Dialogue2Data (D2D): Transforming Interviews into Structured Data for Analysis - Proposal Report

Sienko Ikhabi, Dominic Lam, Wangkai Zhu, Yun Zhou

Table of contents

Executive Summary

Surveys are a vital tool for organizations to capture direct user feedback, understand user needs and priorities, measure satisfaction, and gauge public opinion. Analyzing survey results helps guide meaningful product and service improvements. Surveys are an invaluable set of tools that help organizations know how well they are performing and help shape where they need to go.

Our capstone partner, Fathom, provides client organizations with commercial survey analytics solutions to fulfil this business need. The traditional surveys that their clients have used in the past can however be rigid and impersonal, often resulting in low engagement and limited insights. To address this, Fathom is enhancing its analytics platform to support open, conversational surveys that use natural language, adaptive flows, and personalized questions. This flexible approach improves response quality, completion rates, and the depth of insights gathered. Nonetheless, extracting structured insights from open-ended responses presents a significant challenge. Unstructured data must be processed and standardized to enable aggregation and analysis at scale. Our capstone project tackles that challenge by leveraging advanced natural language processing (NLP) techniques. Specifically, we propose building a model - potentially based on a transformer architecture or Large Language Model (LLM) - to analyze user interview transcripts against a set of guideline queries representing organizational subject interests.

The goal of our system is to capture nuanced semantic relationships, making it well-suited to match free-text responses with key organizational themes. Our system will transform diverse, unstructured survey responses into structured summaries that can integrate seamlessly into Fathom's existing analytics pipeline while acurately capturing the information collected during the interviews.

By enabling automated, scalable analysis of open surveys, this solution will expand the volume and variety of surveys that Fathom can process. Ultimately, it will help their clients uncover richer, faster insights, enhancing strategic decision-making and improving the overall user experience.

Project Introduction

1. Motivation

Large volumes of open-ended interview transcripts contain rich, nuanced information that can provide key takeaways to the survey moderators. However, extracting structured insights from these unstructured transcripts—especially when aligning them to specific research or survey questions—remains a labor-intensive and error-prone process. Our partner's current workflow relies on manually prompting large language models (LLMs) such as ChatGPT to answer a set of pre-defined guideline questions, with extensive human oversight required to validate the outputs.

While this approach can be effective, it is neither scalable nor efficient. As datasets grow in size and complexity, the need for a more automated, robust, and precise solution becomes evident. Our goal is to reduce the human workload while preserving—or even improving—answer quality and reliability.

Objective

Our project aims to develop a natural language processing (NLP) pipeline that automatically matches and extracts relevant information from interview transcripts in response to a predefined set of guideline questions. This system seeks to reduce reliance on human-in-the-loop processes and make the output more consistent and structured for downstream analytics.

2. Data Format: Input and Output

- **Input:**
  - `.txt` files containing unstructured, conversational interview transcripts.
  - `.csv` files containing a fixed set of guideline questions (23 in total) to which responses must be mapped.
- **Output:**
  - A structured `.csv` file where:
    - Columns represent the guideline questions.
    - Rows correspond to the extracted responses for each interviewee.
    - Cells contain either direct or inferred answers from the transcript, matched and formatted for clarity.

This structured output is designed to enable further analysis, such as dashboard visualization, response comparison, and statistical summary.

3. Pipeline Overview

Outline workflow

In Figure 1 we show the outline of the proposed process.

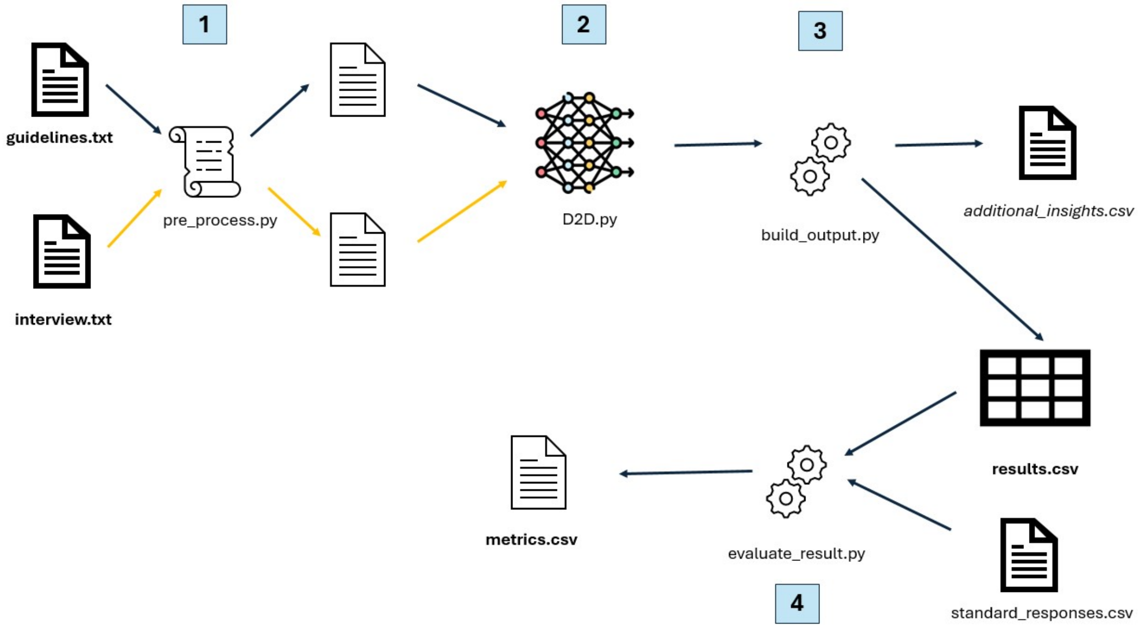


Figure 1: Workflow Outline

1. In the first stage, we perform text pre-processing on the guidelines and the interview transcripts using `pre_process.py`.
2. The cleaned files are then input into the core D2D process, `D2D.py`, to perform the main processing part of our solution. The aim is to generate an appropriate answer to each guideline question from each interview.
3. We then build the main output of our process, the `results.csv` grid. *As as stretch goal, we might also create the file* `additional_insights.csv`*. This file will include any responses that were in the interview but were not relevant to any specific guideline question but could be useful information. This is still vaguely defined and will only be revisited once the main solution is developed.*
4. During development of the solution, we will use custom-made transcripts for which we have the expected `standard_responses.csv` also custom-made. In the final step, we will evaluate the performance of our current model by comparing the output generated against these expected responses and produce metrics to be reported in the file `metrics.csv`. We cover this process in the evaluation section.

We propose to deliver the final solution as a Python-based CLI tool that our capstone partner can incorporate in their current process.

Pseudocode of the core process

In Figure 1 we presented the core D2D.py as a single block. At a high-level, we can envisage this process to be broken down as per the pseudocode below:

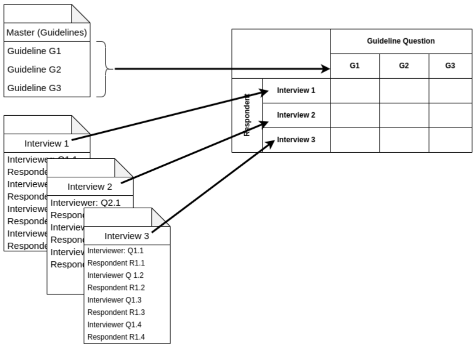
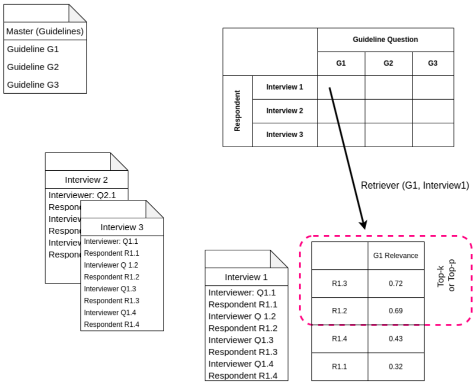
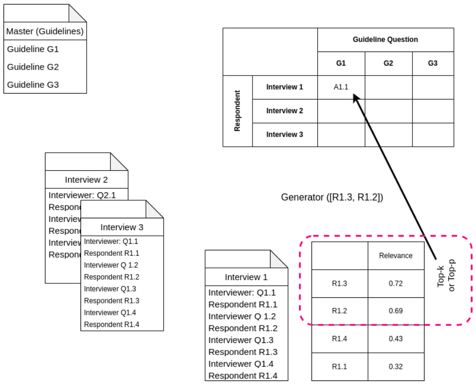


Figure 2: Step 1



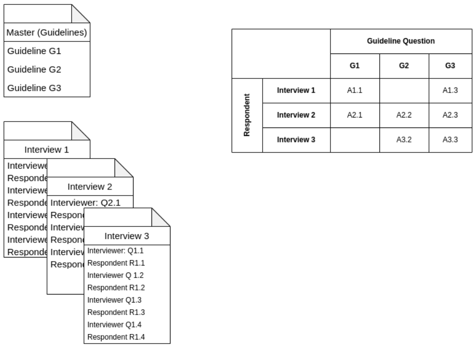
	G1 Relevance
R1.3	0.72
R1.2	0.69
R1.4	0.43
R1.1	0.32

Figure 3: Step 2



	Relevance
R1.3	0.72
R1.2	0.69
R1.4	0.43
R1.1	0.32

Figure 4: Step 3



	Guideline Question		
	G1	G2	G3
Interview 1	A1.1		A1.3
Interview 2	A2.1	A2.2	A2.3
Interview 3		A3.2	A3.3

Figure 5: Step 4

1. Generate an $N \times M$ grid, with N rows corresponding to the number of interviews we have and M columns corresponding to the number of guideline questions.
2. Next we iterate through each guideline question and each interview (each cell in the $N \times M$ grid). In this step, we aim to retrieve answers from the transcript that are relevant to the current guideline question. Note that it is possible (based on the settings) to have 0 or more relevant responses.

3. In the third step, we use the set of top retrieved answers to generate an appropriate response to be populate for the current cell.
4. We repeat this process until we have processed all the cells in the grid

The above is a simple pseudocode version. In the next sections, we will cover the exact data science techniques we plan to use in detail. In addition, the looping process will be optimised for efficiency. For instance, we will use vectorization to process multiple interviews and guideline questions concurrently rather than looping through each one at a time.

Data Science Techniques

1. Proposed Solutions

Traditional Retrieval-Augmented Generation (RAG)

Our baseline solution is built on the Retrieval-Augmented Generation (RAG) framework, which is well-suited for question-answering tasks involving long and unstructured text, such as interview transcripts.

In a traditional RAG setup, the system is split into two parts: a retriever and a generator.

- **Retriever**:
  We first segment the interview transcript into smaller, manageable question-and-answer (QA) pairs. Then, we use a pretrained transformer model such as MiniLM or DPR to convert both the QA pairs and the guideline questions into vector embeddings. These embeddings are compared using cosine similarity to find the top-k (or top-p) QA pairs most relevant to each guideline question. This enables us to narrow down the transcript to only the most relevant content.
- **Generator**:
  The top matches are passed to a large language model (LLM) like ChatGPT, which is then prompted to extract the main ideas or answers from the selected content. This LLM serves as the generator in our pipeline.

This approach allows us to maintain control over what the LLM sees, thereby reducing the likelihood of hallucinated answers. It is also modular: the retriever and generator can be swapped or fine-tuned independently.

We have already implemented this baseline RAG system, and initial results have been well-received by our capstone partner. Even without the generator component, the retriever alone provides helpful insights by surfacing the most relevant parts of the transcript for each question.

However, one limitation of Traditional RAG is that it always returns a fixed number of top results. This means that if the true answer is not ranked within the top-k, it may be missed entirely. Additionally, the generator assumes that the retrieved content is relevant, which can sometimes reduce accuracy.

Self-RAG: Self-Reflective Retrieval-Augmented Generation

To address the limitations of the traditional RAG approach, we propose an extension called Self-RAG, which incorporates an additional step of self-evaluation using an LLM.

The core idea of Self-RAG is to validate the quality of the retrieval before final generation. After the retriever identifies the top-k QA pairs for a given guideline question, we use an LLM to ask:

"Are these QA pairs actually relevant to the question?"

If the answer is yes, we proceed as usual and ask the model to extract the main points. If not — for instance, if the retrieved content is only loosely related or off-topic — the system automatically adjusts its retrieval parameters (e.g., increasing or decreasing k or p) and re-runs the retrieval step.

This dynamic loop gives the system a way to adaptively improve its own performance without human intervention. It can lead to more accurate and focused outputs, especially in cases where the first set of matches is suboptimal.

However, Self-RAG introduces some added complexity. It requires additional LLM calls for self-assessment, and the logic to iterate on retrieval adds to the overall system design. Despite this, we believe that the trade-off is worthwhile, particularly if it helps minimize the need for human oversight — one of our partner's primary goals.

Self-RAG remains transparent, just like Traditional RAG, but adds flexibility and quality control at a key step in the pipeline.

2. Evaluation

In order to evaluate the performance of Dialogue2Data comprehensively, we plan to adopt RAGAS (Retrieval-Augmented Generation Assessment Scores)[1] framework as the core idea of the evaluation, with enhancement experience from Daedalus v4[2].

Core Evaluation Framework - RAGAS

RAGAS is an automated evaluation framework tailored specifically for the RAG architecture. It incorporates LLM-assisted judgments and provides scoring from multiple perspectives within the RAG pipeline, and we can obtain five key metrics from RAGAS[1]:

- **Faithfulness**: The degree of consistency between the generated answers and the corresponding retrieved information.
- **Answer Relevance**: The usefulness of the generated answers for their questions.
- **Context Precision**: The proportion of retrieved information that is actually necessary for generating the answer.
- **Context Recall**: The proportion of information needed to generate the answer that was successfully retrieved.

- **Answer Correctness**: The correctness of the generated answer, compared to the golden sample.

Limitations of Applying RAGAS to Dialogue2Data

- **Limitation in assessing answer style**: Our partner expects the responses to follow a consistent style to facilitate downstream analytics tools in structuring data and generating statistics. However, the current metrics do not assess aspects such as sentence completeness, length control, or clarity of expression.
- **Bias introduced by model preferences**: RAGAS evaluates the performance with the support of LLMs, such as ChatGPT, and Claude. These LLMs may prefer templated answer styles, and the bias from the model preference may affect the objective of the scoring.
- **Lack of feedback mechanism**: RAGAS evaluates by numerical-score output, without corresponding support information. This affects the interpretability of the evaluation process and makes obstacles for quick manual validation.
- **Dependency of golden reference for correctness scoring**: The answer correct in RAGAS relies on the comparison to the golden samples. Due to the limited golden samples, answer correctness may not be applied at scale.

Enhancement by Daedalus v4

According to the above limitations, Daedalus v4 provides some feasible strategies[2].

- **Consistency metric**: We plan to introduce another metric, consistency, to measure if the answer follows a consistent style, thereby indicating the ease with which downstream tools can perform data structuring for statistical purpose.
- **Prompt engineering**: To reduce the bias from LLMs' favor in templated answers, we will use prompt engineering to set the evaluation criteria.
- **Feedback mechanism**: To make the scoring process more transparent, we will add a feedback module (also known as an iteration loop) that provides supporting information for each score. This feedback can also be used to improve the answers, eventually forming an iterative loop to enhance answer quality.
- **Stratified sampling**: We will apply a stratified sampling method to design golden samples that are small in number, high in quality, and broad in speaking style.

Summary of Evaluation

The following table is the summary of our evaluation framework based on RAGAS with enhancement of Daedalus v4.

Table 1: Table 1. Summary of our evaluation framework based on RAGAS and Daedalus v4.

Module	Description
Metrics in RAGAS	Faithfulness, Relevance, Precision, Recall, Correctness (when golden answer available)
Metric extended from Daedalus v4	Consistency
Prompt engineering	To reduce LLM bias
Feedback mechanism	To generate explanation
Stratified sampling strategy	To ensure comprehensive testing

Timeline

Deliverables and Deadlines

- Proposal Presentation: Friday, May 2, 2025, 14:00–16:00
- Proposal Report Draft: Tuesday, May 6, 2025, 12:00
- Proposal Report Final: Friday, May 9, 2025, 17:00
- Runnable Draft to Mentor: Monday, June 9, 2025, 16:00
- Final Presentation: Thursday, June 12, 2025
- Draft Report to Mentor: Wednesday, June 18, 2025, 16:00
- Final Report and Project to Partner/Mentor: Wednesday, June 25, 2025, 12:00

Weekly Milestones

Week 1: April 28 – May 4, 2025 (Project Setup and Proposal Presentation)

- Repository set up with initial files and documentations.
- Proposal presentation delivered.
- Synthetic data preparation started (5 transcripts).
- Proposal report draft 50% complete.

Week 2: May 5 – May 11, 2025 (Proposal Report and RAG Implementation)

- Proposal report draft and final submitted.
- RAG approach script completed and tested.
- Unit tests passing (80% coverage).
- Documentation completed with baseline details.

Week 3: May 12 – May 18, 2025 (Self-RAG Exploration and Implementation)

- Self-RAG approach script completed and tested.
- Documentation completed with Self-RAG details.
- Evaluation preparation with baseline approach.

Week 4: May 19 – May 25, 2025 (Evaluation Development)

- Evaluation completed on both approaches
- Optimization (Hyperparameters/Models Tuning on both approaches
- Decided which approach to develop
- Documentation updated with metrics (Authentic and Synthetic Data)

Week 5: May 26 – June 1, 2025 (Structured Output and End-to-End Testing)

- Finalized structure output
- Documentation finalized for output and testing
- Develop and test Command-Line Interface (CLI)
- Pass end-to-end tests (including CLI interactions)
- Final presentation slides (>= 50%) completed

Week 6: June 2 – June 8, 2025 (Runnable Draft, Final Report and Presentation)

- Bug reduced
- Runnable draft and CLI completed
- Final report (>= 50%) completed
- Final presentation slides completed

Week 7: June 9 – June 15, 2025 (Runnable Draft and Final Presentation)

- Runnable draft submitted (June 9, 16:00)

- Final Presentation delivered (June 12/13)
- Draft report (>= 50%) complete
- Bugs reduced to < 5 critical issues

Week 8: June 16 – June 20, 2025 (Draft Report and Finalization)

- Final report draft submitted (June 18, 16:00)
- Package finalized with tests passing
- Documentation completed.

Post-Week: June 23 – June 25, 2025 (Final Submission)

- Final report (final version) submitted (June 25, 12:00)
- Package submitted (June 25, 12:00)

References

[1] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, “RAGAS: Automated Evaluation of Retrieval Augmented Generation,” arXiv preprint arXiv:2309.15217, Sep. 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.15217

[2] DS-Daedalus Team, “DS-Daedalus v4: Increments and Updates,” Internal Presentation Report, Mar. 2025