
# Assessing Inferencing Capabilities of Generative AI

**A dissertation submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science in The Queen's University of Belfast by**

**Matthew James Stewart**

**2025-04-28**

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

## CSC3002 – COMPUTER SCIENCE PROJECT

## Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Matthew Stewart Student Number: 40332822
Project Title: Assessing Inferencing Capabilities of Generative AI

Supervisor: Professor Austen Rainer

## Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook.
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic).
3. Does not exceed the specified page limit.
4. Is clearly presented and proof-read.
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**By submitting your dissertation you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.**

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

**Student's signature:** Matthew Stewart **Date of submission:** 28/04/2025

QUEEN'S UNIVERSITY BELFAST

## Acknowledgements

First of all, I would like to express my sincere gratitude to Professor Austen Rainer for his invaluable assistance and guidance throughout this project. His insightful feedback, expertise, and continued support were instrumental in shaping this work [1].

## Abstract

The recent rise in popularity of Generative AI models, specifically Large Language Models (LLMs) has been unprecedented. Despite their impressive capabilities, studies show that these models often struggle to maintain coherence during tasks that require navigation through complex inferencing chains. This limitation is evident in tasks such as inferring implicit character relationships from narrative text. To explore this challenge we developed a system that uses a systematic framework to evaluate LLM performance on inferential tasks. Our empirical results align with previous research, confirming clear shortcomings in the ability of current LLMs to solve complex inferences.

# Contents

# 1 Introduction

Large Language Models (LLMs) have shown excellent performance on various benchmarks such as the Multilingual Grade School Math (MGSM) [2], Massive Multitask Language Understanding (MMLU) [3] and HumanEval [4] benchmarks, to name a few. However, recent studies have shown how these models fall short when tested on their general reasoning skills [5]. Considering that reasoning involves deriving logical conclusions from given information a process known as inference [6], these concerning shortcomings have created the need for evaluation frameworks for researchers and developers to to assess the inferencing capabilities of these novel language technologies.

This study will focus on assessing this foundational aspect of reasoning in LLMs. Our work is Inspired by the CLUTRR[7] and GSM-Symbolic[8] research, it proposes a software solution that can be used to assess LLMs capabilities to perform successful inferences when given a narrative text and a question based on that text.

Similarly to CLUTRR, our custom benchmark uses a dataset of narrative texts involving various family relationships. Given the narrative text, to perform a successful inference the LLM must infer the relationship between two characters in the text whose relationship is not explicitly stated. To test model robustness the system can also systematically vary the narrative texts by dynamically randomising variables in the text during each evaluation run. This ensures the narrative maintains the same structure and inferencing chain for testing, while introducing new irrelevant facts which are known to distract LLMs [9]. To complement our custom benchmark we also added functionality to test models against the open source CLUTRR dataset [10].

Following evaluation we assess the performance of several modern LLMs, Llama 3.1 (8B), Llama 3.1 (70B) Llama 3.3 (70B) and Mistral(7B) against each other and against the results shown in the CLUTRR research paper.

# 2 Problem Description

## 2.1 Inferencing in Generative AI

Inferencing is often referred to as a fundamental step in logical reasoning [6]. The ability to perform a successful inference is to examine text and derive conclusions that are not explicitly stated within that text. For example in our study we draw from the CLUTRR benchmark [11], which displays inferencing as the ability to navigate through relationships between characters in narrative text, keep track of the relationships between the characters and then use the relationships identified to reach a conclusion.

A simple example illustrates this, if an LLM reads that "Alice is Bob's mother" and "Jim is Alice's father," inferencing capability would enable it to infer that "Jim is Bob's grandfather" without having seen this relationship explicitly stated. Studies have shown, LLMs such as BERT-LSTM & BiLSTM Attention, typically achieve performance ranging from 75 - 85% accuracy on such straight forward inferencing tasks [7].

However, performance significantly declines when these LLMs face more complex inferencing challenges. For example, Given a short story with the following relationships, "Harry's sister in law is Morgan","Harry's daughter is Isabel", "Joys daughter is Isabel", "Joy is Scott's aunt", "Kevin's daughter is Valerie", "Valerie's sister is Melissa", "Danielle's parents are Dale and Morgan" and "Danielle's Sister is Ouida," for instance if the LLM is then challenged with determining Ouida's relationship to Kevin (granddaughter). Inferencing performance ranges from 35 - 40% [7].

This performance gap between simple and complex inferencing tasks reveals a fundamental limitation in the LLMs tested in this evaluation. While these models excel at straightforward inferencing tasks, their ability to remain coherent during multi-step inferencing chains is mediocre. As we will explore in the following sections, these limitations could potentially stem from the fundamental architecture and training paradigms they are created with.

## 2.2 Generative AI Inferencing Limitations

Although LLMs have shown strong performance on a number of natural language understanding tests [12], recent studies have shown that they experience limitations when it comes to reasoning, which is a fundamental aspect of human learning but a recurring difficulty for LLMs [13]. Concerns regarding LLMs tendencies to take advantage of patterns in data rather than showcasing actual reasoning abilities are becoming more prevalent [14]. According to [8], LLMs show an evident variation in their responses to prompts based on the same narrative, which raises the suspicion that these models are depending more on matching patterns rather than their capacity for reasoning. Given that the ability to reason relies heavily on the ability to perform an inference [15], these concerns raise questions about the reliability of LLM inferencing capabilities.

## 2.3 How Fundamentals of Generative AI Affects Inferencing Capabilities

The architectural foundations of these models could have a direct influence on their inferencing capabilities, Large language models (LLMs) also known as Transformer neural networks are used by auto-regressive GPT style models that aim to solve sequence-to-sequence tasks, first presented in the paper [16], they are now considered the state of the art technique for Natural Language Processing.

Using encoder-decoder architecture based on attention layers, the encoder converts the input into context vectors, which the decoder then uses to generate the output using auto-regressive language modeling. Which means it generates one output “token” (unit of text or a word) at a time, repeatedly using the “context” of the prompt and the new tokens generated thus far in its response, to produce the next best token [17].

Transformer models use a mechanism called self-attention which allow the model to assign weights to key words within a sequence, it focuses on how relevant a particular word is with respect to other words in the sequence. In Table 1 the input prompt “Write me a blog post about going to a football match” provides the model with a task, then based on its prediction the model produces the first token “The”, then when the model generates the following token “anticipation”, it does so using the context of the prompt and the previously generated token “The”. This continues until the model hits

an end of sequence token which signals it to stop producing tokens [18].

| INPUT | NEXT TOKEN |
|---|---|
| "Write me a blog post about going to a football match " | "The" |
| "Write me a blog post about going to a football match The" | "anticipation" |
| "Write me a blog post about going to a football match The anticipation" | "builds" |
| "Write me a blog post about going to a football match The anticipation builds" | "as" |
| "Write me a blog post about going to a football match The anticipation builds as" | "I" |
| . . . and so on, until the model stops producing tokens | |

Table 1: Example of auto-regressive text generation. The input for the generation of the first output token is the prompt alone.

While this self-attention mechanism creates well organised text by capturing contextual relationships between elements in the text [17]. It is important for us to test if these models are mimicking inferencing based on the patterns they have seen in their training data, as they inference well in many scenarios, although still produce hallucinations that appear nonsensical to the human eye [14], showing cracks in their inferencing abilities. Common hallucinations include responding to prompts with incorrect facts, deviation from the input prompt or adding unrelated information to the output [19].

## 2.4 Expected Users

### 2.4.1 Researchers

Systems that provide evaluation frameworks will help researchers examining the strengths and weaknesses of LLMs to methodically evaluate inferencing capabilities of LLMs. Standardised benchmarks provided by this project [11] as well as the freedom to create custom evaluation datasets tailored for particular research requirements help this user group. Through creating their own custom datasets, researchers can focus on specific limitations of inferencing performance or handle concerns that are not addressed by current benchmarks. This allows for a more nuanced investigation into LLM behaviour.

### 2.4.2 Developers

To ensure their systems can meet the inferencing requirements of real-world use cases, particularly when crucial tasks are involved, developers integrating LLMs into systems require evaluation frameworks to benchmark LLM performances. The main advantage of these frameworks for developers is that industry standard benchmarks like CLUTRR [11] offer reliable evaluations of LLM performance. With the help of these benchmarks, developers can easily evaluate models, choose the best one for their system and identify any potential limitations before deployment.

### 2.4.3 General Users and Enthusiasts

Beyond researchers and developers, there is an ever growing community of AI enthusiasts and students experimenting with creating and fine-tuning LLMs. For this user group systems that offer evaluation frameworks are essential. They too can benefit from industry-standard benchmarks such as CLUTRR [11] to test their LLMs capabilities.

# 3 Solution Description

The main focus of this project is to address the need for a software solution that systematically evaluates Large Language Models (LLMs) inferencing capabilities across various complexity levels. Our built in evaluation framework implements both custom and industry standard relational inferencing benchmarks, to assess how models handle multi-step inferencing chains. The software solution systematically prompts LLMs with a set of inferencing tasks, collects their responses and generates quantitative metrics to visualise how the they handle different levels of inferencing chains. Performances can be gathered and compared across various model architectures and parameter sizes to reveal insights about how model complexities correlate to inferencing capabilities.

## 3.1 System Requirements

### 3.1.1 Model Selection & Interaction With Gen AI API's

The system should allow users to explore how various Large Language Models (LLMs) respond through an interactive model playground. Using models provided by Cloudflare Workers AI API [20] and Hugging Face serverless inference API [21]. The system should return responses within

30 seconds and be able to handle API failures by displaying relevant error messages.

This interface should offer a user friendly, easily navigable chat environment where users can send one-shot prompts to a variety of LLMs and receive timely responses to get a feel for how they respond, how intelligent they are and to determine suitability for further experimentation.

### 3.1.2 Dataset Creation

The system should provide an interface for creating and managing datasets to evaluate the relational inferencing abilities of large language models. Users should be able to design contexts (also know as narrative texts), questions and ground truths related to these contexts to assess LLM inferencing capabilities.

Users should be able to add variation to their stories by setting placeholders enclosed within curly braces (i.e. {male_name}), these can be set across the context, question and groundtruth to enable the systematic variation of the input prompts used during evaluation runs.

To systematically explore how models handle increasing complexity, users should be able assign a difficulty level ('k-value') to their datasets, with no restrictions on the range of values. The system should not limit the size of user created datasets. For users needing standardised benchmarks, the application also provides use of the well-regarded CLUTRR benchmark for relational reasoning tasks [7] as a read only predefined dataset.

### 3.1.3 Prompt Engineering techniques

The system should use a predefined base prompt template for all evaluation frameworks. This template must follow industry standard prompt engineering techniques, including:

1. A clear task description that explains what the LLM needs to do
2. Specific instructions for how the LLM should process the information
3. An example response format

During evaluation runs, the system must automatically inject the context and question into the prompt template before sending it to the LLM. The template must be compatible with all models

available on the system.

### 3.1.4 Large Language Model Evaluation

The system should allow users to carry out LLM evaluations through two complementary frameworks. An evaluation using their own custom dataset, where users can select which LLM and dataset difficulty value to evaluate and an evaluation using the predefined CLUTRR dataset, with the same LLM selection and difficulty selection options.

The system should process the evaluations and display results using visualisations upon completion including, accuracy scores to track overall LLM performance, box plots to visualise the distribution of LLM response performance (custom evaluation) and response time values for the LLM (CLUTRR evaluation).

Users should be able to save custom evaluation results to the database for future reference and save CLUTRR evaluation results directly to their device. The system should handle evaluation runs with at least 100 dataset entries within a reasonable time frame and provide intermittent progress indicators.

# 4 Design

## 4.1 Architecture Overview

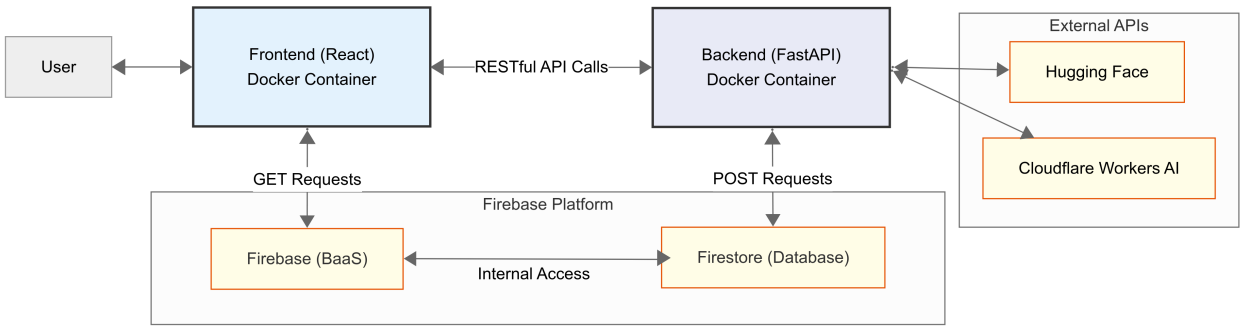



Figure 1: System Architecture

In Figure 1 the architecture of the system is shown with the React front-end which runs in a docker [22] container, The frontend communicates with the FastAPI backend (also containerised) via RESTful API calls. The backend interacts with external API's (Hugging Face and Cloudflare Workers AI for LLM interactions) and persists data to the Firestore database via POST requests. The front-end uses the Firebase [23] Client SDK to make direct GET requests to Firebase services (Firestore) [24], utilising Firebase's Backend-as-a-Service (BaaS) features.

## 4.2 Software System Design

This section details the role of each component and the interfaces between components.

### 4.2.1 Front-end Components

This section contains detailed descriptions of the front-ends key components. A Table containing additional front-end components is available in Appendix B.

#### App

The App component is the central management module for the entire application. Its main duties include, managing global state of the current view, sidebar visibility, and selected model. It implements a view-switching mechanism that renders different page components based on user navigation. This component ties the whole application together while maintaining a clean layer of abstraction.

#### Playground

The Playground component is an interactive chat box for direct interaction with LLMs. It offers a conversational interface that separates user input from model output clearly. It enables users to create and send prompts from a text box and immediately view model responses in the conversational interface. Various tabs can be open at once to allow users to manage multiple prompts simultaneously, supporting comparison between different model responses. A key feature is its real-time feedback loop, users can see the model processing their input prompt while they wait. The playground handles the complete life cycle of prompt submission, including loading states during processing and error handling for failed requests. Its focused design makes it ideal for prompt engineering, allowing users to quickly iterate on prompt wording and structure to get optimal model

responses. As the most direct interface to the externally hosted LLMs, the playground component represents the interactive experience of the application.

**Evaluation Suite**

The Evaluation Suite component is one of the applications interfaces for testing LLMs. In this suite users can run evaluations on their custom datasets and visualise how variables are substituted and randomised during evaluation runs. What makes this component useful is its ability to compare model responses against expected ground truths using various similarity metrics. The component maintains state between context, question and ground truth variables, ensuring that variable substitutions are applied uniformly before the data is injected into the prompt template. Batch evaluation is used to enable large scale testing using custom datasets, this makes it invaluable for systematic model evaluation.

**CLUTRR Evaluation Page**

The CLUTRR Evaluation component is used for evaluating models using the CLUTRR (Compositional Language Understanding and Text-based Relational Reasoning) dataset. It allows users to select from predefined CLUTRR datasets or upload their own generated CLUTRR datasets in CSV format. Once a dataset is selected its processed and the relevant information (story & query) is injected into the prompt template, which is then sent to the LLM to test relational reasoning capabilities. Once evaluation is complete, visual results of the model's performance is presented, breaking down results by accuracy and relation type.

### 4.2.2 Backend Components

This section contains detailed descriptions of the key backend components.

**App**

The App component defines all API endpoints that the frontend interacts with, including routes for context and question creation, model evaluation, data visualisation and interaction with LLMs. This component also interacts with Firestore for data persistence. Key features of this component include the ability to submit prompts to various LLMs, calculate similarity scores between responses and ground truths and generate visual results. The app also implements proper error handling,

logging, and CORS middleware to ensure a solid API service. This component is essential as it ties together all other parts of the system.

**Model Evaluation**

The Model Evaluation component is responsible for evaluating the quality of model responses using a variety of similarity metrics and exact match techniques. It uses pre-trained models from the sentence transformers library to compute semantic similarity scores between LLM generated responses and ground truths. The supported metrics include S-BERT sentence embeddings [25], STS RoBERTa embeddings [26], and ROUGE [27] scores for text comparison. This component is fundamental to the application's core purpose of quantitatively assessing model performance. It provides a way to compare different LLM outputs against reference ground truths and format structures, helping users to understand model capabilities and limitations. The approach to use multiple metrics gives a more nuanced evaluation than any single metric could provide.

**Cloudflare API**

This component manages interactions with the Cloudflare Workers AI API, it allows the application to receive responses from LLMs hosted on Cloudflares infrastructure. It manages the model mapping between internal identifiers (cloudflare-llama-3.1-70b) and Cloudflare's model names (@cf/meta/llama-3.1-70b-instruct). It also handles errors and manages LLM outputs appropriately. This component is essential as it provides a wide selection of Llama models such as Llama 3.1 (70B) and 3.3 (70B). The design of this component also makes adding new models simple.

**Hugging Face API**

This component manages interactions with the Hugging Face's API, enabling the application to receive responses from various large language models. It creates pipelines for different LLMs using haystack [28] and handles the communication with Hugging Face's serverless inference API. This component includes error handling, logging, and proper API credential management through environment variables. This key component offers hundreds of models [21] and it serves as an important fallback in case of rate limiting or availability issues with the Cloudflare API.

Table 2 lists all additional backend components with brief descriptions of each.

| Component | Description |
|---|---|
| database.py | Contains database operations for Firebase integration |
| Dockerfile | Defines the containerisation configuration for the application |
| test_firestore.py | Script to test Firestore database connection |
| tests/ | Contains unit and integration tests for the backend |

Table 2: Additional Backend Components

#### 4.2.3 Interface Design

**Front-end to Backend Communication**

The front-end interacts with the backend through RESTful API calls. Each front-end component requests data from the appropriate backend endpoint. The front-end uses Fetch API [29] to provide an interface to make HTTP requests to backend endpoints. An example of this is when the front-end sends a POST request to the backend endpoint that handles sending prompts to LLMs, this endpoint processes the request by formatting the prompt data, routing it to the appropriate LLM specified in the request and returning the LLM generated response. Once the front-end receives the response it presents the data to the user in relevant front-end interface.

**Backend to External API Communication**

The backend communicates with external LLM providers through their respective APIs, handling authentication, request formatting, and response processing. When a user submits a prompt with a selected model (e.g., "cloudflare-llama-3.1-8B"), the front-end sends a POST request to the `/api/submit_prompts` endpoint in the backend, which then identifies the appropriate provider based on the model ID prefix, maps internal model identifiers to provider-specific model names (e.g., "@cf/meta/llama-3-8b-instruct"), constructs the appropriate request format, makes the API call using the Python requests library, handles any errors or timeouts that occur during the communication process, extracts the generated text and finally returns a response to the frontend while keeping API keys and authentication details secure on the server side.

**Front-end & Backend to Database Communication**

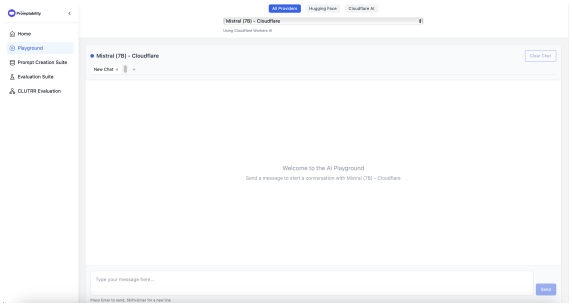
Communication between the front-end and the Firestore database is carried out using the Firebase Client SDK. The front-end can perform simple read only database operations (fetching contexts and questions). During the data fetching process, front-end components use hooks such as `useDatabase()` which utilise Firebase's (Backend-as-a-Service) client-side functions (`collection`, `getDocs`, `addDoc`, etc.) allowing them to interact with Firestore collections. For example, this process is carried out when loading contexts, the front-end executes `getDocs(collection(db, 'contexts'))` instead of making HTTP requests to backend endpoints, this allows for real-time data synchronisation, a reduced backend load and the ability to use Firebase's built-in authentication and security rules directly from the client.

The backend communicates with the Firestore database through the Firebase Admin SDK, providing an abstraction layer for database Write operations (creating contexts and questions). For example, when a client submits a POST request to the `/api/contexts` endpoint, the backend first validates the request body against the `ContextCreate` model using Pydantic, ensuring all required fields are present and properly formatted. After validation, the backend adds timestamps to the context (`createdAt` and `updatedAt`), then establishes a connection to Firestore through the Firebase Admin SDK and creates a new document with an auto-generated ID in the 'contexts' collection.

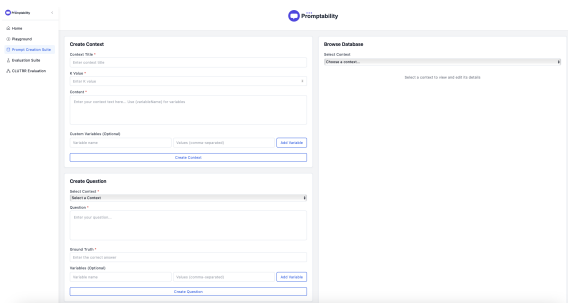
### 4.3 User Interface Design

The user interface follows a simple minimalist design that was easy to replicate across all components. The landing page provides brief details of the purpose of the system and each component, it also allows users to directly navigate to the desired component in one click. The landing page is viewable in Appendix A.

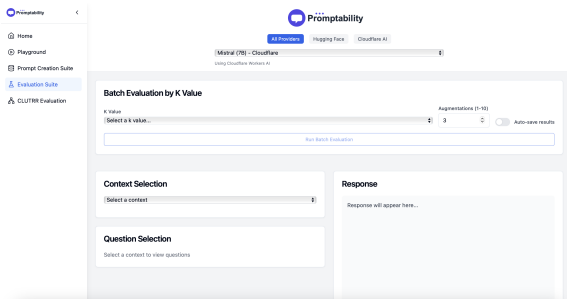
The user interfaces for our four key components are displayed in Figure 2. After implementing the initial playground interface, the boilerplate for this was used across each of the remaining components with variations where necessary, this allowed for re-usability and sped up the development process. Throughout all four components a side bar is provided to simplify navigation through each components of the system and a header is present for selecting which LLM to use.



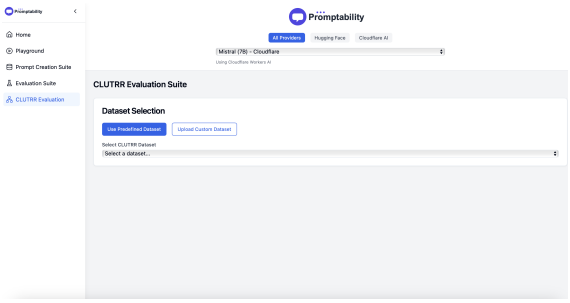
(a) Playground Suite



(b) Prompt Creation Suite




(c) Custom Evalution Suite




(d) CLUTRR Evaluation Suite

Figure 2: Wireframes for the UI of the Assessor application

Figure 3 shows the colour scheme our application uses. White is used throughout for background elements, blue's are used in the logo, for buttons and to highlight selections and black and grey are mainly used for text.


Figure 3: Application Colour Palette

## 4.4 Key Design Decisions

### 4.4.1 Large Language Model Selection

This study uses a variety of instruction-tuned large language models (LLMs) mainly accessed through Cloudflare Workers AI [20] with the option of Hugging Face [30] as a fail-over to address fault tolerance concerns. The selected models represent various architectures and parameter sizes (amount of layers and/or connections between neurons) to provide a comprehensive evaluation of current LLM inferencing capabilities:

**Llama-3.1-8b-instruct:** Llama-3.1-8b-instruct is a compact but highly optimised LLM in the Llama series. With 8 billion parameters, it demonstrates what can be achieved with a constrained model size while implementing advanced architecture and training techniques. Despite its smaller parameter count, it still offers reasonable performance and serves as an important baseline for understanding how inferencing capabilities scale with model size [31].

**Mistral-7b-instruct-v0.2:** The Mistral-7B-Instruct-v0.2 model demonstrates good performance within the 7 billion parameter scale. This version demonstrates better instruction-following capabilities and overall improved performance compared to its predecessor (v0.1). This model uses grouped-query attention and sliding window attention mechanisms to improve computational efficiency while maintaining strong performance [32].

**Llama-3.1-70b-instruct:** As one of the largest publicly available open-source models, Llama-3.1-70B With 70 billion parameters allows for testing how increasing parameter size impacts inferencing capabilities compared to its smaller counterparts. This model provides insights into the performance of current publicly available open-source models [31].

**Llama-3.3-70b-instruct:** Meta's most recent state-of-the-art 70 billion parameter model, Llama-3.3-70b approaches the performance of much larger models such as the 405 billion parameter variants in the Llama 3.1 series. This model is the cutting edge of publicly available language models and incorporates the latest advances in training methodologies, architecture, and optimisation

techniques [33].

**Rationale for Model Selection:** The selection of these specific models provides several advantages for evaluating inferencing capabilities:

- **Parameter Scale Comparison:** Including models ranging from 7B to 70B parameters allows us to assess of how inferencing capabilities scale with model size.
- **Architectural Diversity:** The models represent different architectural approaches (Llama and Mistral architectures), allowing comparison of how different designs impact inferencing.
- **Training Methodology Variance:** Each model series was trained using different methodologies and datasets, providing insights into how training approaches affect inferencing abilities.
- **Range of Current Models:** The selection spans from established models (Llama-3.1) to the most recent releases (Llama-3.3), enabling evaluation of how advancements in LLM technology translate to inferencing improvements.
- **Open-Source Accessibility:** All selected models are open source, so results can be extended by other researchers.

### 4.4.2 Data Model Design

The project uses Firestore as its NoSQL database, for storing custom evaluation datasets that can be created manually through the application, the data models are appropriate for the applications needs allowing for flexible schema evolution if user needs change. The following collections are stored in the Fire Store database:

**Context**

Table 3 displays the data model for contexts, these contexts we refer to are narrative text templates with variable placeholders to enable the variation of prompts.

| Field | Type | Description | Constraints |
|---|---|---|---|
| _id | string | Firebase document ID | Auto-generated |
| title | string | Title of the short story | Required, non-empty |
| content | string | Template text with variables | Required, non-empty |
| k | integer | Complexity value for evaluation (greater value indicates more difficult task) | Required, positive integer |
| variables | object | Map of variable names to value arrays | Optional |
| createdAt | timestamp | Creation time | Auto-generated |
| updatedAt | timestamp | Last update time | Auto-updated |

Table 3: Context Schema

### Question

Table 4 displays the data model for questions, the questions are related to contexts by a one to one relationship, they are used for testing LLM inferencing capabilities based on the narrative text.

| Field | Type | Description | Constraints |
|---|---|---|---|
| _id | string | Firebase document ID | Auto-generated |
| contextId | string | Reference to parent context | Required, valid context ID |
| q | string | Question text with variables | Required, non-empty |
| groundTruth | string | Expected answer for evaluation | Required, non-empty |
| variables | object | Map of variable names to value arrays | Optional |
| createdAt | timestamp | Creation time | Auto-generated |
| updatedAt | timestamp | Last update time | Auto-updated |

Table 4: Question Schema

### Evaluation Result

Table 5 displays the data model for evaluation results which stores model performance metrics from custom evaluation runs.

| Field | Type | Description | Constraints |
|---|---|---|---|
| _id | string | Firebase document ID | Auto-generated |
| scores | object | Evaluation metrics | Required |
| k | integer | Context complexity value | Required, positive integer |
| model | string | Model identifier | Required, non-empty |
| totalContexts | integer | Number of contexts evaluated | Required, positive integer |
| iterations | integer | Iterations per context | Required, positive integer |
| chartImage | string | Base64 encoded chart | Optional |
| timestamp | timestamp | Evaluation time | Auto-generated |

Table 5: EvaluationResult Schema

### Default Variables

Table 6 displays the data model for default variables, this is a map of variable names to value arrays which can be used by all contexts, questions and ground truths in the prompt creation suite for setting random variables to enable the systematic variation of prompts.

| Field | Type | Description | Constraints |
|---|---|---|---|
| variables | object | Map of variable names to value arrays | Required |

Table 6: DefaultVariables Schema

### 4.4.3 Additional Design Decisions

**UI Design Choices**

The front-end follows modern React best practices by using a component based architecture with well structured reusable components. Custom hooks are used for shared logic and Shadcn components are used for system design consistency. The UI is organised around distinct functional areas (Playground, Evaluation Suite, etc.).

**External Interfaces & API Integration**

The system interfaces with multiple external API's (Hugging Face & Cloudflare AI). The project supports provider abstraction by grouping models together by provider, separate backend endpoints are in place for different providers and error handling is implemented for timouts and API failures.

**Concurrency & Asynchronous Operations**

The application uses asynchronous operations, in the front-end through React hooks, fetch API with async/wait and in the backend through FastAPI async handlers which also use timeouts to manage the duration of asynchronous tasks.

**Error & Exception Handling**

Error handling is implemented at multiple levels throughout the project including, try/catch blocks with user friendly messages for failed API requests, component error states which provide UI feedback for errors and structured error responses in the backend.

**Security Considerations**

The system handles API keys and service account credentials using environment variables to protect sensitive information and security rules are configured in Firebase itself to ensure the database is protected from unauthorised read/write injections.

**Scalability & Performance**

Several of the design choices impact scalability and performance, having docker in place allows us to scale horizontally, by creating multiple container instances we can handle increased workloads. API timeouts improve performance by preventing slow model responses from affecting user experi-

ence and the UI shows loading states for long operations so users are aware of model responses in progress. The model evaluation performance visualisations and metrics calculations are designed to scale for handling larger datasets as well as the smaller subsets used in this study.

# 5 Implementation

## 5.1 Front-end Software Language & Libraries

The front-end is built using React (v18) [34], a JavaScript [35] library for building user interfaces. React allows for component-based architecture, which helps organise the code into reusable, maintainable pieces. The project was bootstrapped with Create React App, providing a standardised setup for React applications with built-in tools for development. JavaScript is the primary programming language used throughout the front-end. The front-end follows modern JavaScript practices, including ES6+ features like arrow functions and async/await for handling asynchronous operations. Tailwind CSS [36] is used for implementing the style of the application. This CSS framework allows for fast UI development by applying pre-defined classes directly in the markup. The project uses Tailwind to define custom themes and extend the default styling options. A table providing an overview of additional front-end software libraries is viewable in Appendix B.

## 5.2 Backend Software Language & Libraries

The back-end is built using Python (v3.12) [37] along with FastAPI [38], a modern, high-performance web framework for building APIs. FastAPI provides data validation and asynchronous support making it well suited for creating maintainable APIs. Python was chosen as the primary programming language for the backend due to its large range of machine learning and natural language processing libraries. The backend follows RESTful API principles with endpoints defined for handling operations like context and question management, interaction with LLMs and model evaluation. Appendix B provides a table with an overview of additional libraries used in the back-end.

## 5.3 Development Environment

Visual Studio Code [39] was used for developing both the front-end and the backend, it provides developers with many powerful features and plugins. It is also considered one of the best IDEs for

JavaScript and Python development [40], [41].

### 5.3.1 IDE Extensions

Several VS Code extensions enhanced the development workflow Python for Python language support, Rainbow CSV to improve csv file readability, Docker for containerisation integration and Github Copilot [42] for automating repetitive tasks and boosting productivity.

### 5.3.2 Containerisation

Docker [22] and Docker Compose [43] were used to create consistent, reproducible development environments. This approach allowed for:

- Isolation of the front-end and backend services
- Simple service orchestration with a single `docker-compose up` command
- Hot reloading capabilities for both React and FastAPI components
- Consistent development experience across desktop and laptop devices

### 5.3.3 Package Management & Version Control

For dependency management, npm [44] was used for JavaScript dependencies in the front-end and poetry [45] for Python dependencies in the backend. This pairing simplified managing dependencies across both services of the application. Git [46] was used for version control with feature branches and merge requests to maintain code quality.

## 5.4 Model Evaluation Algorithms & Techniques

This subsection provides an overview of the text similarity metrics used in model evaluation, including their mathematical foundations.

### 5.4.1 Sentence-BERT (Custom Evaluation)

Sentence-BERT (S-BERT) is a modification of the BERT base model architecture that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings from text [47] using the WordPiece subword tokenisation algorithm [48]. The S-BERT MiniLM model [25]

used in this study is a distilled version of S-BERT that maintains high performance while being more efficient and better suited for our use case due to hardware constraints.

**Mathematical Foundation [49]:**

The similarity between two texts is computed as the cosine similarity between their embeddings:

$$\text{S-BERT Similarity}(T_1, T_2) = \frac{\vec{E}_{T_1} \cdot \vec{E}_{T_2}}{||\vec{E}_{T_1}|| \cdot ||\vec{E}_{T_2}||} \tag{1}$$

Where:

- $\vec{E}_{T_1}$ is the embedding vector of text $T_1$
- $\vec{E}_{T_2}$ is the embedding vector of text $T_2$
- $\cdot$ represents the dot product
- $||\vec{E}_{T_x}||$ represents the Euclidean norm of vector $\vec{E}_{T_x}$

### 5.4.2 STS-RoBERTa (Custom Evaluation)

STS-RoBERTa [26] (Semantic Textual Similarity with RoBERTa) also derives semantically meaningful sentence embeddings from text, similarily to S-BERT using the WordPiece subword tokenisation algorithm but with RoBERTa as the base model. The RoBERTa base model is an optimised version of the BERT base model with improved training methodology [50].

**Mathematical Foundation [49]:**

The similarity is also calculated using cosine similarity between two texts:

$$\text{STS-RoBERTa Similarity}(T_1, T_2) = \frac{\vec{E}_{T_1} \cdot \vec{E}_{T_2}}{||\vec{E}_{T_1}|| \cdot ||\vec{E}_{T_2}||} \tag{2}$$

### 5.4.3 ROUGE Metrics (Custom Evaluation)

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics designed to compare the similarity between a machine generated text and a reference text using overlapping n-grams. In this study we will be using ROUGE-N explored below [51].

**ROUGE-N (N-gram Overlap)**

ROUGE-N measures the overlap of n-grams (groups of words or tokens) between the generated text and reference text. For example, ROUGE-1 (unigram) measures the over lap of 1-gram which is a single word and ROUGE-2 (bigram) measures the overlap of 2-grams which is a two word sequence. In order to compare the structure of LLM responses rather than the vocabulary used in their responses we will use ROUGE-2 (bigram) in this study.

**Mathematical Foundation [27]:**

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{Reference}} \sum_{\text{n-gram} \in S} \text{Count}_{\text{match}}(\text{n-gram})}{\sum_{S \in \text{Reference}} \sum_{\text{n-gram} \in S} \text{Count}(\text{n-gram})} \tag{3}$$

Where:

- $\text{Count}_{\text{match}}(\text{n-gram})$ is the maximum number of n-grams co-occurring in the candidate text and the reference
- $\text{Count}(\text{n-gram})$ is the number of n-grams in the reference

# 6 Testing

## 6.1 FastAPI Backend Testing

For testing the FastAPI backend, pytest [52] was used as the testing framework, chosen for its usability and lightweight features. Unit tests were implemented to verify important individual components like the relationship extractor and model evaluator which are critical for model evaluation. Integration tests were essential for testing that our Firebase, Hugging Face and Cloudflare API calls worked seamlessly. Mocking was implemented so we did not have to rely on third party availability or rate limits. Overall our backend tests cover the core functionality that users rely on, coverage scores are viewable in Figure 4.

| Name | Stmts | Miss | Cover | Missing |
|---|---|---|---|---|
| __init__.py | 0 | 0 | 100% | |
| app.py | 209 | 24 | 89% | 32, 38-4 |
| cloudflare_api.py | 54 | 7 | 87% | 29, 33, |
| hugging_face_api.py | 40 | 3 | 92% | 39-41 |
| model_evaluation.py | 59 | 9 | 85% | 38-40, 4 |
| relationship_extractor.py | 31 | 3 | 90% | 81-83 |
| tests/__init__.py | 0 | 0 | 100% | |
| tests/conftest.py | 21 | 5 | 76% | 20, 24-2 |
| tests/unit/__init__.py | 0 | 0 | 100% | |
| tests/unit/test_app.py | 216 | 14 | 94% | 13-14, 5 |
| tests/unit/test_cloudflare_api.py | 82 | 0 | 100% | |
| tests/unit/test_hugging_face_api.py | 61 | 0 | 100% | |
| tests/unit/test_model_evaluation.py | 82 | 0 | 100% | |
| tests/unit/test_relationship_extractor.py | 33 | 0 | 100% | |
| TOTAL | 888 | 65 | 93% | |

Figure 4: Backend Code Coverage

| File | % Stmts | % Branch | % Funcs | % Lines |
|---|---|---|---|---|
| All files | 90.25 | 81.28 | 86.04 | 91.66 |
| src | 93.75 | 87.5 | 100 | 93.75 |
| App.js | 93.75 | 87.5 | 100 | 93.75 |
| src/config | 100 | 100 | 100 | 100 |
| firebase.js | 100 | 100 | 100 | 100 |
| src/pages/clutrrEvaluation | 89.7 | 72.5 | 100 | 90.9 |
| ClutrrEvaluationPage.js | 89.7 | 72.5 | 100 | 90.9 |
| src/pages/evaluationSuite | 96.12 | 92.3 | 87.09 | 96.77 |
| DefaultVariables.js | 100 | 90 | 100 | 100 |
| EvaluationSuite.js | 88.37 | 90.9 | 73.33 | 90.24 |
| PythonBoxPlot.js | 100 | 95 | 100 | 100 |
| src/pages/home | 100 | 100 | 100 | 100 |
| HomePage.js | 100 | 100 | 100 | 100 |
| src/pages/playground | 100 | 100 | 100 | 100 |
| Playground.js | 100 | 100 | 100 | 100 |
| src/pages/playground/components | 80.41 | 74.13 | 74.19 | 83.51 |
| ChatPlayground.js | 80.41 | 74.13 | 74.19 | 83.51 |
| src/pages/promptSuite | 91.66 | 90 | 100 | 91.66 |
| PromptSuite.js | 91.66 | 90 | 100 | 91.66 |

Test Suites: 26 passed, 26 total
Tests: 165 passed, 165 total

Figure 5: Front-end Code Coverage

## 6.2 React Front-end Testing

For testing the front-end react application, Jest [53] was used for its reliability and unique features. Jest tests were created for React components to ensure they manage user inputs and backend endpoint interactions as expected. Tests were also implemented to simulate user interactions with our application, to verify that errors were handled and responses were displayed correctly. Firebase and backend API endpoint connections were mocked so we could test error handling paths that would be hard to trigger with real services. Intermediate test coverage scores of key components of the front-end are available in Figure 5, with a more detailed report highlighting broader coverage viewable in Appendix A.

## 6.3 Model Evaluation Testing

The model evaluation process detailed in the following sections acts as a test of our entire system, from dataset creation, variable substitution, API communication and model response evaluation. Our system is tested here in a real world environment where actual LLM responses are evaluated and results are documented. By collecting metrics and generating graphs we validate our systems core purpose, to interact with LLMs systematically and get insights into their performance on various inferencing tasks.

# 7 Evaluation Methodology

## 7.1 Custom Evaluation Methodology

Our custom evaluation methodology was inspired by two key studies: the Apple GSM-Symbolic study [8] and the Facebook CLUTRR study [7]. Our prompt templating techniques were adapted from the Apple study, they involve using a structured prompt to guide the model's response. The use of relational inferencing challenges to assess Large Language Model capabilities is inspired by the CLUTRR study.

This evaluation utilises our own custom dataset. The fundamental principle behind this dataset is the systematic variation of base narrative text templates. We generate multiple variations of the base narratives by altering specific variables (like names and locations) while preserving the core narrative structure and the inferencing paths. Each varied narrative is paired with a unique question designed to test the model's inferencing capbilities. Using this approach increases the number of evaluation samples and allows us to assess the model's robustness to distracting variables changed in the input.

For instance, in a narrative text a sentence like "John enjoyed a lovely day at the beach" could be varied to "Paul enjoyed a peaceful day at the supermarket," requiring the model to perform the same type of inference despite the altered variables.

Furthermore, to systematically evaluate model capabilities under increasing difficulties, the dataset has five difficulty levels, tasks k=2 to k=6, defined by the number of relational inference steps introduced. Each difficulty level has ten base stories, which are then varied to generate a set of one hundred unique test samples per difficulty level. Details of the dataset creation, difficulty levels, prompt design, and the prompt variation process are discussed in the following sections.

### 7.1.1 Prompt Design

To ensure consistent interaction with the Large Language Model and to guide its output format for easier evaluation, we used the following structured prompt template for all queries:

**Prompt Template**

**Task:** Infer the relationship between characters in a story.

**Format:** Respond only with "*Person1 is Person2's relation*" using an allowed relationship.

**Allowed Relationships:** father, mother, son, daughter, brother, sister, husband, wife, grandfather, grandmother, uncle, aunt, cousin, niece, nephew, father-in-law, mother-in-law, grandson, granddaughter

**Example:** If Tom is married to Lisa and Lisa's son is Mark, then Mark is Tom's *son*.

**Story:** $Context

**Question:** $Question

This template explicitly defines the task for the model. The **Format** instruction, combined with the list of **Allowed Relationships**, strictly guides the model's output structure, minimising ambiguous or incorrectly formatted responses. The **Example** provides a clear illustration of the expected inferencing challenge and output format. Finally, the `$Context` and the corresponding `$Question` are injected into the prompt template for each evaluation run.

### 7.1.2 Dataset Construction and Details

Our evaluation relies on a custom dataset specifically designed to test relational inferencing capabilities of LLMs. This section details the datasets structure, complexity levels, and the nature of the data samples.

#### Base Stories and Systematic Variation

The dataset is built upon a principle of narrative text variations. For each difficulty level, a set of ten base narrative text templates was created. Each template outlines a narrative containing specific relationships. To generate the final evaluation samples, each base narrative template can be varied dynamically up to ten times during evaluation runs. Variation involves systematically replacing placeholder variables within the template (such as names, places, emotions, actions, etc.) with different values from an array of unique or default variables stored in our database.

This process yields $10 \times 10 = 100$ unique narrative text samples per difficulty level, resulting in a total of 500 distinct evaluation samples in total across the five levels. This variation strategy allows us to:

- Increase the number of evaluation samples from a smaller set of base templates.
- Test the model's robustness to variations in the text that do not alter the underlying relational structure or the required inference path.

**Difficulty Levels and Complexity Scaling**

To assess model performance across different inferential challenges, the dataset is divided into five difficulty levels, labeled by $k$, ranging from $k = 2$ to $k = 6$. The value of $k$ directly corresponds to both:

1. The minimum number of explicit relational links or inference steps required to correctly answer the associated question.
2. The number of "noise" variables included in the narrative text. These are variables such as (emotions, places, activities) that are part of the narrative but are not essential for reaching the target relationship, adding a potential distraction for the model.

Therefore, as $k$ increases, the narratives become longer, involve more characters, contain more distracting information, and require longer chains of inferencing to solve. The structure is summarised in Table 7.

| Level ($k$) | Relationships | Noise Variables | Base Stories | Variations | Samples/Level |
|---|---|---|---|---|---|
| 2 | 2 | 2 | 10 | 10 | 100 |
| 3 | 3 | 3 | 10 | 10 | 100 |
| 4 | 4 | 4 | 10 | 10 | 100 |
| 5 | 5 | 5 | 10 | 10 | 100 |
| 6 | 6 | 6 | 10 | 10 | 100 |
| **Total** | - | - | **50** | - | **500** |

Table 7: Dataset Structure per Difficulty Level

**Illustrative Examples**

To show the progression in difficulty level, explore the following examples (using placeholder variables before variation):

**Example:** $k = 2$

> **Story:** {male_name1} and his grandson {male_name2} went to the {place}, and had a good time together. {male_name3} was {emotion} his brother, {male_name2}, was able to make it to the party.
> **Question:** What is {male_name3}'s relationship to {male_name1}?
> **Groundtruth:** {male_name3} is {male_name1}'s grandson

*Inference Path (2 steps):* {male_name3} $\rightarrow$ brother $\rightarrow$ {male_name2} $\rightarrow$ grandson of $\rightarrow$ {male_name1}. Requires identifying the sibling relationship and then the grandparent-grandchild relationship. Contains 2 noise variables (e.g., {place}, {emotion}).

**Example:** $k = 6$

> **Story:** {female_name1} and her son {male_name1} made {food}. {male_name1}'s brother {male_name2} ate one. {male_name2}'s mother, {female_name1}, was {emotion} that he failed his {education} class. {male_name3}'s grandmother, {female_name2}, was {emotion} to spend a weekend with all of her grandchildren. {male_name2} got his son, {male_name4}, a {gift} for his {event}. {male_name2} played {game} with his brother {male_name3}.
> **Question:** What is {female_name2}'s relationship to {male_name4}?
> **Groundtruth:** {female_name2} is {male_name4}'s mother

*Inference Path (6 steps):* This example involves multiple individuals and relationships (parent-child, sibling, grandparent-grandchild). Correctly answering the question requires navigating a chain of six relational links while ignoring six distracting noise variables ({food}, {emotion}, {education}, {gift}, {event}, {game}). The increased number of characters and relationships increases the inferential complexity compared to the $k = 2$ case.

**Unique Name ("Wildcard") Samples**

As an additional exploratory test within the dataset design, one base story template per difficulty level was designated as a "wildcard". During variation, this specific template used variables drawn from lists of unique, fictional or famous, names (e.g., {unique_male_name1}, {unique_female_name1}) instead of the more common names used in the other nine templates per level.

The addition of these wildcard samples was driven by curiosity to observe whether substituting standard names with unique character names which might be less common in the model's pre-training data would yield any noticeable difference in performance on the relational inferencing task. While not designed as a rigorous test of a specific hypothesis, it is an interesting probe into whether the model's inferencing is sensitive to the familiarity of the names involved in the narrative text.

In summary, our custom dataset provides a controlled environment for evaluating relational inferencing, featuring systematic variation, scaled complexity levels ($k = 2$ to $k = 6$) incorporating both inference paths and noise, and probes for model sensitivity using unique names.

### 7.1.3 Evaluation Strategy

To assess the performance of the Large Language Models (LLMs) on our custom relational inferencing dataset, we used an evaluation strategy, focusing on the correctness of the inferred relationship, while also considering the model's adherence to the specified output format.

**Primary Evaluation: Correctness via Two-Level Matching**

The principal method for determining the accuracy of a model's response involves a two-level matching system applied after basic text normalisation (e.g., lowercasing, removal of punctuation) of the model's raw output. This system compares the core relationship extracted from the normalised model response against the relationship specified in the ground truth. The process is illustrated in Figure 6.

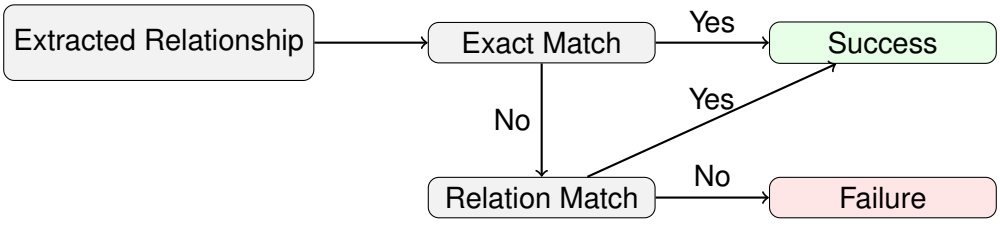



Figure 6: Two-level matching system for CLUTRR relationship evaluation

The matching process is carried out as follows:

1. **Relationship Extraction:** The relationship term (e.g., "grandson", "mother", "brother") is

extracted from the normalised model response and the ground truth string using regular expressions (regex). These regex patterns are designed to extract the relation based on the expected output structure and instructions specified in the prompt template.

2. **Exact Match:** The extracted relationships are then compared together. If they are an identical match (e.g., model responds "grandson", ground truth is "grandson"), the response is marked as a success.

3. **Relational Match:** If the Exact Match fails, a second check is performed using a predefined dictionary of synonymous terms. This ensures a model is not penalised for responding using an equivalent relational term. Examples of accepted mappings include:

   - ‘mother‘ $\leftrightarrow$ ‘mom‘, ‘mum‘
   - ‘grandson‘ / ‘granddaughter‘ $\leftrightarrow$ ‘grandchild‘

   If the extracted model relationship maps to the extracted ground truth relationship via this dictionary, the response is also marked as a success.

4. **Failure:** If the exact match and the relational match are unsuccessful, the response is marked as a failure.

The primary performance metric derived from this process is **Accuracy**, calculated as the total number of successful matches (either exact or relational) divided by the total number of evaluation samples for a given difficulty level. Accuracy $= \frac{\text{Number of Successes}}{\text{Total Samples}}$

**Secondary Analysis: Output Format Adherence via Semantic Similarity**

Initially, we explored using semantic similarity metrics, specifically Sentence-BERT (S-BERT) & RoBERTa embeddings using cosine similarity, and lexical overlap metrics like ROUGE, to compare the model's raw response directly against the ground truth string. However, this approach was unreliable for assessing the factual correctness of the inferred relationship. For instance, responses like "*Trey is John's son*" could achieve high similarity scores when compared to the ground truth "*Trey is John's grandson*", despite being factually incorrect.

After Recognising this limitation, we pivoted these similarity scores to measure the models ability to adhere to the specified output format outlined in the prompt template.

A high semantic similarity score suggests the model generated a response that is structurally similar to the target format ("*Person1 is Person2's relation*"), even if the specific relation term itself is incorrect. On the other hand very low scores indicate responses that deviate from the expected structure (e.g., providing lengthy explanations, refusing to answer, or using a completely different sentence format). These similarity scores are therefore analysed separately to provide insights into instruction following capabilities, rather than contributing to the primary accuracy calculation. The results of this similarity analysis are presented using box plots to visualise the distribution of scores for each model across the samples.

## 7.2 CLUTRR Benchmark Methodology

CLUTRR (Compositional Language Understanding with Text-based Relational Reasoning) [7] is a diagnostic benchmark designed to test inductive reasoning capabilities, specifically focusing on inferring complex family relationships from narrative text. Models are provided with stories containing explicit relationships and are queried about implicit relationships requiring multi-step inferential reasoning.

### 7.2.1 CLUTRR Dataset Structure and Challenges

The CLUTRR dataset presents relational reasoning problems of increasing complexity, organised into tasks. We used tasks ranging from 1.2 (requiring a two-step inference) up to 1.10 (requiring a ten-step inference). The following examples in this section show the simpler task 1.2 and the more complex task 1.10.

**Example: CLUTRR Task 1.2 (2-step inference)**

> **Story:** [Scott] and [Lewis] are brothers. [Jason] is the father of their father.
> **Query:** Lewis is Jason's _____?
> **Target:** grandson

*This task requires composing two relationships (brother, father of father) to infer the target (grand-*

*son).*

**Example: CLUTRR Task 1.10 (10-step inference)**

> **Story:** [Kathleen] took her sister, [Mabel], out to dinner for her birthday. [Mabel], who is the sister of [Sharon], is a lovely girl. [Mabel] took her sister, [Kathleen], out to dinner for her birthday. [Nadia] and her sisters [Sharon] went to the spa. [Mabel], another sister of [Nadia], had to babysit and couldn't join them. [Ellen] went to the mall, because she wanted to look for a present for her daughter, [Mabel]. [Ellen] asked her daughter, [Mabel], if she would like to go to a movie with her on Saturday night. [Ellen] became concerned when she hadn't heard from her husband [James] all day. [Mabel] went over to her uncle [William]'s house for dinner. [Mabel] vowed to never trust her father, [James] with her debit card again.
> **Query:** William is Ellen's _____?
> **Target:** brother

*This task involves a significantly longer story with multiple characters and relationships, requiring the model to trace a 10-step inference path while ignoring distractors to find the relationship between William and Ellen.*

Key challenges presented by the CLUTRR dataset as complexity increases include:

- **Increasing Chain Length:** Requiring maintenance of coherence across more inferential steps.
- **Narrative Distraction:** Testing the ability to bypass irrelevant facts from longer stories.
- **Relational Diversity:** Broader knowledge of different family relationship terms is needed.
- **Increased coherence:** Requiring keeping track of characters mentioned multiple times.

### 7.2.2 Evaluation Strategy for CLUTRR

The original CLUTRR paper focuses on dataset generation and does not provide a standardised script or methodology for evaluating model outputs against the target labels [7]. While Jaccard similarity metrics were used during dataset creation, they were only intended for ensuring example diversity, not for evaluating model correctness.

For consistency across our evaluations, we used the evaluation strategy previously detailed for our custom dataset. The key components applied are:

- **Prompting:** Models were given the CLUTRR stories and queries using a structured prompt template similar to the one used in our custom evaluation, it was specifically adapted to use the 'Story:', 'Query:' and 'Target:' fields provided by CLUTRR.
- **Relationship Extraction:** The target relationship was extracted from the model's cleaned response using the same regular expression techniques employed for our custom dataset.
- **Correctness Matching:** We applied the same two-level matching system (Exact Match followed by Relational Match) to determine if the extracted relationship was correct. Viewable in Figure 6.
- **Metric:** The performance metric used to quantify model performance is accuracy, calculated as the number of successful matches (Exact or Relational) divided by the total number of samples evaluated for each CLUTRR task level (1.2 through 1.10).

The secondary analysis using semantic similarity (S-BERT) and lexical overlap (ROUGE) metrics, which was used to assess format adherence for our custom dataset, was not applied to the CLUTRR evaluation. To align with the CLUTRR benchmark's goal of testing relational reasoning capabilities, the focus for this benchmark was on the accuracy of the inferred relationship, determined by our two-level matching logic.

### 7.3 Apple Dataset

The Apple dataset is a modified version of the GSM8K dataset [54], known as the GSM-Symbolic introduced by [8]. While the original GSM-Symbolic dataset evaluated mathematical reasoning in LLMs through grade-school math problems, our adaptation pivots to assess spatial inferencing capabilities. Although the Apple dataset inspired our custom evaluation templating approach for prompt variation, we were unable to use this dataset in our evaluation due to CloudFlare API and Hugging Face API rate limitations.

### 7.3.1 Dataset Transformation

We transformed the mathematically focused GSM-Symbolic dataset by changing numerical operations into spatial inferencing scenarios while trying our best to preserve the multi-step reasoning structure that made the original dataset valuable.

**Example Transformation**

**Original Example:**
"Sanjay saw a 60-foot dolphin with 16 12-inch remoras attached to it. But a quarter of the remoras go away. What percentage of the dolphin's body length is the combined length of the remaining remoras?"

**Transformed Example:**
**Context:** "Sanjay saw a dolphin to the east of the coral reef. The coral reef is to the east of a sunken ship. The sunken ship is to the south of a fishing boat. There is a island north of the fishing boat."
**Question:** How would would the Dolphin get to the island?
**Ground Truth:** west west north north

Figure 7: Example of transformation from a mathematical reasoning problem to a spatial inferencing problem.

Figure 7 represents an example that shows our transformation process, showing an original GSM-Symbolic problem alongside our modified version, this transformation shifts from a multi-step mathematical calculation to a spatial inferencing problem requiring inference across multiple statements.

### 7.3.2 Challenges and Limitations

The dataset transformation process raised several challenges:

- Maintaining consistent complexity when changing from a mathematical to textual spatial inferencing problems
- Preserving the multi-step inference chains in the transformed examples
- Manually transforming a dataset takes significant time

Despite our methodical approach, the manual conversion process made it difficult to maintain inference quality across all transformed examples. Additionally, When transforming mathematical

reasoning problems into spatial inferencing equivalents, maintaining consistent difficulty levels is challenging to quantify due to the differences in assessment criteria.

# 8 Evaluation Results

This section assesses the performance of four large language models Mistral (7B), Llama 3.1 (8B), Llama 3.1 (70B), and Llama 3.3 (70B) on two relational inferencing benchmarks: our custom dataset and the original CLUTRR dataset.

## 8.1 Custom Evaluation Performance

First we evaluated the models on our custom dataset, focusing on tasks $k = 2$ through $k = 6$. The accuracy scores for each model across these tasks are presented in Figure 8 and Table 8.

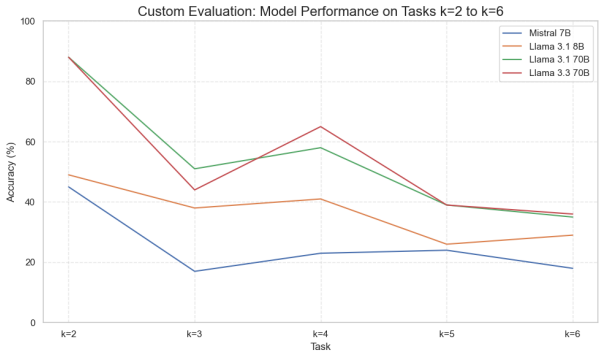



Figure 8: Custom Evaluation Model Performance Visualisation

| Model | k=2 | k=3 | k=4 | k=5 | k=6 |
|---|---|---|---|---|---|
| Mistral | 45% | 17% | 23% | 24% | 18% |
| Llama 3.1 (8B) | 48% | 38% | 41% | 26% | 29% |
| Llama 3.1 (70B) | 88% | 51% | 58% | 39% | 35% |
| Llama 3.3 (70B) | 88% | 44% | 65% | 39% | 36% |

Table 8: Accuracy results (%) for each model across different values of $k$.

### 8.1.1 Model Comparison:

Overall in Figure 8 and Table 8 we can see a decline in performance as the number of inferencing steps ($k$) increase, this indicates that the models begin to be challenged with added complexity. Llama 3.1 (70B) and Llama 3.3 (70B) both demonstrate reasonable performance, particularly at $k = 2$ where they both achieve 88% accuracy. Their performance drops considerably at $k = 3$, then recovers slightly at $k = 4$ before beginning to steadily decline again at the higher $k$ values. Overall both models are equally matched, with Llama 3.3 showing slightly better performance at $k = 4$ (65% vs 58%) but otherwise performing similarly to Llama 3.1 (70B). This suggests that

architectural refinements don't always correlate to improved performance on inferencing tasks.

Llama 3.1 (8B) and Mistral (7B) also show a drop in performance after $k = 2$. Overall Llama 3.1 (8B) consistently outperforms Mistral (7B) across all $k$ values, despite Mistral having only slightly fewer parameters. For instance, at $k = 3$, Llama 3.1 (8B) scores 38% accuracy compared to Mistral's 17%. This dominance over Mistral indicates that the Llama architecture is better suited for inferencing tasks.

The impact of parameter size is evident when comparing Llama 3.1 (8B) and Llama 3.1 (70B). Although both models use the same architecture and training paradigms the 70B model has higher accuracy across all tasks, often by 15-40% (e.g., 88% vs 48% at $k = 2$ & 58% vs 41% at $k = 4$). This highlights the importance of model size in carrying out complex inferencing tasks.

#### 8.1.2 Wildcard Sample Results:

We included a small subset of "wildcard" samples containing unique, potentially unfamiliar names to test model robustness. As shown in Table 9, performance on these samples was generally poor across all models and tasks, often falling well below the average accuracy for the corresponding $k$ value seen in Table 8. For instance, Llama 3.1 (70B) scored 100% at $k = 2$ but 0% at $k = 3$ and $k = 4$. While the limited size of this subset prevents definitive conclusions, it suggests that models may struggle when they encounter unfamiliar entities within an inferencing chain.

| Model | k=2 | k=3 | k=4 | k=5 | k=6 |
|---|---|---|---|---|---|
| Mistral (7B) | 0% | 0% | 0% | 10% | 0% |
| Llama 3.1 (8B) | 0% | 0% | 0% | 20% | 0% |
| Llama 3.1 (70B) | 100% | 0% | 0% | 10% | 0% |
| Llama 3.3 (70B) | 80% | 0% | 10% | 10% | 0% |

Table 9: Overall Accuracies for Wildcard Samples (%) for Tasks k=2 to k=6 across Models

#### 8.1.3 Prompt Variation Insights:

The model performances observed in our custom evaluation are very similar to those seen in the CLUTRR benchmark evaluation (discussed next). Considering the CLUTRR benchmark uses a

unique inferencing path for each sample, the score similarity between our benchmarks suggests that variations of the base prompt structure has a negligible impact on overall performance. The main factor seems to be the increasing difficulty associated with the number of inferencing steps required.

### 8.1.4 Model Response Structure Analysis:

To assess the consistency and structural quality of model outputs, we compared the similarity of model responses to the ground truth expected format using S-BERT, RoBERTa, and ROUGE-2 metrics. The box plots in Figure 9 compare one of our larger models (Llama 3.1 70B) and one of our smaller models (Llama 3.1 8B), the results indicate that Larger models generally produce responses with higher median similarity scores and lower variance (tighter boxes, fewer outliers) compared to smaller models, this can be seen clearly with the RoBERTa and ROUGE-2 boxplots. This shows that the larger model often generate more consistent and better-structured outputs, demonstrating its ability to follow the desired response given in our prompt template. However, the presence of outliers for both models indicates instances of deviation even for the larger model.

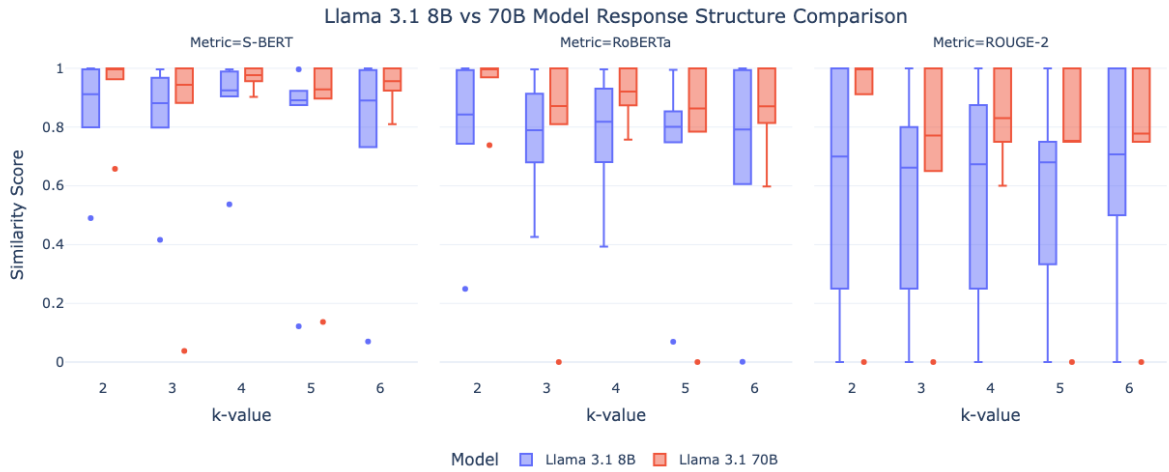



Figure 9: Response Structure Comparison

## 8.2 CLUTRR Benchmark Evaluation

We also evaluated the same four models on the CLUTRR dataset, focusing on tasks 1.2 through 1.10, which represent increasing reasoning chain lengths. The overall accuracy scores of this

evaluation are summarised in Table 10.

| Model | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 |
|---|---|---|---|---|---|---|---|---|---|
| Mistral (7B) | 39.5% | 13.2% | 21.1% | 21.1% | 17.1% | 17.1% | 28.9% | 22.4% | 25.0% |
| Llama 3.1 (8B) | 44.7% | 30.3% | 39.5% | 34.2% | 31.6% | 11.8% | 31.6% | 23.7% | 19.7% |
| Llama 3.1 (70B) | 90.8% | 53.9% | 53.9% | 42.1% | 46.1% | 26.3% | 40.8% | 30.3% | 39.5% |
| Llama 3.3 (70B) | 88.2% | 46.1% | 52.6% | 42.1% | 42.1% | 25.0% | 40.8% | 31.6% | 36.8% |

Table 10: Overall Accuracies (%) for Tasks 1.2 – 1.10 across Models

The performance trends seen on the CLUTRR dataset align with the findings from our custom evaluation. Accuracy generally decreases as task complexity increases. The performance ranking of the models also remains consistent: the (70B) Llama models outperform the (8B) Llama and (7B) Mistral models, and Llama 3.1 (8B) generally performs better than Mistral (7B).

### 8.2.1 Comparison to Original CLUTRR Evaluation:

An Important part of this evaluation is comparing the performance of these modern LLMs against the models evaluated in the original CLUTRR paper [7], primarily BERT and BiLSTM variants. Figure 12 presents a side-by-side comparison of our results (Figure 10) and the original paper's results (Figure 11).

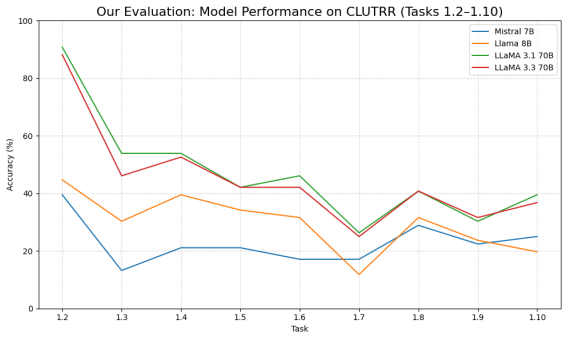



Figure 10: Our CLUTRR Evaluation

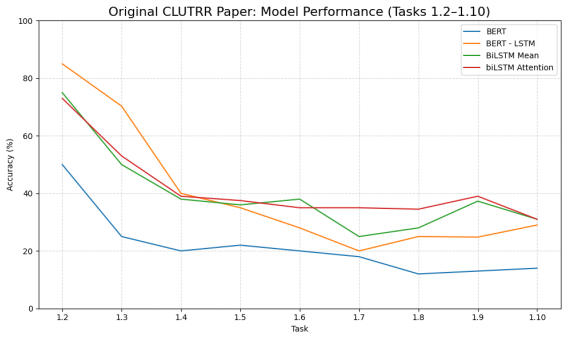



Figure 11: Original CLUTRR Evaluation

Figure 12: Side-by-side comparison of CLUTRR performance: Our evaluation (left) vs. Original paper (right).

The overall similarity of decreasing accuracy with increasing task complexity (longer reasoning chains) is consistent across both evaluations. Task 1.7 appears to be the most challenging in both studies, suggesting tough inferencing tasks for the models.

Our Llama 3.1 (8B) model achieves performance levels that are competitive with, and sometimes exceed, those of the smaller original models. Our Llama 3.1 (70B) and Llama 3.3 (70B) models consistently outperform the best models from the original paper, however the more modern Llama models demonstrate a lack of significant improvement in efficiency, especially considering the more advanced training techniques, model architectures and parameter sizes these models use. The BERT-LSTM and BiLSTM Attention models used in the original CLUTRR benchmark are only reported to have up to 110M parameters [55], (although it does have to be noted that they were trained specifically on the CLUTRR dataset before testing).

While the 70B Llama variants do outperform their predecessors, the efficiency compared to the specialised 110M models from the original paper indicate potential limitations in how modern architectures approach inferential reasoning tasks.

#### 8.2.2 CLUTRR Evaluation Key Insights:

The superior performance-per-parameter of the original smaller models highlights the effectiveness of training models on complex inferencing tasks. While scaling parameter size (8B vs 70B) clearly boosts performance in the newer models, the fact that modern 70B models almost fall behind older smaller models, suggests training models on complex inferential reasoning tasks is essential to improving Generative AI inferencing capabilities going forward.

However, the persistent difficulty with certain tasks for both the newer and older models (e.g. 1.3 through 1.10) indicates that even models trained on the data struggle with the inferencing tasks at hand. The performance drop-off after the initial tasks suggest potential limitations in maintaining coherence over long inference chains.

Additionally, in Figure 13 we can see how model performance seemed to vary across the relationship types being inferred. With strong performance on first degree relations (parent/child) and extremely poor performance on second degree relations (niece/nephew). This highlights how ex-

tra inferencing steps required to determine second degree relations hinders model performance further.

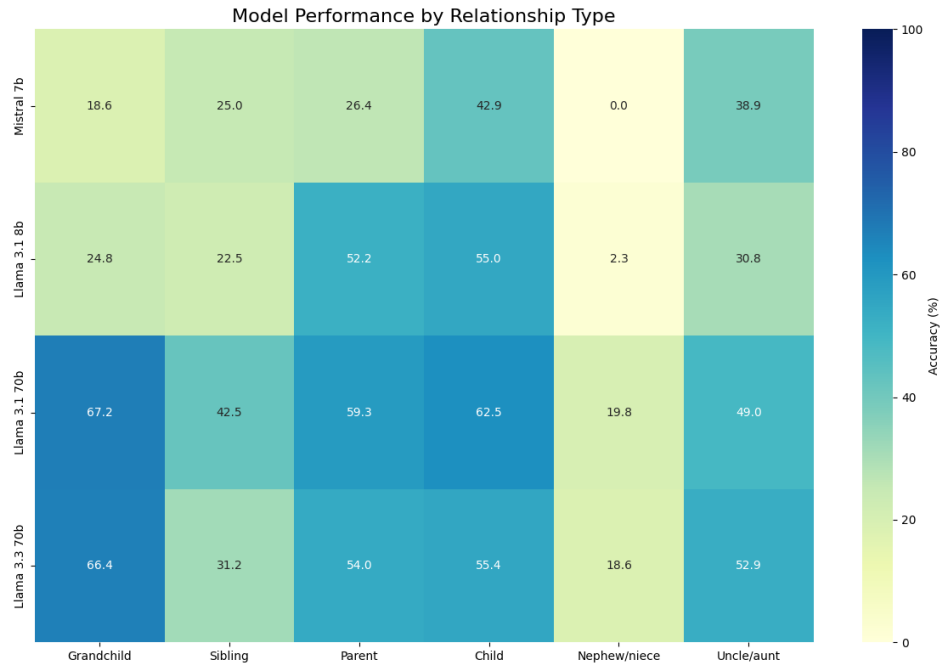



Figure 13: This figure illustrates how model accuracy varied across different relation types in our CLUTRR Evaluation.

## 8.3 Conclusion

Our evaluation of modern LLMs on both custom and CLUTRR relational inferencing benchmarks highlight important insights about the state of inferencing capabilities in current Generative AI models. While models like Llama 3.1 (70B) and Llama 3.3 (70B) demonstrate improved performance over smaller more constrained models, our comparison with models from the original CLUTRR paper shows a concerning pattern in parameter size efficiency.

The BERT-LSTM and BiLSTM Attention models (with only 110M parameters) from the original CLUTRR benchmark achieved performance levels that required modern models that are much larger to marginally improve upon, despite five years of architectural advancement and training innovations. This suggests that while raw parameter scaling improves performance, the efficiency

of how models handle inferencing has not significantly progressed.

Our findings highlight three key conclusions:

1. **Parameter scaling improves performance but with diminishing returns:** The substantial performance gap between Llama 3.1 (8B) and Llama 3.1 (70B) confirms that scaling helps, yet only small improvements over 110M specialised trained models from five years ago indicates inefficient utilisation of these additional parameters.

2. **All models seem to struggle with maintaining long inferencing chains:** The consistent decline in performance as inferencing chain length increases suggests limitations in how current architectures attempt to maintain coherent over multiple inferencing steps.

3. **Specialised training is crucial:** The competitive performance of the original smaller models, which were specifically trained on CLUTRR data, underscores the importance of targeted training for complex inferencing tasks rather than relying solely on scale and architectural advancements.

The implications for future research are clear, while evaluating even larger models like Llama 3.1 (405B), GPT-4o, or Claude 3 Sonnet would be valuable to understand increased parameter sizes, equal focus should be placed on improvements specifically designed to improve multi-step inferencing. The wildcard sample results further show the need for improved model understanding when handling unfamiliar entities within inferencing chains.

Comparing to human performance benchmarks from the original CLUTRR paper where humans achieved 100% accuracy with unlimited time but only 40-50% accuracy for $k > 3$ under time constraints, suggests that current LLMs are approaching time constrained human performance but still fall far short of human capabilities.

# 9 Project Evaluation

## 9.1 Future work

Future work should investigate two aspects: (1) investigating whether parameter scaling continues to yield improvements with even larger models, and (2) developing specialised training approaches focused on multi-step inferencing. By combining these approaches, we may be able to develop models that maintain better coherence over multi-step inferencing chains.

## 9.2 Learning outcomes

This project has reinforced the general consensus that modern LLMs face significant difficulties when required to perform successful inferences, given complex inferencing chains. On a more technical note, building knowledge around front-end development and backend API's has been a key take away from this project. Additionally, navigating through various software provider challenges and project adaptations has enhanced our adaptability skills in the constantly changing software development landscape.

## 9.3 Project Adaptations

Throughout this research project, several technical adjustments were made in response to challenges and new discoveries:

- **Dataset**: To begin with the project was only intended to utilise a custom dataset. To address concerns around reliability of this dataset and enable comparisons with industry standard datasets, the CLUTRR dataset was included in our research.
- **Database**: The original database implementation used MongoDB for data storage and retrieval. Due to MongoDB's port being blocked on the Queens network, the project was migrated to Firestore which made it available on all the required networks.
- **LLM Provider**: The system initially used the Hugging Face API as its main provider for LLMs. Updates to Hugging Face API terms and conditions forced us to find a new LLM provider, Cloudflare Workers AI API presented itself as a clear option as it offered increased

rate limits, improved reliability and a better range of LLMs to test with.

- **Data Visualisation**: Early on in development, box plot visualisations were directly created by the front-end using JavaScript. This presented problems with the data processing. To solve this issue visualisation logic was moved to the backend, making the front-end solely in charge of loading and presenting pre-generated graphs.

**Rationale for Changes**

Each adjustment represented a critical learning opportunity and ultimately strengthened the research outcomes:

- The addition of an established dataset provided further insights into how models perform on inferencing tasks.
- Firestore integration improved database query performance and was simple to work with.
- Transitioning to Cloudflare's Workers AI for main LLM interactions resolved the rate limits hugging face imposed and improved the LLMs available for research.
- Moving graph visualisation processing to the backend resulted in more reliable graphs in the front-end.

# List of Figures

# List of Tables

# Appendices

Link to meeting minutes.

# A Additional Figures

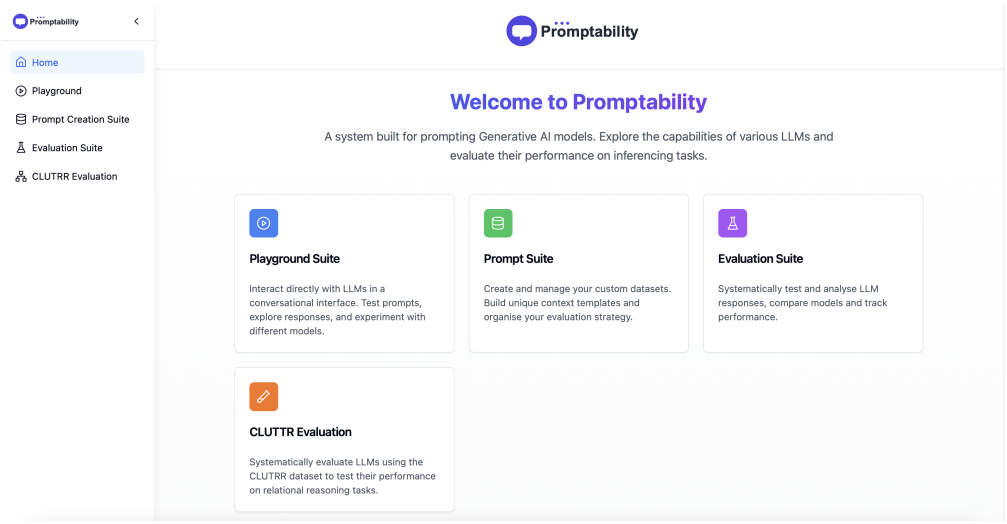



Figure 14: UI Landing Page

| File | % Stmts | % Branch | % Funcs | % Lines |
|---|---|---|---|---|
| All files | 82.87 | 76.5 | 83.41 | 83.71 |
| src | 93.75 | 87.5 | 100 | 93.75 |
| App.js | 93.75 | 87.5 | 100 | 93.75 |
| src/components/clutrr | 84.9 | 81.81 | 92.3 | 86.27 |
| ClutrrResultsVisualiser.js | 84.9 | 81.81 | 92.3 | 86.27 |
| src/components/database | 64.16 | 64.7 | 74.07 | 62.61 |
| BrowseDatabase.js | 60.18 | 59.72 | 70.21 | 58.33 |
| QuestionDisplay.js | 100 | 92.3 | 100 | 100 |
| src/components/layout | 96 | 93.1 | 92.85 | 95.45 |
| header.js | 95 | 88.23 | 90.9 | 94.11 |
| sidebar.js | 100 | 100 | 100 | 100 |
| src/components/ui | 100 | 92.85 | 100 | 100 |
| alert.js | 100 | 80 | 100 | 100 |
| button.js | 100 | 100 | 100 | 100 |
| card.js | 100 | 100 | 100 | 100 |
| switch.js | 100 | 100 | 100 | 100 |
| textArea.js | 100 | 100 | 100 | 100 |
| src/config | 100 | 100 | 100 | 100 |
| firebase.js | 100 | 100 | 100 | 100 |
| src/hooks | 72.88 | 51.42 | 80 | 72.97 |
| useDatabase.js | 53.12 | 0 | 60 | 53.33 |
| useDefaultVariables.js | 100 | 100 | 100 | 100 |
| usePrompts.js | 94.28 | 60.86 | 87.5 | 93.75 |
| src/pages/clutrrEvaluation | 89.7 | 72.5 | 100 | 90.9 |
| ClutrrEvaluationPage.js | 89.7 | 72.5 | 100 | 90.9 |
| src/pages/evaluationSuite | 96.12 | 92.3 | 87.09 | 96.77 |
| DefaultVariables.js | 100 | 90 | 100 | 100 |
| EvaluationSuite.js | 88.37 | 90.9 | 73.33 | 90.24 |
| PythonBoxPlot.js | 100 | 95 | 100 | 100 |
| src/pages/home | 100 | 100 | 100 | 100 |
| HomePage.js | 100 | 100 | 100 | 100 |
| src/pages/playground | 100 | 100 | 100 | 100 |
| Playground.js | 100 | 100 | 100 | 100 |
| src/pages/playground/components | 80.41 | 74.13 | 74.19 | 83.51 |
| ChatPlayground.js | 80.41 | 74.13 | 74.19 | 83.51 |
| src/pages/promptSuite | 91.66 | 90 | 100 | 91.66 |
| PromptSuite.js | 91.66 | 90 | 100 | 91.66 |

Test Suites: 26 passed, 26 total
Tests: 165 passed, 165 total

Figure 15: Additional Front-end Tests

# B Additional Tables

| Component | Description |
|---|---|
| Header | Displays the app logo and model selection dropdown when appropriate |
| Sidebar | Navigation component that allows users to switch between different app sections |
| HomePage | Landing page that introduces the application features and technologies |
| PromptSuite | Interface for managing prompts, contexts, and question creation |
| BrowseDatabase | Interface to browse, view and edit contexts and questions stored in the database |
| ContextForm | Form for creating new contexts with variables and metadata |
| QuestionForm | Form for creating questions associated with specific contexts |
| QuestionSelector | Component for selecting and managing questions associated with a context |
| QuestionDisplay | Displays question details and allows editing question properties |
| PromptTabs | Tab interface for managing multiple prompts in the playground |
| TextWithVariables | Interactive component that renders text with variable placeholders and dropdown selectors |
| ClutrrResultsVisuals | A component for generating visualisations of CLUTRR evaluation results |
| Button | Reusable button component with different variants |
| TextArea | Custom textarea component with consistent styling |
| Card/CardHeader | Layout components for creating card-based UI elements |
| Alert | Component for displaying different types of alert messages |
| Switch | Toggle component for boolean settings |

Table 11: Additional Front-end Components

| Category | Library | Purpose |
|---|---|---|
| Core Frontend | React v18.3.1 | Main UI framework |
| | Firebase v11.0.2 | Database functionality |
| | TailwindCSS | Utility-first CSS framework for styling |
| UI Components | Lucide React | Icon library |
| | Recharts | Chart visualisation |
| | React-Plotly | Advanced data visualisation |
| | shadcn/ui | Reusable components library |
| Data Processing | PapaParse | CSV parsing |
| | Lodash | JavaScript utility functions |
| | seedrandom | Dataset reproducibility |
| Testing | Jest | JavaScript testing framework |
| | testing-library/react | Component testing utilities |
| | testing-library/jest-dom | DOM testing extensions |

Table 12: Additional Software Libraries Used in the Front-end

| Category | Library | Purpose |
|---|---|---|
| Core Backend | FastAPI v0.104.0 | Web framework for building APIs |
| | Uvicorn | ASGI server for FastAPI application |
| | Poetry | Dependency management and packaging |
| Configuration | dotenv | Environment variable management |
| | logging | Application logging and error tracking |
| Model Evaluation | SentenceTransformer | Text embedding and similarity computation |
| | Hugging Face Evaluate | Model evaluation metrics (ROUGE) |
| | NumPy | Numerical operations for calculations |
| API Integration | Haystack | Framework for Hugging Face LLM pipelines |
| | requests | HTTP library for API calls (i.e. Cloudflare) |
| Data Handling | firebase-admin | Client for Firebase services |
| | Pydantic | Data validation and settings management |
| | Pandas | Data manipulation and analysis |
| | Asyncio | Asynchronous programming support |
| | JSON | Standard library for JSON operations |
| Testing | pytest | Python testing framework |
| | TestClient | FastAPI testing utilities |

Table 13: Additional Software Libraries Used in the Backend

## References


[1] M. Stewart, "Promptability, Software Solution For This Project." Available: https://gitlab.eeecs.qub.ac.uk/dashboard/projects, [online]. [Accessed: April. 26, 2025].

[2] MGSM, "Multi-task Language Understanding on MGSM." Available: https://paperswithcode.com/sota/multi-task-language-understanding-on-mgsm, [online]. [Accessed: April. 26 2025].

[3] MMLU, "(Massive Multitask Language Understanding)." Available: https://paperswithcode.com/dataset/mmlu, [online]. [Accessed: April. 26 2025].

[4] HumanEval, "Code Generation on HumanEval." Available: https://paperswithcode.com/sota/code-generation-on-humaneval, [online]. [Accessed: April. 26 2025].

[5] S. F. Institute, "Study: Large language models still lack general reasoning skills." Available: https://www.santafe.edu/news-center/news/study-large-language-models-still-lack-general-reasoning-skills, March, 2025. [Accessed: April. 12, 2025].

[6] Wikipedia, "Inference." Available: https://en.wikipedia.org/wiki/Inference. [Accessed: April. 20, 202].

[7] K. Sinha, S. Sodhani, J. Dong, J. Pineau, and W. L. Hamilton, "CLUTRR: A Diagnostic Benchmark for Inductive Reasoning from Text." Available: https://arxiv.org/abs/1908.06177, 2019. [Accessed: Dec. 23, 2024].

[8] I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar, "GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models." Available: https://arxiv.org/abs/2410.05229, 2024. [Accessed: Oct. 16, 2024].

[9] F. Shi, X. Chen, K. Misra, N. Scales, D. Dohan, E. Chi, N. Schärli, and D. Zhou, "Large Language Models Can Be Easily Distracted by Irrelevant Context." Available: https://arxiv.org/abs/2302.00093, January, 2023. [Accessed: April. 26 2025].

[10] K. Sinha, S. Sodhani, J. Dong, J. Pineau, and W. L. Hamilton, "Open Source CLUTRR/v1 Dataset." Available: https://huggingface.co/datasets/CLUTRR/v1, [online]. [Accessed: April. 22 2025].

[11] K. Sinha, S. Sodhani, J. Dong, J. Pineau, and W. L. Hamilton, "clutrr github." Available: https://github.com/facebookresearch/clutrr, 2019. [Accessed: Feb. 08, 2025].

[12] S. Minaee, T. Mikolov, N. Nikzad, M. C. R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey." Available: https://arxiv.org/html/2402.06196v2, February, 2024. [Accessed: April. 03 2025].

[13] B. L. Laura Ruis, "Unreasonable AI - The Difference Between Large Language Models (LLMs) and Human Reasoning." Available: https://arxiv.org/abs/2202.10745#:~:text=Systematic%20generalization%20is%20the%20ability,weakness%20of%20neural%20network%20learning., 2024. [Accessed: April. 07, 2025].

[14] adiuvo, "Unreasonable AI - The Difference Between Large Language Models (LLMs) and Human Reasoning." Available: https://www.adiuvo.org.uk/post/unreasonable-ai---the-difference-between-large-language-models-llms-and-human\protect\penalty\z@-reasoning, 2024. [Accessed: April. 08 2025].

[15] D. Moshman, "From inference to reasoning: The construction of r easoning: The construction of rationality ationality." Available: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1044&context=edpsychpapers, December, 2004. [Accessed: April. 15 2025].

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need." Available: https://arxiv.org/abs/1706.03762, 2023. [Accessed: Jan. 06, 2025].

[17] Cravath, "How ChatGPT understands context: The power of self-attention." Available: https://www.cravath.com/a/web/25fvkMDn6Q8MyAtaPpsLf2/8BaHMZ/cravath-tech-explainers-how-chatgpt-understands-context-022024.pdf, 2024. [Accessed: Jan. 08, 2025].

[18] L. Bouchard, "How LLMs Know When to Stop Talking?." Available: https://www.louisbouchard.ai/how-llms-know-when-to-stop/, May, 2024. [online]. [Accessed: April. 17 2025].

[19] K. Naminas, "LLM Hallucination: Understanding AI Text Errors." Available: https://labelyourdata.com/articles/llm-fine-tuning/llm-hallucination, January, 2025. [Accessed: April. 12 2025].

[20] CloudFlare, "Workers AI Landing Page." Available: https://developers.cloudflare.com/workers-ai/, [online]. [Accessed: April. 06 2025].

[21] HuggingFace, "Inference Providers." Available: https://huggingface.co/docs/inference-providers/en/index, [online]. [Accessed: April. 22 2025].

[22] G. Copilot, "Develop faster. Run anywhere." Available: https://www.docker.com/, [online]. [Accessed: April. 18 2025].

[23] Firebase, "Firebase, a platform designed to support you throughout your app development lifecycle." Available: https://firebase.google.com/, [online]. [Accessed: April. 22 2025].

[24] FireStore, "Use our flexible, scalable NoSQL cloud database,." Available: https://firebase.google.com/docs/firestore, [online]. [Accessed: April. 22 2025].

[25] HuggingFace, “sentence-transformers/all-MiniLM-L6-v2.” Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, [online]. [Accessed: April. 22 2025].

[26] HuggingFace, “sentence-transformers/stsb-roberta-base-v2.” Available: https://huggingface.co/sentence-transformers/stsb-roberta-base-v2, [online]. [Accessed: April. 22 2025].

[27] C.-Y. Lin, “ROUGE: A Package for Automatic Evaluation of Summaries.” Available: https://aclanthology.org/W04-1013.pdf, July, 2004. [Accessed: April. 14 2025].

[28] Haystack, “The Production-Ready Open Source AI Framework.” Available: https://haystack.deepset.ai/, [online]. [Accessed: April. 22 2025].

[29] Mozilla, “Using the Fetch API.” Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch, [online]. [Accessed: April. 17 2025].

[30] Huggingface, “Hugging Face Landing Page.” Available: https://huggingface.co/, [online]. [Accessed: April. 06 2025].

[31] Meta, “Introducing Llama 3.1: Our most capable models to date.” Available: https://ai.meta.com/blog/meta-llama-3-1/, July, 2024. [online]. [Accessed: April. 03 2025].

[32] Mistral, “Mistral 7B.” Available: https://mistral.ai/news/announcing-mistral-7b, September, 2023. [online]. [Accessed: April. 03 2025].

[33] Meta, “Llama 3.3.” Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/, [online]. [Accessed: April. 03 2025].

[34] React, “The library for web and native user interfaces.” Available: https://react.dev/, [online]. [Accessed: April. 18 2025].

[35] Javascript, “JavaScript (JS) is a lightweight interpreted programming language .” Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript, [online]. [Accessed: April. 18 2025].

[36] Tailwind, “A utility-first CSS framework.” Available: https://tailwindcss.com/, [online]. [Accessed: April. 18 2025].

[37] Python, “Programming Language.” Available: https://www.python.org/, [online]. [Accessed: April. 18 2025].

[38] FastAPI, “FastAPI framework, high performance, easy to learn, fast to code, ready for production.” Available: https://fastapi.tiangolo.com/, [online]. [Accessed: April. 18 2025].

[39] V. S. Code, “Your code editor. Redefined with AI.” Available: https://code.visualstudio.com/, [online]. [Accessed: April. 18 2025].

[40] P. Turchenko, “Choosing the Right IDE for JavaScript Development: Exploring the Differences.” Available: https://medium.com/@pavelturchenko89/choosing-the\protect\penalty\z@-right-ide-for-javascript-development-exploring-the-differences-f14dcde6aeea, July, 2023. [Accessed: April. 25 2025].

[41] A. Gupta, “20 Most Popular Python IDEs in 2025: Code Like a Pro.” Available: https://www.simplilearn.com/tutorials/python-tutorial/python-ide, April, 2025. [online]. [Accessed: April. 23 2025].

[42] G. Copilot, “Copilot is your AI pair programmer tool in Visual Studio Code.” Available: https://code.visualstudio.com/docs/copilot/overview, [online]. [Accessed: April. 18 2025].

[43] Docker, “Docker Compose.” Available: https://docs.docker.com/compose/, [online]. [Accessed: April. 06 2025].

[44] NPM, “Take your JavaScript development up a notch.” Available: https://www.npmjs.com/, [online]. [Accessed: April. 18 2025].

[45] Poetry, “Python packaging and dependency management made easy.” Available: https://python-poetry.org/, [online]. [Accessed: April. 18 2025].

[46] Gitlab, “Gitlab Version Control.” Available: https://about.gitlab.com/, [online]. [Accessed: April. 06 2025].

[47] Z. Li, Y. Cao, X. Xu, J. Jiang, X. Liu, Y. S. Teo, S. wei Lin, and Y. Liu, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.” Available: https://arxiv.org/abs/1908.10084, 2019. [Accessed: April. 07, 2025].

[48] HuggingFace, “Summary of the tokenizers.” Available: https://huggingface.co/docs/transformers/en/tokenizer_summary, [online]. [Accessed: April. 22 2025].

[49] L. Huang, “Measuring Similarity Between Texts in Python.” Available: https://www.simplilearn.com/tutorials/python-tutorial/python-ide, March, 2017. [online]. [Accessed: April. 23 2025].

[50] K. Kacprzak, “RoBERTa vs. BERT: Exploring the Evolution of Transformer Models.” Available: https://www.dsstream.com/post/roberta-vs-bert-exploring-the-evolution-of-transformer-models, April, 2025. [online]. [Accessed: April. 23 2025].

[51] S. Santhosh, “Understanding BLEU and ROUGE score for NLP evaluation.” Available: https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb, April 16, 2023. [online]. [Accessed: April. 23 2025].

[52] Pytest, “Python testing framework.” Available: https://docs.pytest.org/en/stable/, [online]. [Accessed: April. 24 2025].

[53] Jest, “Javascript testing framework.” Available: https://jestjs.io/, [online]. [Accessed: April. 24 2025].

[54] K. Cobbe, V. Kosaraju, M. Bavarian, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, “Training Verifiers to Solve Math Word Problems.” Available: https://arxiv.org/pdf/2110.14168v1, October, 2025. [Accessed: April. 12 2025].

[55] Wikipedia, “BERT Model Parameter Size.” Available: https://en.wikipedia.org/wiki/BERT_(language_model), [online]. [Accessed: April. 17 2025].