BDM - Second Deliverable

By

Kristóf Balázs
Nima Kamali Lassem
Stefanos Kypritidis

Professors:
Besim Bilalli
Achraf Hmimou Ham Man
Marc Maynou
Uchechukwu Fortune Njoku
Sergi Nadal

Barcelona, June 2025

Architecture design

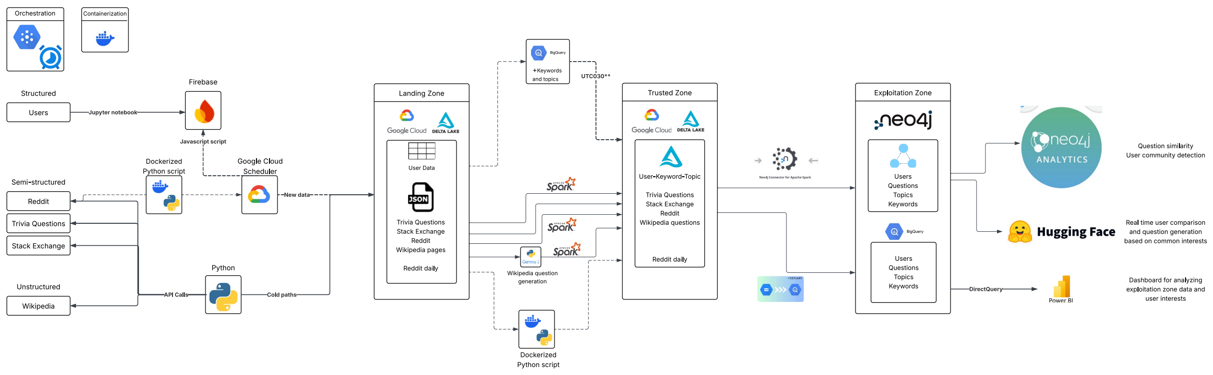


Figure 1: Architecture Diagram

Trusted Zone

Trivia Questions

For the trivia questions, we had to transform and cleanse them from a raw JSON format stored in the Google Cloud Storage (GCS) landing zone bucket of *trivia_questions* into a structured and trusted format saved as a Delta table. Here's a breakdown of the operations implemented:

1. Firstly, the raw trivia data was read from the GCS bucket in JSON format. We targeted the "results" array within the JSON structure, exploding it so each question becomes a row in the spark DataFrame. This step flattens the nested JSON data into a tabular format for easier processing.

2. We performed several data cleansing operations:

   - Deduplication based on the actual question text. Since the retrieval of questions would run in a loop to get more questions than the limit of the API, we retrieved multiple duplicated trivia questions.
   - Null filtering to ensure only records with both a non-null question and a correct answer are retained.
   - Text normalization, where questions are lowercased and stripped of special characters using regular expressions.
   - HTML character unescaping, to convert HTML entities (like *&amp;*) back into readable characters.

3. For feature engineering, we extracted keywords from the cleaned question text using the *summa NLP* library. These keywords are added as a new column to help with matching users to questions later on.

4. To align the trivia data with a broader schema, the script renames and drops certain columns. Finally, the end columns are:

Column Name	Description
id	Unique identifier (hash of question text)
title	The trivia question text (HTML-unescaped)
correct_answer	The correct answer to the question
incorrect_answers	List of incorrect answer options
topic	The category/topic of the question
keywords	Extracted keywords from the question text
source	Data origin, set to ”trivia”
subreddit	Placeholder for subreddit name (null)
author	Placeholder for content author (null)
upvotes	Placeholder for upvotes (null)
num_comments	Placeholder for number of comments (null)
created_utc_ts	Placeholder for creation timestamp (null)

Table 1: Final schema of the trivia questions in trusted zone

5. Finally, the cleaned and transformed DataFrame is written to a different GCS bucket, this time in Delta Lake format, under the Trusted Zone.

Wikipedia Questions

After thousands of Wikipedia articles with the fields ”title” and ”content” from the three different topics of ”Tech”, ”Sport”, and ”Film” were collected, they were fed into a question-and-answer generation pipeline. This pipeline, using the state-of-the-art efficient LLM, Gemma3, generated ”Reddit-like” question and answer pairs. Gemma3 is Google’s most recent efficient open-source LLM with a small size of only 3.3GB with 4 billion parameters. Using such a sophisticated model allowed us to generate question-and-answer pairs of high quality with little to no hallucinations. The efficient size of this LLM also allowed the generation of question-and-answer pairs in a timely manner. However, due to the sheer number of articles across all categories, all contents were limited to 3000 characters. The character limitation did not pose an issue at this stage since we were only to generate 2 question-and-answer pairs per article. Furthermore, multiple prompts were tested to find the best possible results, and eventually, the following prompt was chosen and used:

Prompt: Generate Reddit Q&As

You are an expert in {`category`} and a highly knowledgeable Wikipedia editor.
Based on the following Wikipedia article about `"{article['title']}"`, generate {`num_pairs`} pairs of questions and answers that might appear on `r/Ask{category.capitalize()}`.
The questions should:

- Be naturally curious and conversational in tone (like genuine Reddit questions)
- Focus on interesting facts or concepts mentioned in the article
- Range from beginner to more advanced knowledge levels
- Be diverse in their focus (not all about the same subtopic)

The answers should:

- Be informative and factually accurate based **EXCLUSIVELY** on the Wikipedia content
- Include specific details from the article (not general knowledge)
- Have a helpful, somewhat casual tone like a knowledgeable Reddit user
- Be 3–5 sentences in length
- **NOT** include any personal opinions or claims not supported by the article

Here's the article content: {`article['content'][:3000]`}
Format your response exactly as shown below:

```
Q1: [Question 1]
A1: [Answer 1]

Q2: [Question 2]
A2: [Answer 2]
```

ONLY return the formatted Q&A pairs, no introduction or additional text.

For the Wikipedia questions, we ingested data from a structured JSON file containing article-based QA pairs and transformed it into a trusted Delta table. Below is a step-by-step outline of the processing logic:

1. The raw Wikipedia QA data was read from a local JSON file. Each record included metadata and a list of question-answer pairs related to a specific article. The JSON structure was flattened by exploding the "qa_pairs" array, resulting in each QA pair becoming a distinct row within the spark DataFrame.

2. Several transformation and feature enrichment steps were applied:

- **Keyword extraction:** The *summa NLP* library was used to extract up to three relevant keyword phrases from the question text.
- **Field normalization and augmentation:** New metadata columns such as *source*, *id*, and placeholder fields (e.g., *subreddit*, *author*) were added to align with a common schema across datasets.

Column Name	Description
id	Unique identifier (hash of question text)
title	The Wikipedia question text
correct_answer	The corresponding answer from the Wikipedia article
incorrect_answers	Placeholder for incorrect answers (null)
topic	The article category or topic
keywords	Extracted keywords from the question text
source	Data origin, set to "wikipedia"
subreddit	Placeholder for subreddit name (null)
author	Placeholder for content author (null)
upvotes	Placeholder for upvotes (null)
num_comments	Placeholder for number of comments (null)
created_utc_ts	Placeholder for creation timestamp (null)

Table 2: Final schema of the Wikipedia questions in trusted zone

3. Finally, the cleaned and enriched DataFrame was written to a Google Cloud Storage bucket in Delta Lake format in the Trusted Zone.

Reddit Questions

Static Data

For the historical Reddit posts, we ingested raw JSON from the GCS landing-zone bucket `reddit_posts` into a trusted Delta table. Below is a breakdown of the generic cleaning and preparation steps we did (besides the keyword generation, which we discuss later):

1. We have to run this pipeline in a dedicated Conda environment to avoid version conflicts, which were caused by the ML libraries we needed for TF_IDF. See the following discussion on why this was needed.

2. We read the raw JSON files from GCS into a Spark DataFrame, flattening any nested structures so that each post's fields become top-level columns.

3. We remove duplicate posts based on the Reddit post `id` and filter out any records missing both a non-null title and non-null body text.

4. Text Normalization is needed for the keyword extraction in the script, we do not send this data to the trusted zone. We lowercase all text in the title and body, and strip punctuation and unescape HTML entities.

5. **Schema:** We cast each column to its intended type (e.g. `score` and `num_comments` to integer, `created_utc_ts` to timestamp), and rename fields to fit the common schema we defined (e.g. `selftext` to `body`). Moreover, we also add a static column `source = "reddit"`.

We performed keyword extraction using Spark's ML library. The keyword generation works as follows: first, it creates a text column that uses `selftext_clean` if non-empty or falls back to `title_clean`. Then the Spark ML Pipeline tokenizes on non-word characters, removes default English and custom stop-words, generates 2- and 3-grams, and unions all tokens into `all_grams`. Finally, CountVectorizer and IDF compute TF–IDF weights, and a UDF picks the top N terms by filtering out tokens under four characters, purely numeric tokens, and a hard-coded list of generic words.

Column Name	**Description**
id	Unique post identifier
subreddit	Name of the subreddit
title_clean	Normalized, HTML-unescaped title text
selftext_clean	Normalized, HTML-unescaped body text
author	Reddit username of the poster
score	Number of upvotes (INT)
num_comments	Number of comments (INT)
created_utc_ts	Original creation timestamp (UTC, TIMESTAMP)
source	Data origin, set to `"reddit"`

Table 3: Final schema of the Reddit questions in trusted zone

Streaming Data

The data cleaning and transformation follows the same logic as in the Static Data section, so we will not go over it again here. The main difference is that we do not perform keyword generation as in our application questions for hot topics only show up daily, so we only group by topic (it does not make sense to go to the keyword granularity). Moreover, we perform the transformation tasks with `pandas` and we upload the data in CSV format in this case as the daily data size does not (and will not) exceed a level where anything more advanced is needed in this case. If we had larger Volume and Velocity streaming data, we would have needed to spin up a Dataproc cluster (which we cannot do with the free credits available).

On the other hand, the proper orchestration between the streamed data to the landing zone and the trusted zone was one of the most challenging parts of P2. The architecture that we implemented after many trial and error attempts is summarized below:

- **Daily collector**
  - **Cloud Scheduler** (`0 0 * * *`, UTC) → HTTPS call to **Cloud Run** service *reddit-daily-top*.

  - Container scrapes Reddit API for the top daily question per subreddit and by upvote, writes raw JSON to `gs://*-landing-*/reddit/daily_top_posts/` *`YYYY-MM-DD`*`/reddit_daily_top_posts.json`.
  - Service account *ingest-sa*: `storage.objectCreator` → immutable landing.

- **Event fan-out**
  - **GCS notification** (prefix filter `reddit/daily_top_posts/`, event = OBJECT_FINALIZE) publishes a small JSON envelope to Pub/Sub topic *reddit-landing-new*.
  - **Eventarc trigger** (same region `europe-southwest1`) routes each Pub/Sub message as a CloudEvent to Cloud Run service *reddit-to-trusted*.

- **Trusted converter**
  - **Cloud Run** container receives the CloudEvent, downloads the raw object, flattens it with `pandas`, assigns a high-level topic (`TOPIC_MAP`), and uploads a CSV to `gs://*/trusted_zone/reddit/daily_top_posts/`*`YYYY-MM-DD`* `/reddit_daily_top_posts.csv`.

At the end of the pipeline, we have one clean, schema-fixed CSV per day in the trusted bucket, which can be used by other downstream jobs. The deployment instructions, along with the code and other files (e.g. Dockerfile) can be found under `RedditStream/TrustedZone` in our repository. The `deployment.md` file contains the whole setup, along with verification, both of which are lenghty and complex processes.

Stack Exchange Questions

Stack Exchange contains a large amount of historic data. It is therefore, important that we collect high-quality and highly related ones to our existing questions. The Stack Exchange data collection system gathers content from ten Stack Exchange sites, including `stackoverflow`, `softwareengineering`, `superuser`, `askubuntu`, `gaming`, `worldbuilding`, `movies`, `cooking`, `travel`, and `philosophy`.

Data Collection: The system uses the Stack Exchange API (`api.stackexchange.com/-2.3/search/advanced`) to collect top-voted posts. It fetches up to 25 pages per site with 100 items per page, collecting up to 25,000 posts per run. The system uses filter `!9_bDDxJY5` to capture detailed question data including metadata and voting information.

Data Structure: Each dataset contains site-specific metadata, collection timestamps, item counts per site, and complete API response data with pagination information.

Rate Limiting: The system includes 2-3 second delays between requests and 3-5 second delays between sites. It has a five-retry mechanism for failed requests with 60-second backoffs and automatically respects API backoff requests.

Schema: To ensure the compatibility of these data with the existing schema, we first rename the ″tags″ column to ″keywords″, then, we bin multiple topics together. This is done since the overall schema has 3 predefined topics; two sets of our topics here match with the overall schema, while the rest are grouped together as ″other″.

Category	**Topics**
Tech	stackoverflow, softwareengineering, superuser, askubuntu
Film	movies, gaming, worldbuilding
Other	cooking, travel, philosophy

Table 4: Binning of individual keywords into broader categories for schema compatibility

After this, the data is put into a Spark dataframe to prepare for uploading to the Trusted Zone. The schema of the data looks like the following:

Column Name	**Description**
source	Data origin, set to ″stack_exchange″
original_topic	The raw topic/category from the source site before binning.
topic	The cleaned or binned topic label (e.g., Tech, Film, Other).
id	Unique identifier of the post.
author	Username or identifier of the post's author.
title	The title or headline of the post.
keywords	An array of keywords or tags associated with the post.
upvotes	Number of upvotes received by the post.
num_comments	Number of comments the post has received.
created_utc_ts	Timestamp indicating when the post was created (in UTC).
body	The full body text/content of the post.

Table 5: Schema of Stack Exchange data in trusted zone

Trusted Zone Upload: Collected data is uploaded to the Google Cloud Storage bucket `group-1-`
`landing-lets-talk` under the `stack_exchange/` prefix. Files are stored as timestamped JSON files using the format `stackexchange_raw_YYYYMMDD_HHMMSS.json`. The process includes local backup creation, secure upload using service account authentication, and automatic cleanup after successful transfer.

Users

The three *Firestore* collections that describe our synthetic user base (see P1 for more details) are exported once per day by a Cloud Run job (*firebase-scheduled-export*) that is itself triggered by Cloud Scheduler at **00:00 UTC**. The export endpoint writes a Datastore-backup shard set to `gs://*-landing-*/users/`*`YYYY-MM-DD`*`/all_namespaces/`.

Because the export format matches BigQuery's *Cloud Datastore Backup* loader, we keep the ingestion layer extremely thin. A one-off "+ Create Table" in the BigQuery UI loads each collection pattern `gs://*-landing-*/users/*/*/kind_private_users/output`into a *staging* dataset. The loader converts the protobuf documents into native, typed BigQuery tables automatically. On the same dataset we register external BigLake tables (from the trusted zone of our four main sources for questions), that points to the Delta-Lake path for each source in the Trusted Zone. At this point, every data source needed to compute the `user_topic_keywords` mart is queryable with standard SQL.

The transformation logic lives in a single scheduled query named *refresh_user_topic_keywords*. The query can be found in our repository under `scripts_trusted_zone/User-to-Topic-to-Keyword-BigQuery.sql` Every night at 00:30 UTC (after the export has arrived) the scheduler:

1. merges the fresh shards into slowly-changing tables in `landing_eusw1`,
2. joins each user with the cross-source topic/keyword taxonomy (it assigns randomly 5 topics for each user from all available topics, and around 500 keywords within each topic for the same user),
3. and overwrites the denormalised table `landing_eusw1.user_topic_keywords`.

The end results is a Parquet file in our trusted zone (under `/YYYY-MM-DD/` as it updates daily), where we make the most out of our synthetic data generated in P1 (as we now assign **real** keywords and topics to our synthetically generated users). We later use this data in our exploitation zone as it has the following structure:

Field	**Type**	**Description**
`user_id`	STRING	Identifier inherited from *Firebase*.
`name`	STRING	Public display name of the user.
`topic`	STRING	One of the topics available from our four main sources.
`keywords`	ARRAY<STRING>	Up to 500 sampled keywords related to the `topic`.

Table 6: Schema of the `user_topic_keywords` Parquet file in the Trusted zone.

Exploitation zone

In the exploitation zone of our pipeline, we utilized Neo4j, a graph database management system, to model and store complex relationships among users, questions, topics, and keywords. Our goal was to represent the semantic and contextual links within the dataset using a graph structure to support richer insights and advanced querying capabilities.

We chose Neo4j for our exploitation zone primarily because of its efficient implementation of the property graph model, which is well-suited to the needs of our Let's Talk application. This simpler structure benefits from the schema-optional and lightweight nature of property graphs. Neo4j's use of the Cypher query language allows for fast and efficient graph traversal, which is critical for delivering real-time conversation starters and identifying user interests with minimal delay. These capabilities give Neo4j a significant performance edge over traditional semantic graph databases, making it ideal for high-frequency, user-facing interactions.

Additionally, Neo4j offers a highly scalable and developer-friendly environment, especially when deployed via Neo4j Aura on Google Cloud. This fully managed service enables us to easily scale our infrastructure without the burden of manual setup or maintenance. As our user base and data volume grow, we can simply adjust our subscription to access more compute power and storage. Features like automated backups, high availability, and integrated Graph

Data Science tools further support our growth. These capabilities not only help us maintain system responsiveness but also enable advanced features like community detection and similarity scoring, which were used in the exploitation tasks and are essential for enhancing user engagement.

Architecture and Tools

To build the exploitation zone of our pipeline, we employed Apache Spark in combination with the Neo4j Spark Connector to efficiently ingest and process data from our trusted Zone, which uses Delta Lake storage. The Spark application script was configured to load and transform questions from various curated sources, including Trivia, Wikipedia, Stack Exchange, Reddit, and users (containing topics and keywords). Spark's distributed processing capabilities, combined with the Neo4j connector, enabled us to transform and stream this data directly into a graph database model within Neo4j aura.

Graph Model Design

Our graph schema was carefully designed to capture the semantic structure and interrelationships present in the data. The schema consists of four main types of nodes: *Question* nodes represent individual questions sourced from the knowledge bases; *Keyword* nodes capture key terms extracted from the content or associated with users; *Topic* nodes categorize questions and keywords under broader thematic umbrellas; and *User* nodes represent individuals and their expressed interests. These nodes are interconnected by several relationship types. This schema supports rich, flexible querying of the data using graph traversal and pattern matching.

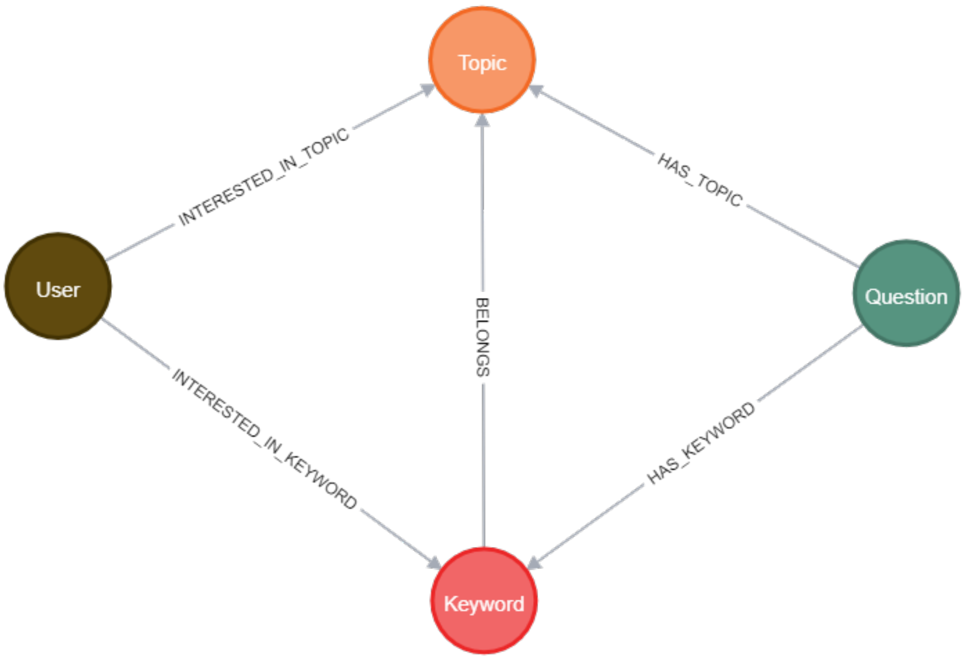


Figure 2: Neo4j Schema

Implementation Details

After configuring a Spark session tailored to our requirements, including support for Delta Lake and Neo4j, as well as integration with Google Cloud through service account credentials, the appropriate JAR files of the Neo4j connector to apache spark were loaded to enable connectivity and data writing capabilities. Once configured, Spark was used to load the question datasets from Delta table format into spark dataframes. Also, Keywords were exploded from nested arrays to normalize the many-to-one relationship between questions and keywords, facilitating clean graph modeling.

We then proceeded to create the core node types: Question, Keyword, and Topic, using Spark's DataFrame API. Relationships between these nodes were established using key-based mapping strategies. For large datasets such as Stack Exchange, we had to employ data repartitioning and in-depth the *.repartition(1)* to prevent Spark deadlocks during write operations to Neo4j.

User-related data was then incorporated by aggregating users' interests from their trusted zone records. We applied collect_set and flatten functions to generate unique sets of keywords and topics per user. This is done since previously the users data is normalized, meaning each row represents a single (user_id, topic, keywords) combination. Finally, all nodes and relationships were exported to the Neo4j database using the org.neo4j.spark.DataSource format. We used overwrite mode throughout the export process to ensure clean, consistent data loads and maintain the integrity of the graph model with each execution.

Final Outcome

The result of this process is a rich, semantically connected knowledge graph stored in Neo4j. This graph structure enables advanced exploration of how users, topics, questions, and keywords relate to each other. It forms a robust foundation for various applications, explained later. Furthermore, the use of Neo4j's Cypher query language allows for intuitive, expressive querying of complex patterns.

Consumption tasks

Dashboard

Based on the up-to-date data that is stored in BigQuery, we created views that are the base for our dashboards. We connected to these views using Power BI in Direct Query mode. Direct Query uses a live connection to the data, meaning that every time a dashboard is opened by any user, Power Query will make a query to the data to get the current version. Moreover, the views are standard BigQuery views, so any change in the base tables is reflected live in Power BI Direct Query. We defined the following views for our dashboards:

1. `v_all_content`: Unions content from Reddit, StackExchange, Trivia Q&A and Wikipedia into one consolidated table.

2. `v_user_kw_exploded`: Takes each user's `keywords` array and unnests it so we get one row per (user, keyword).

3. `v_user_summary`: Aggregates, per user, how many distinct topics they follow and the total number of keywords.

All the work related to the creating the dashboard can be found under the `dashboard` folder within our repository. The dashoard itself can also be accessed online directly. A snapshot of the dashboard is visible on Figure 3.

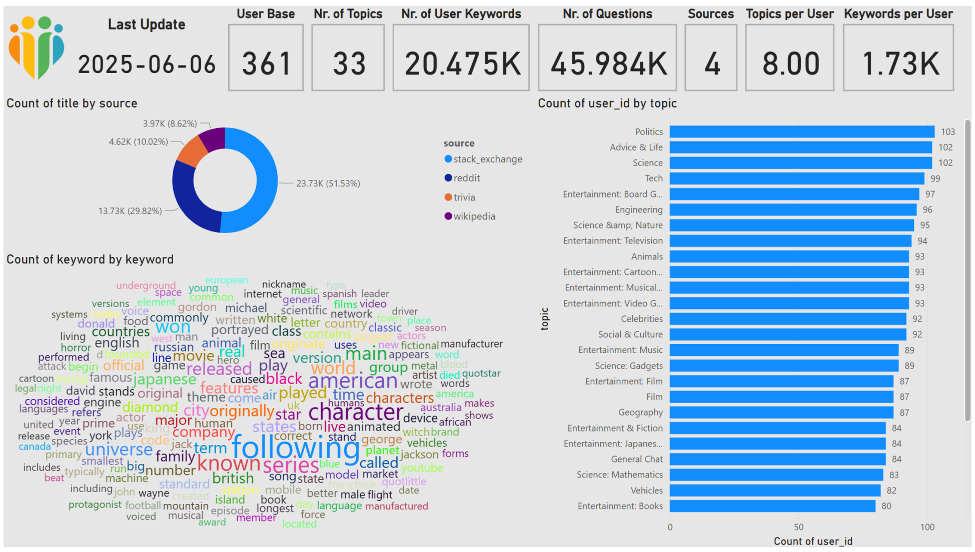


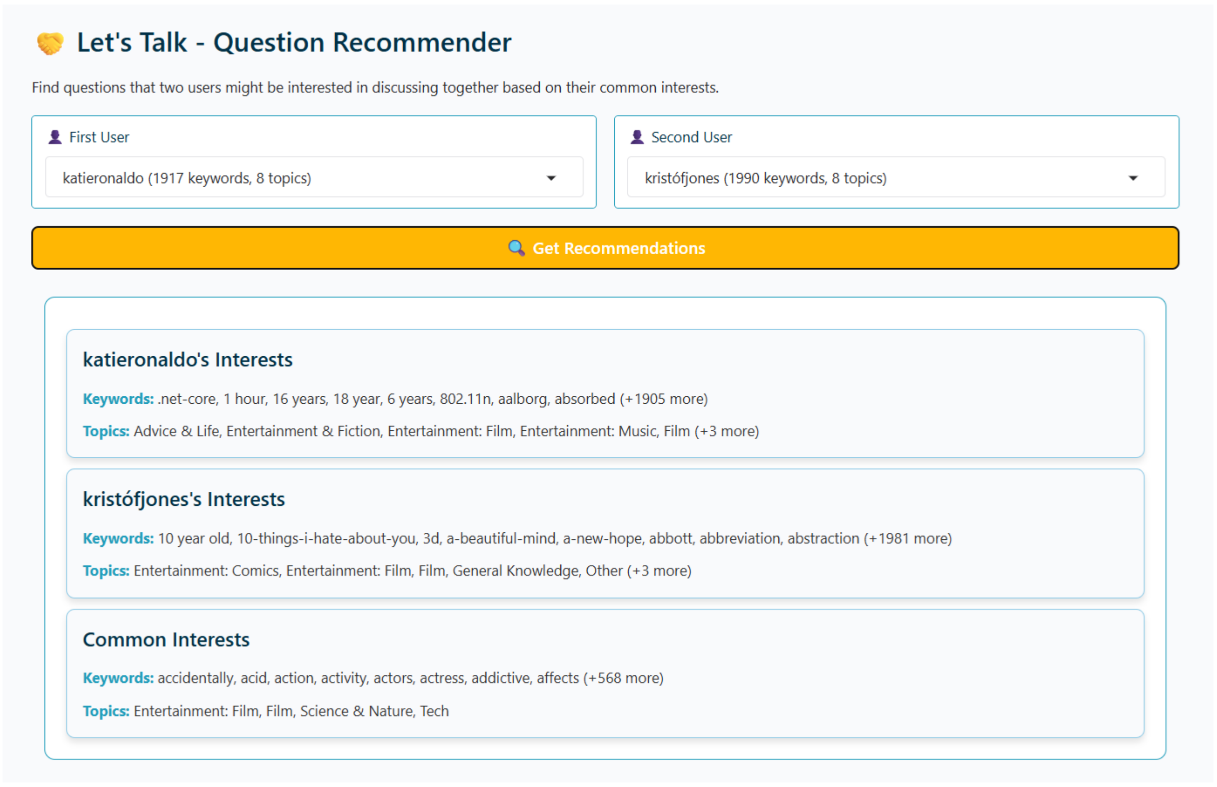
Figure 3: Snapshot of our Power BI Dashboard

The work we did within Power BI can be found in the `Let'sTalkDashoard.pbix` file. We will not go over it here as it is beyond the scope of this report.

Proof of Concept

We built and deployed a working version of our graph-based question recommendation system to test how well it works in practice. The application can be accessed through our demo website and is hosted on Hugging Face Spaces, showing how "Let's Talk" works in real situations.

The system implements our graph design and provides a simple web interface where users can pick two people from our database and get personalized question suggestions based on the interests they have in common. The application shows how graph search methods can effectively find shared topics between users and then suggest relevant questions from different sources like Stack Exchange, Reddit, Wikipedia, and Trivia.



Let's Talk - Question Recommender

Find questions that two users might be interested in discussing together based on their common interests.

katieronaldo (1917 keywords, 8 topics)

kristófjones (1990 keywords, 8 topics)

katieronaldo's Interests

Keywords: .net-core, 1 hour, 16 years, 18 year, 6 years, 802.11n, aalborg, absorbed (+1905 more)

Topics: Advice & Life, Entertainment & Fiction, Entertainment: Film, Entertainment: Music, Film (+3 more)

kristófjones's Interests

Keywords: 10 year old, 10-things-i-hate-about-you, 3d, a-beautiful-mind, a-new-hope, abbott, abbreviation, abstraction (+1981 more)

Topics: Entertainment: Comics, Entertainment: Film, Film, General Knowledge, Other (+3 more)

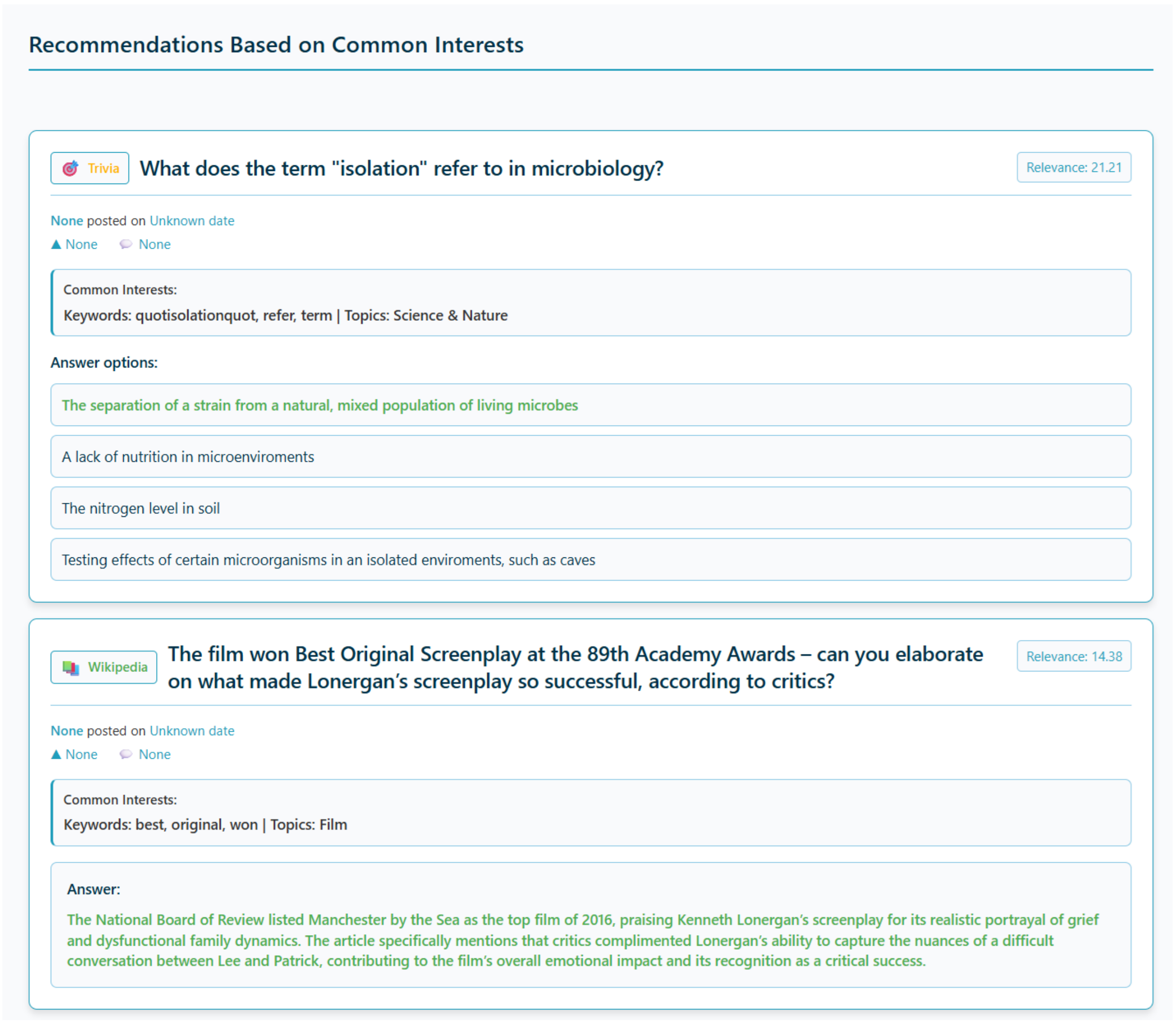
Common Interests

Keywords: accidentally, acid, action, activity, actors, actress, addictive, affects (+568 more)

Topics: Entertainment: Film, Film, Science & Nature, Tech

Figure 4: System Interface - Finding Common Interests

We built the system using Gradio as our web framework, which lets us create prototypes quickly and works well with our Python-based Neo4j database. Gradio is good for data science projects because it lets us focus on the graph analysis instead of complex web programming. The application connects directly to our Neo4j Aura cloud database, showing that our graph-based approach can handle larger workloads and performs well in cloud environments. Using Hugging Face Spaces also gives us continuous deployment, so users can keep using the old version while we update to a new one. Hugging Face Spaces also makes it easy to add this app to our main website.


Recommendations Based on Common Interests

Trivia What does the term "isolation" refer to in microbiology? Relevance: 21.21

None posted on Unknown date
▲ None 💬 None

Common Interests:
Keywords: quotisolationquot, refer, term | Topics: Science & Nature

Answer options:

The separation of a strain from a natural, mixed population of living microbes

A lack of nutrition in microenviroments

The nitrogen level in soil

Testing effects of certain microorganisms in an isolated enviroments, such as caves

Wikipedia The film won Best Original Screenplay at the 89th Academy Awards – can you elaborate on what made Lonergan's screenplay so successful, according to critics? Relevance: 14.38

None posted on Unknown date
▲ None 💬 None

Common Interests:
Keywords: best, original, won | Topics: Film

Answer:
The National Board of Review listed Manchester by the Sea as the top film of 2016, praising Kenneth Lonergan's screenplay for its realistic portrayal of grief and dysfunctional family dynamics. The article specifically mentions that critics complimented Lonergan's ability to capture the nuances of a difficult conversation between Lee and Patrick, contributing to the film's overall emotional impact and its recognition as a critical success.

Figure 5: System Interface - Question Suggestions

The user interface, shown in Figure 4, has a clean design where users can select two people from dropdown menus that show real users from our database. After making selections, the system shows a complete analysis of interests, displaying each user's individual interests in keywords and topics, plus their shared interests. The recommendation system then shows a list of questions with relevance scores, source information, and detailed data including who created them, when they were made, and how much engagement they received. The relevance score uses a multi-part algorithm that combines interest matching with time factors and randomization. The base score gives higher weights (2.0) to interests that both users share and lower weights (1.0) to interests that match only one user. This base score gets improved with time relevance that favors newer questions, and a randomization factor `(0.6 + 0.8 * rand())` plus source-specific random boosts to make sure we get different recommendations from all question sources (Stack Exchange, Reddit, Wikipedia, Trivia) as shown in Figure 5.

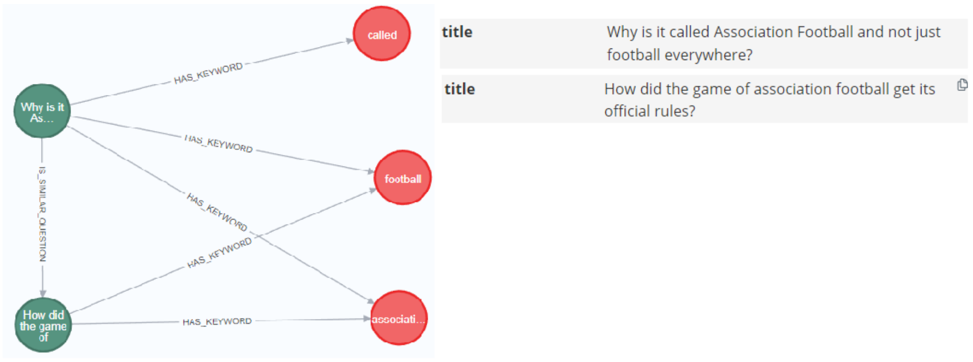
The working system proves our idea that graph-based recommendation systems can provide better personalization than traditional methods. The application shows good performance, handling complex graph searches and returning recommendations in under 20 seconds, even when looking at users with thousands of interest keywords. This performance testing is important for our goal of helping spontaneous conversations, where response time directly affects how users experience the system.

The implementation also shows how flexible our graph model is by handling different question types and sources smoothly. Questions from Stack Exchange display with formatted code and technical details, trivia questions show multiple-choice formats with highlighted correct answers, and Wikipedia questions include complete answers. This variety shows how our graph structure successfully handles different types of conversation content while keeping consistent relationship patterns for recommendation algorithms.

Graph Analytics

We implemented two key graph analytics techniques—node similarity and community detection—as part of our exploitation tasks to enhance the intelligence and personalization of our application. Due to limited free credits from Google Cloud Services, we used the minimal configuration for our cloud Neo4j Aura instance, which did not support the Aura Graph Analytics Serverless feature. Instead, we ran the graph analytics tasks locally using a sampled dataset and a separate Neo4j database instance. The portability of Neo4j's graph analytics syntax meant that the same code would run seamlessly on higher-tier instances, ensuring scalability. These analytics tasks could bring immense value to our system, and one of the primary reasons for choosing a graph database was its natural fit for these operations.

The node similarity algorithm was used to measure the overlap between questions based on shared keywords. We created a graph data science (GDS) projection that included only question and keyword nodes and then applied the Node Similarity algorithm using the overlap similarity coefficient. This metric, which prioritizes the intersection relative to the smaller set of keywords, is more effective than Jaccard in cases with uneven keyword counts. A threshold of 0.2 was set to establish meaningful similarity edges between questions. These new edges allow the app to surface related questions dynamically, especially after users interact with specific questions. This form of question recommendation could leverage graph structures to create a more fluid, intelligent user experience and help keep conversations engaging and on-topic.



title	Why is it called Association Football and not just football everywhere?
title	How did the game of association football get its official rules?

Figure 6: Question similarity example

In our second graph analytics task, we focused on community detection among users with similar interests. We first computed user-to-user similarity using the overlap coefficient on shared interest topics, adding weighted edges for similarities above 0.1. This similarity graph became the foundation for running the Louvain algorithm, which identified tightly-knit user clusters by optimizing modularity. The resulting communities could allow us to offer new features like interest-based group suggestions, meetups, and more personalized conversation starters in the future. From a product perspective, this opens opportunities for community-building and user engagement, and from a business standpoint, it enables more accurate and high-value targeting for advertisements. These enhancements show the strong strategic value of using a graph database like Neo4j, where analytics and relationships are core to the data model.

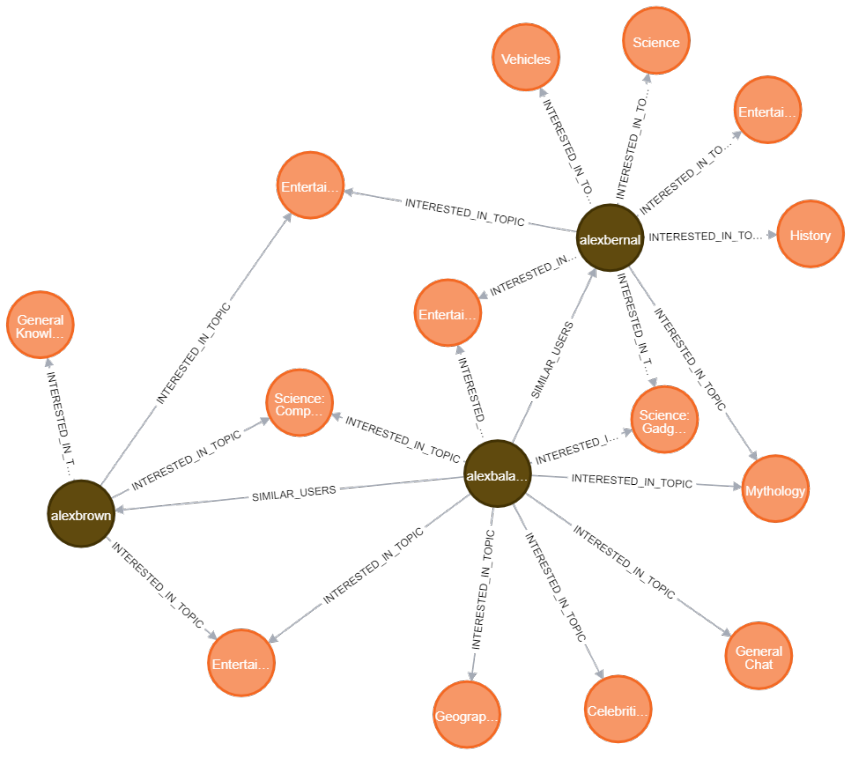


Figure 7: Community detection example

Approach and technology

Data Governance

Access Control

Every role and setting was set up within the Google Cloud Console. Each zone is isolated in its own bucket or dataset; only the team that needs to mutate it can do so. Pipelines run as dedicated service accounts ("ingest-sa", "transform-sa"). This is a best practice done to keep human keys out of production. BigQuery and Neo4J are the only places analysts can query; they never touch raw objects, protecting data integrity and cost. Lastly, it is important to mention that we have a dedicated service account, so the roles currently do not have any members as we all work using the same service account credentials. The exact settings applied in each case are summarized in Table 7 and Table 8.

Zone	Admins	Data Eng.	Data Sci.	Data Analyst
Landing	Owner	Read / Write	–	–
Trusted	Owner	Read / Write	Read	–
Exploitation	Owner	Load	Read / Write	Read

Table 7: Who can touch what inside each zone.

Service / Resource	Admins	Data Eng.	Data Sci.	Data Analyst	Service Acct.
Cloud Storage – Landing Zone	storage.admin	storage.admin	–	–	ingest-sa: objectCreator
Cloud Storage – Trusted Zone	storage.admin	storage.admin	objectViewer	–	transform-sa: objectAdmin
Pub/Sub topic *reddit-landing-new*	pubsub.admin	pubsub.editor	–	–	ingest-sa: publisher
Cloud Run (ELT/ETL jobs)	run.admin	run.invoker	–	–	scheduler-sa: invoker
BigQuery datasets (Exploitation)	bq.admin	bq.dataEditor	bq.dataEditor	bq.dataViewer	transform-sa: dataEditor
Artifact Registry (Docker)	ar.admin	ar.writer	–	–	ci-sa: writer
Neo4J connection[1] (Exploitation)	owner	readWrite	readWrite	readOnly	graph-sa: readWrite
Cloud Scheduler (jobs)	cs.admin	cs.jobEditor	–	–	scheduler-sa: jobRunner
Dashboards (Power BI)[2]	datasetOwner	–	connectionUser	connectionUser	–

Table 8: IAM bindings used for the main managed components. Role names follow the *google.cloud* predefined roles (bq = BigQuery, ar = Artifact Registry, cs = Cloud Scheduler).

Metadata Catalog for Various Zones

A metadata catalog was created to document all data assets across the Landing, Trusted, and Exploitation zones of the google cloud services. The primary objective was to provide a struc-

[1]Managed Neo4J Aura or GCE VM; mapped to IAM custom roles: *readWrite*, *readOnly*.

[2]Power BI connects through a BigQuery service principal; Analysts and Data Scientists need the `bigquery.connectionUser` role.

tured overview of the datasets, offering traceability, discoverability, and clarity on data ownership and quality, which ensures the different zones remain organized, scalable, and reviewable. Each data asset is represented as a dictionary containing rich metadata, while all those dictionaries are exported as a JSON file for storage.

The metadata entries contain key attributes for each dataset, including: *source* (e.g., Reddit, Stack Exchange, Trivia etc.), *zone* (Landing, Trusted, Exploitation), *file_name* (dataset identifier), *path_or_target* (storage location such as a Google Cloud Storage bucket or Neo4j endpoint), *owner* (responsible team), *description* (summary of the dataset's purpose), *update_frequency* (e.g., "daily", "none"), *tags* (for categorization), and *data_quality_score* (a numerical measure of dataset quality). These descriptors help ensure that data assets are well understood and governed throughout their lifecycle.

To ensure this catalog is centralized and accessible, the file is then uploaded to Google Cloud Storage landing zone so that it is easily accessible and readable. Saving it inside a deltalake would be unnecessary since this file only contains limited information and should not be partitioned. The file is saved inside:
`gs://group-1-landing-lets-talk/metadata_catalog/catalog.json`. This makes the metadata catalog available for integration with other tools, dashboards, or pipelines.

Appendix

Access to our GitGub

The source code for this project is available at the following GitHub repository: **GitHub Repository:** `https://github.com/UPCLetsTalk/BigGroup`. To test the application, use the following credentials:

- **Username:** testupc
- **Password:** Y7d!qR#3mVzP@9xTasdfsdfd

Hugging Face Spaces - Gradio

The source code for the prototype of this project is publicly accessible at:

- **Repository:** https://huggingface.co/spaces/NimaKL/LetsTalk/tree/main
- **Gradio App Code:** https://huggingface.co/spaces/NimaKL/LetsTalk/blob/main/app.py
- **Test the Prototype:** https://huggingface.co/spaces/NimaKL/LetsTalk/