidents are related to Bugs?
- $\mathbf{UQ}_9$: What is the longest dependency chain in the system?
- $\mathbf{UQ}_{10}$ Which User_Stories are most similar based on their title?

Participants were instructed to think aloud to articulate their thought processes during hands-on testing. They were informed that the researchers would not intervene unless explicitly requested. The participants then followed these steps:

1) Attempt to find the answer to the user's question using Azure DevOps and record the expected answer in Notepad++.

2) Interact with the system by asking the same user question.
3) Validate whether GDI-generated response aligned with their expected answer by verifying it using Azure DevOps or Neo4j.

We chose this setup as the interaction with GDI may influence the answers given by the participants. By comparing their expected answers with the system-generated responses, we are able to establish a baseline to assess the alignment between user expectations and system responses. After the hands-on section, we held 15-minute interviews to validate GDI. The interview questions can be found in the replication package [16]). We used a 5-point Likert scale due to ease of understanding for the questions related to perceived usability, perceived accuracy, and likelihood to adopt. During the interviews, we showed participants a visual representation of the Likert scale.

Pilot: The protocol was tested in a pilot study with one participant who was not included in the final participant group. After the pilot, the participant provided feedback, noting that while the UI is user-friendly, the retrieved context should not be expanded by default as it occupies too much space. Based on this feedback, we adjusted the UI so that the retrieved context no longer expands by default. Additionally, the participant pointed out that formulating user queries was difficult due to issues with capitalization of entity names and that the system is currently unable to incorporate the previous responses as context. We leave these points for future work.

RQ1.b: To validate the perceived usability of GDI, participants were asked: *"How easy was it to interact with the system compared to acquiring the required information from other tools (e.g., Azure DevOps, Jira, GitHub Issues)?"*

Finding 6: Participants generally found GDI easy to use for answering questions compared to issue tracking systems. One participant found it neither difficult nor easy, three found it easy, and one found it very easy.

Fig. 8 summarizes the responses. P7 had no prior experience with chatbot systems and found GDI neither difficult nor easy to use, while the other participants found it easy to use. P6 found using the system easy but noted that formulating queries still involved *"trial and error, but it is easier than composing a query in DevOps."* Similarly, P9 found the system easy to use, stating *"It is easier because you do not need to know the query language, and you can ask it in your own language."*

To identify areas for improvement, we asked participants *"What aspects of the system could be improved?"*

P10 recommended taking inspiration from the UI of ChatGPT and WhatsApp. P9 recommended making it clearer when GDI is unable to generate an answer or execute a query, as it only displays an error. Finally, P8 stated that integrating the system into Azure DevOps would enhance its usability.

RQ2.b: To validate the perceived accuracy, participants were asked: *"How accurate were the responses provided by the system? If you found the responses inaccurate, misleading, or unnecessary, please provide examples."*

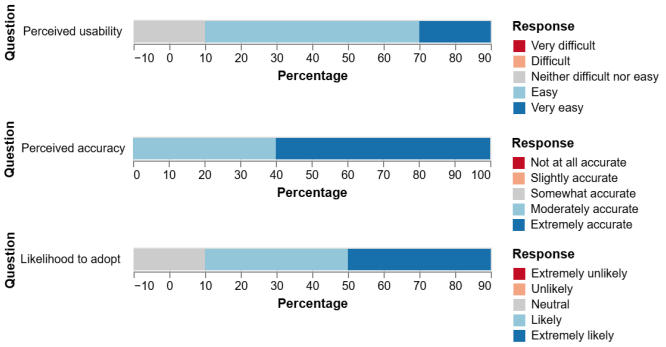

Fig. 8. Overview of perceived usability, perceived accuracy, and likelihood to adopt GDI by participants (TLR use case).

Finding 7: For the TLR use case, two participants rated GDI's responses as moderately accurate, while three found them extremely accurate.

Fig. 8 presents the answers to these questions. P6 and P7 found the responses to be moderately accurate. P8, P9, and P10 found the responses generated by GDI extremely accurate. P8 noted, *"The answers it gives are good, but there may be more answers that I have not seen, but I am not really sure, without this I would not have found these answers at all, you should mention that too."* We observed that most participants verified the results for simpler user questions (UQ6–UQ8) but relied on GDI for complex questions (UQ9 and UQ10).

RQ3.b: To validate the adequacy of supporting information, participants were asked: *"Did the response provide sufficient supporting information to justify the response?"* If they answered no, they were prompted to elaborate with *"Please explain what information was missing in the responses."*

Finding 8: All participants agreed that the responses included sufficient supporting information to justify the answers.

P6 found the queries useful, stating *"The generated queries are indeed very helpful and really good."* Additionally, P7 elaborated *"Yes, you can indeed find that with it, and in combination with the answer itself, you can understand why it gave that response."* In contrast, P9 elaborated *"I would just assume that they are the correct answers. I would not look at the supporting information; I would just go to DevOps myself to check the descriptions or comments."*

RQ4.b: To validate the perceived usefulness, we asked participants *"Which tasks would you use this system for?"* with pre-determined answer options, followed by *"Are there any other tasks that you would use this system for? Please specify."*

Finding 9: Participants would use GDI for onboarding and knowledge sharing, TLR, troubleshooting and work item resolution, understanding context, and progress tracking.

P8 and P9 stated that they would use the system for performing impact analysis. P8 expressed a willingness to use GDI to track the progress of a work item, including when it is implemented and the version of NPO in which specific bugs were fixed. All participants expressed interest in using GDI to find similar user stories, bugs, and documentation, as searching in Azure DevOps requires exact text matches, which makes it difficult to locate relevant work items. Building upon this, P9 emphasizes the value of GDI understanding the connections between the wiki, work items, and the source code.

RQ5.b: To validate the likelihood of adopting GDI, participants were asked *"How likely are you to use this system to find answers related to the questions?"*

Finding 10: The majority of participants expressed a willingness to adopt the system, with two indicating they were extremely likely to use it, two likely, and one neutral.

Fig. 8 presents the responses. P6, P9, and P10 expressed that they would use Azure DevOps to retrieve information for simpler questions but prefer GDI for complex questions, possibly due to their familiarity with Azure DevOps. P7 stated that *"I would not use it very often, but I do not frequently need to search in work items, but if it were easily accessible, I would still use it."* Finally, P8 does not find Azure DevOps user-friendly or intuitive, and GDI is easier to use.

VI. Discussion

We present the lessons learned (LL) labeled as *LL 1*, *LL 2*, and so on, in Section A. We discuss the threats to the validity and limitations in Section B.

A. Lessons learned

LL 1. GDI can improve transparency and trustworthiness by making AI responses easier to understand and trace: Trust between humans and AI is a critical concern, mainly due to the lack of transparency in AI decision-making. Research indicates that providing insights into an AI system's reasoning process can improve user trust [34]. This is important for practitioners who rely on AI for their work. Most participants were able to understand why the LLM generated a particular answer by analyzing the generated query used to query the knowledge graph and the retrieved context as a result of the query.

LL 2. To increase the adoption rate of GDI, customization and a user feedback loop are essential: By analyzing GDI's logs, we gain insights into different types of queries asked by users, which enables the refinement of prompts based on users' needs. For example, while participants in our studied context preferred short answers, this may not be the case in other settings. Additionally, implementing a mechanism to collect user feedback on the generated responses allows developers to iteratively refine GDI and improve adaptability over time.

LL 3. Humans should stay in the loop: While GDI can streamline tasks like onboarding and TLR, relying entirely on AI without human supervision introduces potential risks. In the TLR use case, most participants verified responses for simple questions but did not verify the responses for complex questions. This reliance on GDI for complex questions increases the risk of misinformation. Moreover, the use of GDI shifts the time from searching to validation, which does not necessarily reduce the total human effort. We recommend software companies train employees on how to use GDI, which could be delivered through an e-learning module or a manual.

In the onboarding use case, we noticed that participants needed support from the researchers to validate the responses of GDI as they were not familiar with the dataset. We recommend that software companies ensure that experienced team members mentor new team members to prevent them from being completely unsupervised. This guidance helps to identify potential problems early and supports ongoing learning.

LL 4. Incorporating heterogeneous data sources enhances the value of the knowledge graph and enables a greater range of use cases: The effectiveness of GDI relies on the quality and completeness of the data sources that form the knowledge graph. Participants suggested incorporating additional data, such as comments, technical alignments on how to structure REST-APIs into work items of ITSs to support compatibility between teams for seamless integration, documentation, and code into the graph to enable the system to generate responses with richer contextual understanding to resolve work items. By incorporating these data, we could use GDI to analyze the communication of software development teams in ITSs [35].

LL 5. Domain experts are necessary for knowledge graph creation: We initially attempted to create a knowledge graph from an open-source dataset; however, we encountered challenges in cleaning the data due to inconsistencies that required domain expertise. Thus, we chose to use an existing open-source knowledge graph [15]. With the proprietary data set, we did not have this issue due to access to domain experts. Domain experts can assist developers in interpreting, cleaning, structuring, and validating the data, ensuring that entities and relationships are correctly represented in the knowledge graph.

LL 6. Continuous monitoring, maintenance, data security, and access control are critical for long-term system reliability: Integrating GDI into existing workflows and ITSs of software companies is crucial for system adoption. Regular updates and snapshots are required to monitor and maintain the knowledge graph effectively. We suggest using version control such as *Git* to track changes in the CSV files. Additionally, we recommend creating a pipeline that is automatically triggered, for example, when a new user story is added. This pipeline can use a command in the preferred query language (e.g.,*"MERGE"* command in Cypher) to automatically add a new node into the knowledge graph, thereby eliminating the need to reload the entire graph. Moreover, security measures and role-based access control should be implemented in the knowledge graph and GDI to prevent any security breaches and data leaks.

B. Threats to validity and limitations

We address potential threats to validity and limitations according to the categories proposed by Wohlin et al. [36]. We also outline the measures taken to mitigate these threats.

Construct validity validates the extent to which the treatment accurately represents the theoretical construct of the cause, and the outcomes reflect the construct of the effect. The proprietary repository contains data in English and Dutch, which we retained in its original form to avoid introducing translation bias and preserve its real-world characteristics. To ensure that the proprietary knowledge graph represents the data of NPO, we perform validation checks during its formulation by randomly selecting work items and manually verifying that their representations align with the source data. We also iteratively refine the graph through discussions. Moreover, the domain experts validated the graph by querying GDI and cross-referencing the results with Azure DevOps.

Internal validity refers to the risk of uncontrolled factors that influence the observed causal relationship between treatment and outcome. The proprietary knowledge graph was created based on the authors' domain knowledge and technical expertise, which could introduce bias. To mitigate this bias, two authors who were not involved in graph creation validated the process. Additionally, domain experts validated parts of the knowledge graph.

Conclusion validity pertains to the relationship between the treatment and the outcome, ensuring that a statistically significant relationship exists. Given the novelty of GDI, we lack a baseline. To address this, we asked participants familiar with the data of NPO to first answer user queries with Azure DevOps and record their expected answers to establish a baseline. Participants then interacted with GDI and validated whether its responses aligned with their expected answers.

External validity assesses the degree to which the causal relationship between the cause and effect observed in the study can be generalized to other contexts. The representativeness of the data sets is limited by variations in issue-tracking systems, work items, relationships, and data quality. To mitigate this threat, we use both an open-source and a proprietary data set, which vary in technology, architecture, number of work items, and relationships. However, further research applying GDI to data sets with a higher number of work items is necessary to assess its scalability and impact on performance and generalize our findings. We do not claim that our findings are generalizable due to the small number of participants. However, this setup reflects real-world contexts where a small group of team members with diverse knowledge contribute to projects. Our study is limited because it only consists of a user-centered validation of GDI. However, this is an exploratory study, and we plan to extend the validation in future work.

VII. Related work

We contrast our work with existing studies. We observe that prompt engineering, RAG, and GraphRAG have been applied to the tasks of onboarding and TLR in a limited capacity.

Schuszter and Cioca [3] introduce a system that optimizes onboarding processes using an LLM with RAG, enhanced with domain-specific knowledge of the team and internal documentation. They combine chatbot development techniques for an industrial context at CERN. They encountered issues with hallucinations, emphasized the importance of cross-checking responses, and highlighted the tools' effectiveness as a starting point for new team members. These findings align with ours. However, we use GraphRAG and data from ITSs, which enables us to provide the users with supporting information to understand how the LLM queried the knowledge graph, which can help users improve trust in GDI.

Ionescu *et al.* [4] introduce the *"Onboarding Buddy"* system using LLM, RAG, and chain-of-thought prompting to improve the onboarding process of software developers by delivering dynamic, context-specific support within the development environment. While their agent-based approach demonstrates promising results in initial evaluations, it requires improvements in technical reliability and UI. Our study aims to improve onboarding across all roles within software development teams in ITSs by facilitating task-related information retrieval, extending beyond programming tasks alone.

Rodriguez *et al.* [5] explore to use of prompt engineering to predict trace links between software artifacts, demonstrating that even small modifications to prompts can substantially influence model outputs. We also observed enhanced model performance and resolution of prior issues after incorporating example questions into the query generation prompt.

The studies by Fuchß *et al.* [7] and by Ali *et al.* [6] focus on traceability from various software engineering artifacts to source code, whereas our study focuses on traceability between work items in ITSs.

Fuchß et al. [7] propose a framework that enhances LLM performance using RAG for TLR for three tasks: requirements to code, documentation to code, and documentation to architecture models. Their framework outperforms state-of-the-art methods in code-related tasks but requires performance improvements through improved retrieval techniques, aggregation methods, and code preprocessing. They also plan to incorporate the LLM's reasoning into the decision-making process of creating trace links and adding contextual information to their prompts to make their framework applicable in practice.

Ali *et al.* [6] introduce a RAG-based approach to enhance traceability between high-level use-case requirements and source code using keyword, vector, and graph indexing. While we also use vector and graph indexing, we do not use keyword-based indexing, as our knowledge graphs inherently capture structured relationships between entities, facilitating retrieval based on semantic relationships rather than textual matches. They found that their approach improves the efficiency and accuracy of establishing traceability links compared to the baseline approaches.

The study by Hey et al. [8] investigates inter-requirements traceability. They present a RAG-based approach for recovering inter-requirements traceability without the need for pre-existing links. Their approach outperforms state-of-the-art and baseline approaches. However, our scope extends beyond requirements as we cover work items within ITSs.

While the previously mentioned studies focus on TLR and use open-source data, our study uses both open-source and proprietary data and empirically validates two use cases: 1) on-

boarding and 2) TLR. These studies validate their approaches using precision, recall, F1, and F2 scores, while we validate ours empirically using user-centered metrics. As the first and second authors work in software companies, we have access to practitioners, including domain experts, to validate GDI. The previously mentioned metrics are complementary to the ground truth and help measure performance.

Finally, Abedu et al. [37] introduce *"RepoChat"*, a web-based tool designed to answer repository-related questions by combining LLMs with GraphRAG. This tool is the closest to GDI. However, there are four main differences. First, they collect data from open-source GitHub repositories, including work items, commit history, and pull requests, and use the Smart, Zero, and Zipped (SZZ) algorithm [38] to identify bug-introducing changes. This algorithm requires that issue IDs are included in commit logs, which may not always be present in proprietary projects. Second, they automatically construct a knowledge graph based on GitHub's schema. In contrast, GDI currently only contains work items. Software systems and projects in the industry may include custom work item types and attributes; thus, we need to define a separate schema. Moreover, our proprietary data set includes sensitive information, requiring pre-processing due to the GDPR and privacy concerns. We included domain experts to interpret, clean, structure, and validate the data, ensuring that entities and relationships are accurately represented in the knowledge graph. Third, the answers provided by RepoChat include assumptions and are more lengthy than GDI's. In our context, practitioners preferred shorter answers to longer ones. Lastly, they conducted a user study with four graduate students (with an average of 4 years of GitHub experience), and RepoChat answered 36 out of 40 questions correctly, achieving 90% accuracy. In contrast, we conducted user-centered validations with 10 practitioners from two companies, focusing on five criteria: perceived usability, perceived accuracy, the adequacy of supporting information, perceived usefulness, and the likelihood of adoption in the context of onboarding and TLR.

VIII. Conclusions

In this paper, we present GDI, a novel framework that uses GraphRAG and LLMs to assist software development teams in navigating work items in ITSs to streamline tasks. The main advantage of GDI is its ability to let users query work items in NL and provide supporting information that explains how the LLM queried the knowledge graph. We validate GDI through two proof-of-concept implementations for i) onboarding at itemis, and ii) trace link recovery (TLR) at VX Company.

To answer our research question on onboarding, we conducted a group cognitive walkthrough, short user-centered validation sessions, and surveys with practitioners who have work experience in ITSs and GraphRAG systems. Our findings indicate that GDI is generally perceived as user-friendly and accurate for onboarding tasks. Most participants found the system useful and expressed willingness to adopt the system, except a systems engineer who is proficient in writing queries.

To answer our research question on TLR, we conducted user-centered validation sessions and interviews with domain experts who validated the content of the responses generated by GDI. Our findings show that participants found the system easy to use and its responses accurate for TLR. Most participants found the system useful and expressed willingness to adopt the system primarily for complex questions, while preferring Azure DevOps for simple questions.

Despite the positive initial user validation, further refinement of GDI is required to deploy the system in practice. We aim to expand the knowledge graphs by integrating additional data, including comments, history, pull request links, and documentation, to provide a more comprehensive representation of the data. We plan to conduct a more comprehensive evaluation by applying GDI to larger systems and projects. We intend to collaborate with other software organizations to increase the number of participants. These collaborations will enable us to assess GDI's scalability and enhance the generalizability of our findings. Moreover, to ensure that GDI interprets queries correctly, regardless of capitalization or syntactic features, and to minimize hallucinations, we aim to optimize the prompts and fine-tune the model based on the data for query generation.

Acknowledgments

We would like to thank VX Company for providing the data used in the proprietary knowledge graph. We also extend our gratitude to the colleagues at itemis and VX Company for their valuable contributions to the user-centered validations of GDI.

References

[1] O. Habeh, F. Thekrallah, S. A. Salloum, and K. Shaalan, "Knowledge Sharing Challenges and Solutions Within Software Development Team: A Systematic Review," in *Recent Advances in Intelligent Systems and Smart Applications*, M. Al-Emran, K. Shaalan, and A. E. Hassanien, Eds. Cham: Springer International Publishing, 2021, vol. 295, pp. 121–141. [Online]. Available: https://doi.org/10.1007/978-3-030-47411-9_7

[2] R. E. de Souza Santos and P. Ralph, "A grounded theory of coordination in remote-first and hybrid software teams," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 25–35. [Online]. Available: https://doi.org/10.1145/3510003.3510105

[3] I. C. Schuszter and M. Cioca, "Increasing the Reliability of Software Systems Using a Large-Language-Model-Based Solution for Onboarding," *Inventions*, vol. 9, no. 4, 2024. [Online]. Available: https://doi.org/10.3390/inventions9040079

[4] A. C. Ionescu, S. Titov, and M. Izadi, "A Multi-agent Onboarding Assistant based on Large Language Models, Retrieval Augmented Generation, and Chain-of-Thought," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2503.23421

[5] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability ," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep. 2023, pp. 455–464. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/REW57809.2023.00087

[6] S. J. Ali, V. Naganathan, and D. Bork, "Establishing Traceability Between Natural Language Requirements and Software Artifacts by Combining RAG and LLMs," in *Conceptual Modeling*, W. Maass, H. Han, H. Yasar, and N. Multari, Eds. Cham: Springer Nature Switzerland, 2025, pp. 295–314. [Online]. Available: http://doi.org/10.1007/978-3-031-75872-0_16

[7] D. Fuchß, T. Hey, J. Keim, H. Liu, N. Ewald, T. Thirolf, and A. Koziolek, "LiSSA: Toward Generic Traceability Link Recovery Through Retrieval-Augmented Generation," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1396–1408. [Online]. Available: https://doi.org/10.1109/ICSE55347.2025.00186

[8] T. Hey, D. Fuchß, J. Keim, and A. Koziolek, "Requirements Traceability Link Recovery via Retrieval-Augmented Generation," in *2025 IEEE 31st International Working Conference on Requirements Engineering Foundation for Software Quality (REFSQ)*, 2025. [Online]. Available: https://doi.org/10.5445/IR/1000178589

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3495724.3496517

[10] Y. Zhao, H. Wang, L. Ma, Y. Liu, L. Li, and J. Grundy, "Knowledge Graphing Git Repositories: A Preliminary Study," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 599–603. [Online]. Available: http://doi.org/10.1109/SANER.2019.8668034

[11] Cooperants, "Cooperants," 2025, Accessed: 2025-03-24. [Online]. Available: https://cooperants.de/

[12] Y. Lyu, H. Cho, P. Jung, and S. Lee, "A Systematic Literature Review of Issue-Based Requirement Traceability," *IEEE Access*, vol. 11, pp. 13 334–13 348, 2023. [Online]. Available: http://doi.org/10.1109/ACCESS.2023.3242294

[13] M. Rath, D. Lo, and P. Mäder, "Analyzing requirements and traceability information to improve bug localization," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 442–453. [Online]. Available: https://doi.org/10.1145/3196398.3196415

[14] D. Falessi, J. Roll, J. L. Guo, and J. Cleland-Huang, "Leveraging Historical Associations between Requirements and Source Code to Identify Impacted Classes," *IEEE Transactions on Software Engineering*, vol. 46, no. 4, pp. 420–441, 2020. [Online]. Available: http://doi.org/10.1109/TSE.2018.2861735

[15] T. Bratanič, "Using a Knowledge Graph to Implement a DevOps RAG Application," https://gist.githubusercontent.com/tomasonjo/08dc8ba0e19d592c4c3cde40dd6abcc3/raw/e90b0c9386bf8be15b199e8ac8f83fc265a2ac57/microservices.json, 2023, Accessed: 2025-02-18.

[16] D. Ly, S. Radhakrishnan, F. B. Aydemir, and F. Dalpiaz, "GraphRAG Dialogue Insights (v1.0-updated-docs)," 7 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15804044

[17] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt Engineering in Large Language Models," in *Data Intelligence and Cognitive Informatics*, I. J. Jacob, S. Piramuthu, and P. Falkowski-Gilski, Eds. Singapore: Springer Nature Singapore, 2024, pp. 387–402. [Online]. Available: http://doi.org/10.1007/978-981-99-7962-2_30

[18] L. S. Lo, "The CLEAR path: A framework for enhancing information literacy through prompt engineering," *The Journal of Academic Librarianship*, vol. 49, no. 4, p. 102720, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0099133323000599

[19] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS)*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022. [Online]. Available: https://dl.acm.org/doi/10.5555/3600270.3601883

[20] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," in *Proceedings of the 30th Conference on Pattern Languages of Programs*, ser. PLoP '23. USA: The Hillside Group, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/3721041.3721046

[21] Microsoft, "Welcome to GraphRAG," https://microsoft.github.io/graphrag/, 2025, Accessed: 2025-02-18.

[22] J. Larson and S. Truitt, "GraphRAG: Unlocking LLM Discovery on Narrative Private Data," February 2024, Accessed: 2025-02-18. [Online]. Available: https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/

[23] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2404.16130

[24] ontotext, "What Is Graph RAG?" April 2025, Accessed: 2025-02-18. [Online]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-graph-rag/

[25] M. Hunger, "What Is GraphRAG?" https://neo4j.com/blog/genai/what-is-graphrag/, December 2024, Accessed: 2025-02-18.

[26] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2312.10997

[27] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler, "How to Build a Knowledge Graph," in *Knowledge Graphs: Methodology, Tools and Selected Use Cases*. Cham: Springer International Publishing, 2020, pp. 11–68. [Online]. Available: https://doi.org/10.1007/978-3-030-37439-6_2

[28] Streamlit, "A faster way to build and share data apps," 2025, Accessed: 2025-01-11. [Online]. Available: https://streamlit.io/

[29] LangChain, "The platform for reliable agents." 2025, Accessed: 2025-01-11. [Online]. Available: https://www.langchain.com/

[30] Ollama, "Get up and running with large language models." 2025, Accessed: 2025-01-11. [Online]. Available: https://ollama.com/

[31] ChatGPT, "ChatGPT," 2025, Accessed: 2025-01-11. [Online]. Available: https://chatgpt.com/

[32] T. Mahatody, M. Sagar, and C. Kolski, "State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions," *International Journal of Human–Computer Interaction*, vol. 26, no. 8, pp. 741–785, 2010. [Online]. Available: https://doi.org/10.1080/10447311003781409

[33] R. Spencer, "The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 353–359. [Online]. Available: https://doi.org/10.1145/332040.332456

[34] A. Kartikeya, "Examining Correlation Between Trust and Transparency with Explainable Artificial Intelligence," in *Intelligent Computing*, K. Arai, Ed. Cham: Springer International Publishing, 2022, pp. 353–358. [Online]. Available: https://doi.org/10.1007/978-3-031-10464-0_23

[35] D. Ly, M. Overeem, S. Brinkkemper, and F. Dalpiaz, "The Power of Words in Agile vs. Waterfall Development: Written Communication in Hybrid Software Teams," *Journal of Systems and Software*, vol. 219, p. 112243, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121224002875

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer, 2012, vol. 236. [Online]. Available: https://doi.org/10.1007/978-3-662-69306-3

[37] S. Abedu, L. Menneron, S. Khatoonabadi, and E. Shihab, "RepoChat: An LLM-Powered Chatbot for GitHub Repository Question-Answering," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, 2025, pp. 255–259. [Online]. Available: https://doi.org/10.1109/MSR66628.2025.00045

[38] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 641–657, 2017. [Online]. Available: http://doi.org/10.1109/TSE.2016.2616306