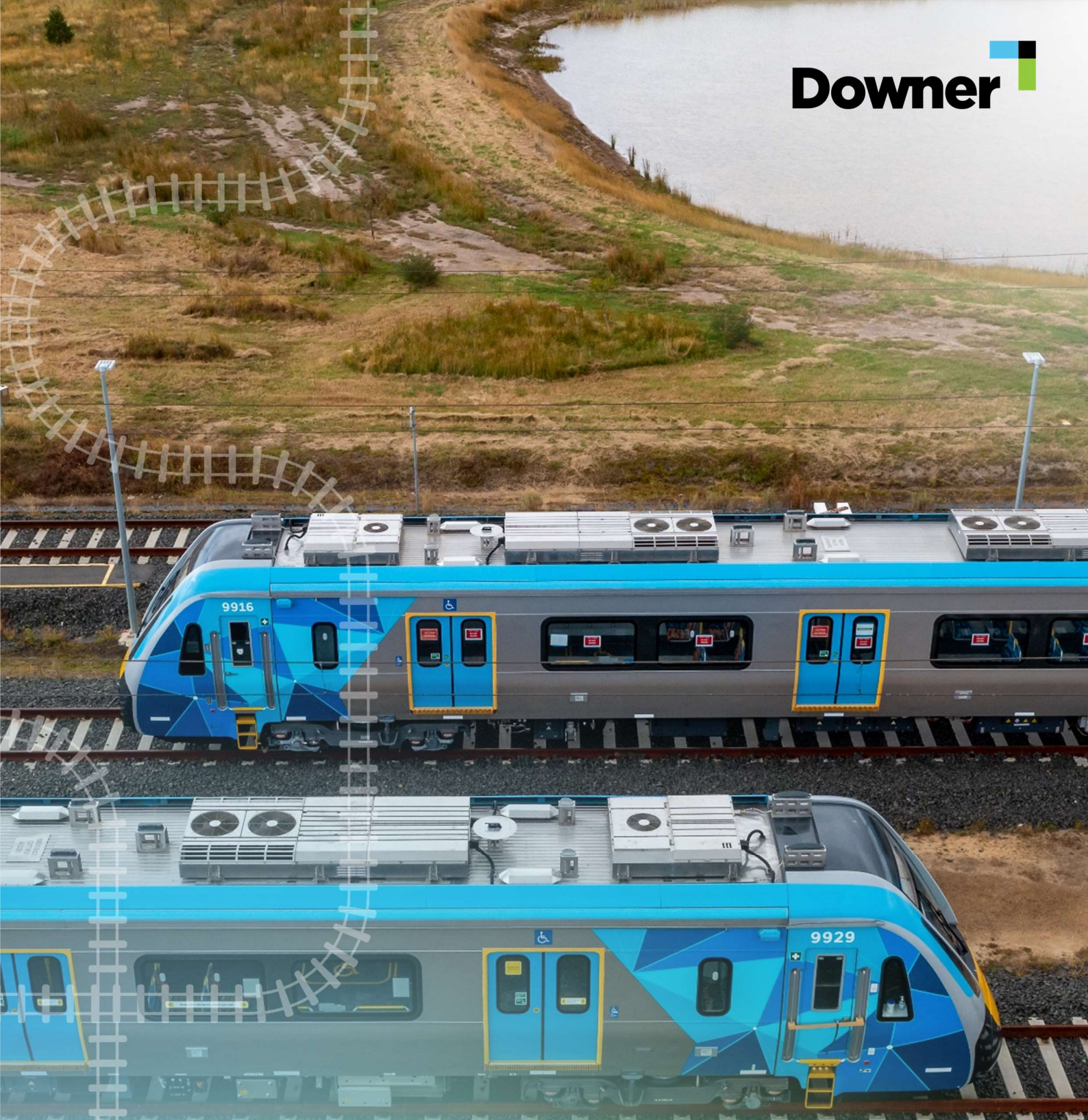

Enabling Natural Language Queries for Data Retrieval

Joseph Samuel
Downer RTS, Sydney (NSW), Australia
joseph.samuel@downergroup.com

Abstract

As we enter the era of AI, data access and data retrieval from maintenance management systems or IOT devices are becoming increasingly important for asset management, but industrial data access is often gated by specialised user interfaces (UIs) or programming (usually structured query language - SQL) expertise, requiring training and excluding non-technical users. By leveraging generative-AI's strong capabilities in writing code, we can enable workers to access data using natural language, significantly speeding up informed decision making in industry.

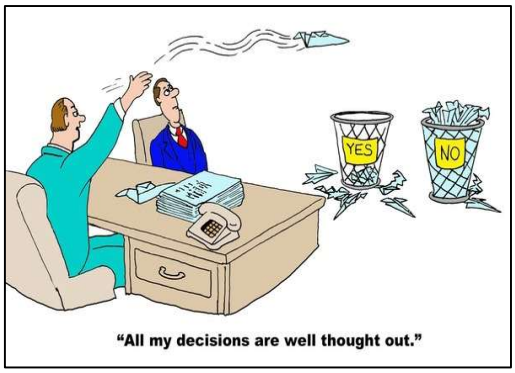

Figure 1 – Satirical depiction of making decisions without data. Image Source: (Chelsea School of Business Communication and Management, 2026)

This paper presents practical architectures – ranging from using generative-AI to retrieve pre-written queries, to generating request SQL code completely from scratch. Additionally, we will examine advantages and disadvantages of 'retrieve-and-execute' architectures versus 'retrieve-and-edit' architectures. This paper presents methods to validate retrieved data and delves into the difficulties and requirements in setting up a system with these capabilities. We will delve into what is necessary to build or set up a natural language data retrieval system in-house with and provide guidance on how to choose between different architectures.

In a rail industry case study, an implementation of a natural language data retrieval system shows early signs of success, finding significant timesaving, increased productivity and increased data retrieval capability for the average employee, allowing for better-informed decision making (e.g. queries about current stock levels, fault logs, item locations, work orders, and so on). As newer, improved models become available, the architecture used in this case study allows for easy drop-in upgrades that are expected to significantly increase complexity of requests that can be addressed, while retaining security controls.

This work demonstrates multiple practical paths for enterprises to deploy secure natural-language data retrieval at scale, accelerating decision-making across both technical and non-technical roles in asset management.

Keywords data, data retrieval, AI, artificial intelligence, natural language, SQL, guide, decision-making, CAG, RAG, Fine-tuning, generative-AI, LLM

1 Introduction

1.1 Why do we need data retrieval?

Decisions in industry are only as good as the information behind them. This is the key factor as to why data retrieval is important. If people made uninformed decisions, it would lead to financial losses, flawed conclusions and it could introduce reputational risk. (Ghorna et al., 2024)

As such, data retrieval is a necessary practice for every single person in an organisation, and every single business, whether it be retrieving information from emails, a database, a software application, or reading pdf documents.

This paper will primarily be addressing retrieving data from a database, where data is stored in tables (you can think of a database as a large excel spreadsheet), with clearly labelled rows and columns, and many sheets. When data is not stored in a database, information is often fragmented (people's

The author of this paper is affiliated with Downer EDI Limited

files, emails, spreadsheets, apps), leading to misalignment – different teams make decisions using different 'truths' and everything slows down. (Aravindhan, 2025)

1.2 What is the current state of data retrieval?

In the case of databases, data retrieval is most often done through 1. A pre-built software optimised for user interaction, or 2. Using tools like Structured Query Language (SQL), a data retrieval programming language through which users can apply filters, join tables, and do much more, to extract insights from their data. Pre-built software typically offers:

1. A custom user interface that allows users to easily drag and drop from lists of tables/fields to create visualisations.
2. Easy management, organisation and storage of data for any fields that would be required (e.g. work orders, work logs, faults, etc)
3. The ability to use SQL/programming to do data retrieval
4. API access, allowing automated data retrieval
5. Some tools are beginning to integrate AI-enabled data retrieval into their platforms. (E.g. IBM's Watsonx, Databrick's Genie, etc).

There are some other features that these companies offer, but we won't include a comprehensive list here – noteworthy is that additional features always come with additional costs when using a software product.

1.3 What are the limitations of current data retrieval tools?

These tools often come at significant cost, and often follow a licensing payment model, rather than a one-time fee – with higher fees being incurred the more features or licenses that an organisation purchases. Due to the costs, often companies will have to pick and choose which features will be effective for their applications to reduce costs, rather than having the whole suite.

Additionally, users will have to learn how to use their custom interfaces – which can often get complex. Typically, this means that employees will require training on using the software, incurring a cost in time and effort, and even then, if the users don't know how to program in SQL, sometimes it can be a complicated procedure to set up a visualisation using the user interface that could be set up in a much shorter time period using a SQL or python script. Also noteworthy is that some of this software comes with its own customised programming language/syntax, requiring significantly more effort to learn how to work with it, even for programmers.

1.4 What is AI's role in data retrieval?

Most generative-AI tools (tools like ChatGPT, Copilot, Gemini, etc) use something called a Large Language Model. Large Language Models (LLMs) are advanced AI systems built on deep neural networks, designed to process, understand and generate human-like text. (Vaswani et al., 2017)

Data retrieval primarily requires 1. An understanding of what data is available, 2. An understanding of how to program in SQL/python OR how to use a custom interface, and 3. The ability to execute this code/interact with the interface and hence retrieve the data. It has become evident over the last couple years that one of the fields that AI excels in is coding. (Chen et al., 2024) There are many ways that LLMs can be applied to generate code to respond to user queries, and as such, we will explore a few options and their advantages and disadvantages.

To build a functional AI coding assistant, we will need 3 primary things:

1. A LLM that can program in the required programming language,

2. The ability to teach LLMs the context of our data tables and environments (e.g. custom code syntax, table names, column names, context of what is in columns),

3. A feedback loop that validates the code that is produced by the LLM, or some means of ensuring that it produced code that answers the question the user asked.

Note that 3 is not strictly necessary, but without it, it is likely that users will get frustrated with the tool, particularly if they are not familiar with programming. It could also lead to incorrect conclusions being drawn from data that was not actually what the user requested.

2 Practical Architectures

2.1 Retrieving pre-written queries

In many applications, data retrieval can be broken down into several standard queries that are done often. For example, engineers may wish to check the status of a modification to the train fleet, and to do so need a SQL query that will apply a filter based on the name of the work orders. This is something that may be done frequently. Practically, there are a couple ways to implement this:

1. Present the worker with a list of pre-prepared 'reports' that they can run to fetch this data.
2. Have the worker write the code every time. Note that this can often lead to duplication of effort.

Despite clearly being better, even this first option is not necessarily a scalable approach – if there is large variety in requests, or workers have a large list of pre-written queries that they use often, then it could lead to difficulty in locating the appropriate query and make it significantly more difficult to onboard new workers. It is common that workers will require customised data retrieval requests. Not every request can be delegated to a pre-prepared query, as often it won't be worth the effort to set this up and clearly document it for a request that will only be made once. The best way to implement data retrieval is a balance of both approaches.

2.2 Using AI to write custom queries

As mentioned above, often users will require one-off queries – or even slight variations on the known 'reports' that have already been prepared. Additionally, any serious data analysis will consist completely of custom queries – and this means that any internal auditing process, cost investigation, fault investigation process, etc will often require these custom queries. This is where we can utilise AI's capabilities in writing code. If a system is set up well, the AI could additionally conduct the data analysis steps for the users, as newer AI models are very capable of assisting in this kind of work. Now I will briefly explain some of the architectures that can be used to build these AI coding assistants and explore their advantages and disadvantages.

2.2.1 Basic Implementation – Using tools like Copilot or ChatGPT

There are many places where tools like ChatGPT, Claude, Gemini or Copilot (generalised tools) can provide value, and data retrieval is definitely one of them. All organisations should attempt to use them and likely already have workers attempting to use them to assist with writing SQL or python queries (or write excel formulas) to do data retrieval or data analysis. Unfortunately, there are a number of places where these tools fall short, including data security, not knowing custom programming language syntax, not knowing a business's context (e.g. table names, column names,

jargon/abbreviations, etc) and not knowing best practices (e.g. using case-insensitive filters, knowing what data types to expect, etc). Ultimately, if you want a reliable tool that is user friendly, you will need to teach it your context. Next, we will explore some methods through which you can do this.

2.2.2 Methods for Adding Context - RAG and CAG

There are two primary methods we will examine for adding context to the AI's (LLM's) knowledge. These are Retrieval-Augmented Generation (RAG) (GeeksforGeeks, 2024) and Cache-Augmented Generation (CAG) (Chan et al., 2024). To understand these, first I will explain how we provide context to LLMs in general. There are two approaches I will mention here:

Including the context in the LLM's training data.

1. Adding the context to the LLM's system prompt/input query.

Of particular note is that adding context to the training data of the LLM is difficult. It is far beyond most companies means to train a LLM from scratch themselves, and so the primary method to include context in a LLM's training data is through a process called "Fine-tuning", where you can add to the model's training data. Again, this is a reasonably involved process due to the need to come up with appropriate question answer pairs, and as such, typically most applications will go with the second approach – adding the context to the LLM's system prompt/input query. This is the category that RAG and CAG fall under.

For those unaware of what a 'system prompt' is – you can think of it as simply appending a bunch of context to a user query. E.g. "Context" in Figure 2 is the 'system prompt'. (Paul, 2025)

Context: The work logs are within the work log table, which can be accessed using Table.Worklog. It contains the field 'Worklog.CreateDate' which contains the date that the work logs were created.

John: Please write SQL code to fetch me the work logs for the last week

Response: SELECT WORKLOG.CREATEDATE FROM TABLE.WORKLOG

Figure 2 - Example CAG request, using a basic example.

Using CAG for data retrieval is essentially just what is shown in Figure 2. We append any relevant context to the user query (by putting it into the 'system prompt') and hence the LLM has the context it requires to fulfil the request. Of course, it will need a lot more context than what is shown in Figure 2 for a data retrieval tool to function – it should have the schema, descriptions of fields, etc (See Figure 3).

```
Here are some columns you can return from workorder WO:
-- ==== WORKORDER (WO) COLUMNS ====
WO.WONUM,                   -- Unique identifier for the work order
WO.DESCRIPTION,             -- Text describing the purpose or scope of the work order
WO.STATUS,                  -- Current status code of the work order (e.g., WAPPR, APPR, INPRG, COMP, CLOSE); do not add this as a filter unless the user requests it.
WO.WORKTYPE,                -- Category of work (e.g., CM, PM, SVCREP, NCR)
```

```
## SQL Formatting Rules
Only ever use functions from the "SQL_Functions" knowledge file allowed functions list.

Do not use semicolons at the end of ORDER BY statements. Do not add semicolons at the end of the file, no matter what, otherwise you will cause an error and confuse the user.

Case-insensitive filters – For every text search, wrap both the column and the keyword in UPPER() (or LOWER()), e.g. WHERE UPPER(column) LIKE UPPER('%KEYWORD%') so every LIKE comparison
ignores case.

For text-based searches, split the phrase into individual keywords and join them with AND clauses. It also may be necessary to remove suffixes, such as "BATTERY" becomes "BATTER" to account
for the case where "BATTERIES" is listed instead of "BATTERY". For example, to find "battery control PCB," use:
WHERE UPPER(column) LIKE '%BATTER%'
  AND UPPER(column) LIKE '%CONTROL%'
  AND UPPER(column) LIKE '%PCB%'

Suggest that you can limit the number of rows returned if the query takes too long.

Always alias table names, even if they are the same name. E.g. "Maximo..MAXIMO WORKORDER WORKORDER"
---
```

Figure 3 - Example context to use for data retrieval requests. Shown is an extract of a schema, and SQL formatting rules that an AI can use to perform reliable data retrieval.

Ultimately the context you will need to provide depends on your organisation, your schema, your best practices and the programming language you use for data retrieval. This is something that needs to be built individually for each agent, and each separate database should have its own agent that understands it. It is different for each organisation, each tool and each database, and as such I am just providing guidance on what to include and not files. Also notice how they are in natural language – assembling these files is not a difficult task, it just takes some time and is helped significantly by having proper documentation of your systems.

An important thing to note is that LLMs come with some associated cost per number of tokens. Tokens correspond to words/characters (e.g. the word "hello" is 1 token) (GeeksforGeeks, 2025). The more tokens that are provided to the LLM, the higher your costs will be – essentially, the more context your LLM has, the higher the cost of running it and making data retrieval requests. This is where RAG is valuable.

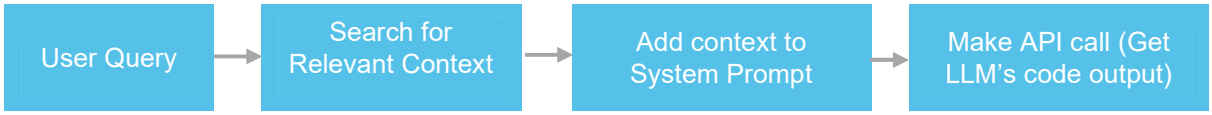

Figure 4 - The flow for implementing a RAG to do data retrieval. Note this is the same as CAG, with the addition of the second 'Search for Relevant Context' step.

RAG consists of three steps – 'Retrieve', 'Augment' and 'Generate'. (GeeksforGeeks, 2024) Essentially, it involves some form of search algorithm that is used to retrieve any relevant information to the user's request, then this retrieved data is used to 'augment' the user query (i.e. you add the retrieved information into the system prompt), and then the LLM 'generates' a response. (See Figure 4) The performance of RAG is heavily dependent on your search algorithm, and as such needs proper trialling and testing prior to providing it to users.

2.2.3 Validation

Validation is one of the most important considerations that must be made prior to giving users access to a tool. There needs to be some means of allowing the user to validate the AI response, as most users will not necessarily trust the tool (and these tools are often not 100% reliable – validation is a necessary step if you are basing business decisions on this information). This could be done by providing the user with the code, or a summary of the code that was used to do the data retrieval (this requires users to feel okay with validating by reading SQL code, or you could just provide filters/assumptions in a concise, user-friendly format for the user too.)

It is also important to consider building some form of syntax checker for your SQL code. Sometimes the AI will produce invalid code, with syntax errors, or having ignored some of the best practices it was told to follow. For this form of error, these are often easily found using a code syntax checker of some sort. Rather than sending incompatible/incorrect code to the database, it is better to check the code automatically and rebuild it prior to data retrieval, by providing clear error messages for different error types that the AI can use to rebuild/fix the code.

Note that the 'Retrieve-and-execute' architecture mentioned in the abstract consists of 'executing' the code – i.e. doing the data retrieval, while 'Retrieve-and-edit' involves just providing the user with the code and having them run it. This can be a viable alternative if data connections are difficult, but requires the user to interact with the code, meaning that it would likely mean that users require some basic training to learn how to use it – thus a 'Retrieve-and-execute' architecture is better if it is reliable and its results can be easily validated.

2.3 Case Study – TrainDNA Coder

At Downer, we have implemented a data retrieval tool 'TrainDNA Coder' using a retrieve-and-edit architecture, which allows the users to validate the code themselves. The tool has been adopted and users are using it – although many of the difficulties/trade-offs I mentioned in this paper have come up as considerations or issues throughout our work building this tool and ensuring it addressed the users' needs. Users who formerly were not capable of doing data retrieval now have a viable path to doing this, significantly increasing their capability to make data-driven decisions without leaning on technical experts that know how to do data retrieval or program in SQL.

'TrainDNA Coder' has been implemented in a way such that we can easily switch out our models for better ones (e.g. upgrade from GPT-4 to GPT-5) and can be easily improved/updated if there are changes in schema or additional abbreviations/jargon that is required for it to deal with. There are also potential plans to upgrade to a retrieve-and-execute architecture at some point, but this is still in development/testing.

3 Closing Remarks

To conclude - if a reliable data retrieval tool can be built, it does not just benefit users directly but also can act as a foundation for other tools. Suddenly, any other tool can be connected to it and have data retrieval at their fingertips – any tool can query your database and retrieve the relevant data, and as a result, make informed decisions, or gather the much-needed context or information they need to do what the tool was designed to do. It is important to realise that a tool like this is not simply something that allows users to work more productively, require less training/onboarding and make informed decisions – it is the foundation of your organisation's AI journey.

Conversely, it is important to realise that it should not always be used as a means of data retrieval. These tools will often not provide the same response if you ask the same query twice, and as such, for tools like dashboards – data that is accessed frequently, a tool like this could help with initial development, but not be a reliable long-term solution. For static tools, static data retrieval should be used (Just use the same code every time). For dynamic tools with wide breadth in capabilities, or quick queries, AI-enabled data retrieval provides immense value.

Finally, one of the primary gaps that is emerging in implementation of AI in the enterprise is a lack of connection of data sources. Many companies are providing solutions to this, e.g. Databricks, Microsoft Azure, etc, but this often comes at significant work and expense. These connections are something that are going to act as roadblocks regardless of how much you pay an organisation for their software. There is no question that every organisation that wants to implement any AI at scale will need to implement connections to their data sources that an AI can access – whether that be via an API, or some other means. (Green et al., 2025) They will need clear documentation of these connections and the data sources themselves so that the AIs can understand when and how to access these data sources – both of which are pre-requisites for building a tool for AI-enabled data retrieval.

4 References

Aravindhan, M. (2025). Breaking Down Data Silos: How AI 'Builds Bridges' in the Cloud. *European Journal of Computer Science and Information Technology*, [online] 13(48), pp.177–187. doi:https://doi.org/10.37745/ejcsit.2013/vol13n48177187.

Chan, B.J., Chen, C.-T., Cheng, J.-H. and Huang, H.-H. (2024). *Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks*. [online] arXiv.org. Available at: https://arxiv.org/abs/2412.15605.

Chelsea School of Business Communication and Management. (2026). Can you make an unbiased decision? [online] Available at: https://chelseaschool.net/article/can-you-make-an-unbiased-decision/.

Chen, L., Guo, Q., Jia, H., Zeng, Z., Wang, X., Xu, Y., Wu, J., Wang, Y., Gao, Q., Wang, J., Ye, W. and Zhang, S. (2024). *A Survey on Evaluating Large Language Models in Code Generation Tasks*. [online] arXiv.org. Available at: https://arxiv.org/abs/2408.16498?utm_source=chatgpt.com [Accessed 25 Feb. 2026].

GeeksforGeeks (2024). *What is RetrievalAugmented Generation (RAG) ?* [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/nlp/what-is-retrieval-augmented-generation-rag/.

GeeksforGeeks (2025). *Tokens and Context Windows in LLMs*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/artificial-intelligence/tokens-and-context-windows-in-llms/.

Ghorna, L., Elly, A., Oladele, S. and Ahsun, A. (2024). *The Importance of Data Quality in Data-Driven Decision- Making*. [online] Available at: https://www.researchgate.net/publication/387366223_The_Importance_of_Data_Quality_in_Data-Driven_Decision-_Making [Accessed 24 Feb. 2026].

Green, A., Makan, A., Sahgal, V., Terry, M. and Booth, T. (2025). *Mastering the data challenge in AI Insights from global CIOs and technologists Mastering the data challenge in AI 2*. [online] Economist Impact. Available at: https://www.databricks.com/sites/default/files/2025-07/mastering-the-data-challenge-in-ai.pdf?itm_source=www&itm_category=resources&itm_page=thank-you&itm_location=body&itm_component=hero&itm_offer=mastering-the-data-challenge-in-ai.pdf [Accessed 25 Feb. 2026].

Paul, D. (2025). *System Prompts Explained: How AI Models Actually Work Behind the Scenes*. [online] Medium. Available at: https://medium.com/@david.p.lemon79/system-prompts-explained-how-ai-models-actually-work-behind-the-scenes-2265f14e3eba.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł. and Polosukhin, I. (2017). *Attention Is All You Need*. [online] Available at: https://arxiv.org/pdf/1706.03762.

Note: AI-assisted writing tools were **not** used in the writing of this paper (e.g. chatgpt, gemini, etc). It has been written and prepared wholly and completely by the author. Although the paper discusses AI tools and provides insights gained through working with them, no AI-assisted writing tools were used in its composition.


Downer