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

GitHub Repository

Sienko Ikhabi, Dominic Lam, Wangkai Zhu, Yun Zhou

Table of contents

Executive Summary

Surveys help organizations gather user feedback, understand needs, measure satisfaction, and guide product or service improvements. Fathom, our capstone partner, offers commercial survey analytics solutions that support open, conversational interviews using natural language, adaptive flows, and personalized questions. This flexible format improves response quality but poses challenges in extracting structured insights from the unstructured interview transcripts data.

To address this, we propose a system leveraging advanced natural language processing (NLP), potentially using a transformer architecture or large language model (LLM). The goal is to minimize the human effort required by accurately capturing semantic relationships and matching free-text responses to key organizational questions. This system will convert varied survey inputs into structured summaries that fit into Fathom's current analytics pipeline, enabling scalability, in-depth analysis and faster, more informed decision-making for their clients.

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 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.

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

3. Pipeline Overview

Outline workflow

Figure 1 shows the proposed pipeline.

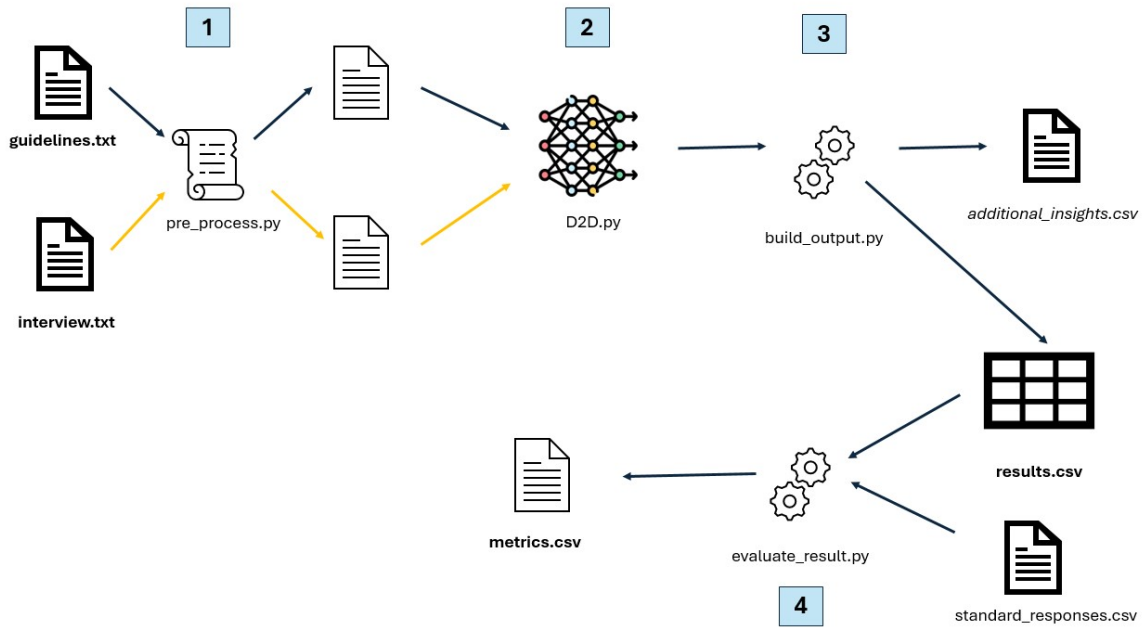

Figure 1: Workflow Outline

1. Perform text pre-processing on the guidelines and the interview transcripts using `pre_process.py`.
2. Feed the cleaned files into the core `D2D.py` script to generate an appropriate answer to each guideline question from each interview.
3. Build final `results.csv` output. *Stretch goal: Create* ***`additional_insights.csv`*** *containing responses in the interview that are useful information but do not answer any specific guideline question.*

We propose to deliver the 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 script to be broken down as per the pseudocode below:

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. Iterate through each guideline question and each interview (each cell in the $N \times M$ grid). In this step, we retrieve answers from the interview transcript that are relevant to the current guideline question. It is possible to have 0 or more relevant responses.
3. Use the set of top retrieved responses to generate an appropriate answer to be populated for the current cell.
4. Repeat this process for all the cells.

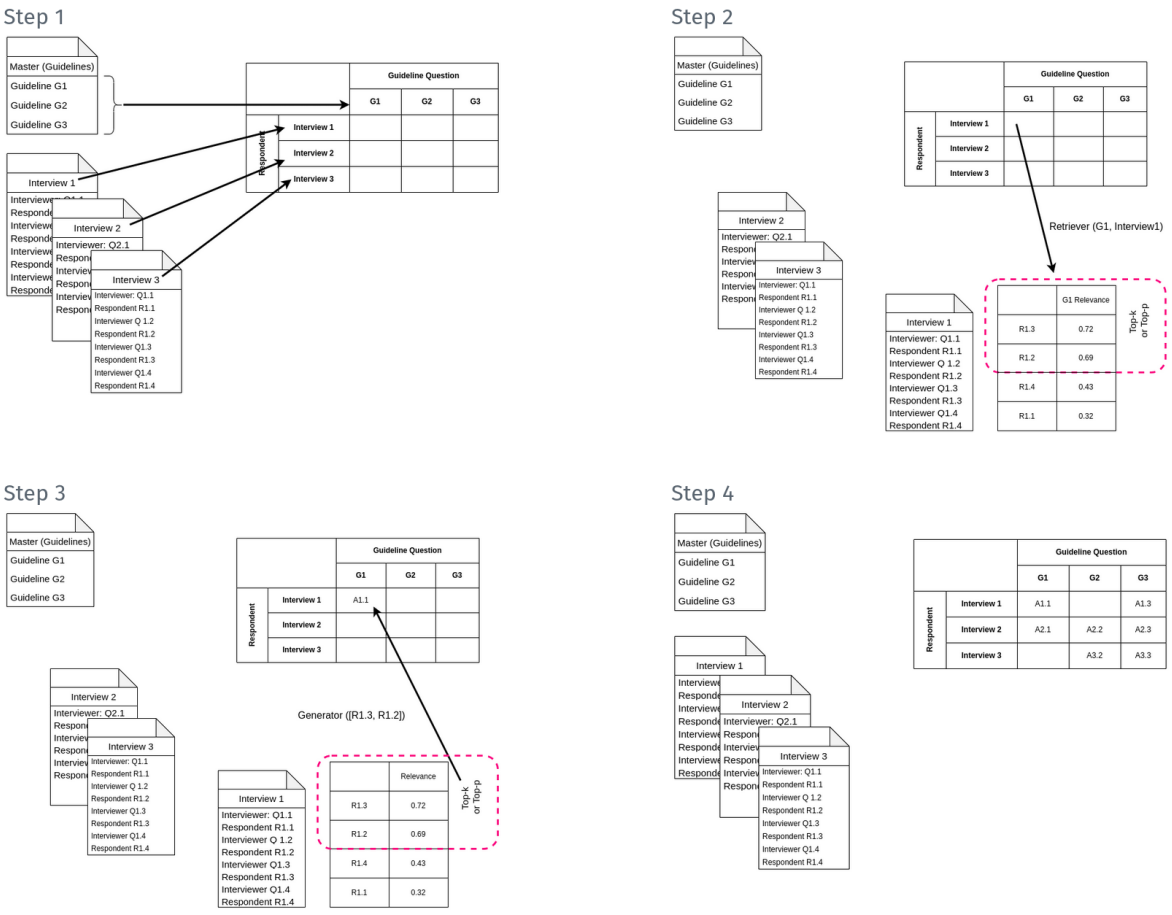

Figure 2: D2D Pseudocode

Data Science Techniques

1. Data Description and Technique Appropriateness

Use of Data Science Techniques

Retrieval-Augmented Generation (RAG) and Self-Reflective Retrieval-Augmented Generation (Self-RAG) will transform interview transcripts into structured data by extracting entities and sentiments, will then be evaluated using RAGAS and Daedalus for accuracy.

Data Description

Interview transcripts (observational units) are text data guided by questions.

Example:

Speaker	Text
Interviewer	"Thanks for joining me today. What's your name and where are you located?"
Participant	"My name's Camila, outside Jackson, Mississippi."
Interviewer	"Great to meet you, Camila! Who lives with you at home?"
Participant	"I live with my husband, dogs, and mom."

Appropriateness of Data

Transcripts are ideal for RAG/Self-RAG, as LLMs handle natural language for entity extraction.

Difficulties

- **Filler Words**: Greetings and fillers (e.g., "sure thing") may obscure key content, requiring filtering.
- **Volume**: Lengthy transcripts demand efficient RAG indexing.

2. Proposed Solutions

RAG (Baseline):

Our baseline method features a **Retrieval-Augmented Generation (RAG)** pipeline, adapted to our project, to obtain answers from interview transcripts

- **Retriever**: The transcript is broken down into question-answer (QA) pairs. QA pairs as well as guideline questions are translated into embeddings using a pre-trained transformer model. These embeddings are compared using **cosine similarity** to find the **top-k** (or **top-p**) most relevant matches.
- **Generator**: The relevant matches are sent to an LLM like **ChatGPT** as context, to extract the core meaning from the answers that are relevant to the guideline question. This setup **reduces hallucinated responses** by providing context to the LLM.

But RAG has drawbacks: if the information lies outside the top matches. The generator assumes the relevance of retrieved content, at the risk of compromising accuracy.

Self-RAG:

As a response to the vulnerability of Traditional RAG, **Self-RAG** has an additional self-evaluation layer with an LLM.

After it retrieves the **top-k QA pairs**, the LLM evaluates their relevance to the guideline question. If found relevant, pass along. If not, it adjusts **k** or **p** of the retriever, and attempts again. If it fails to retrieve any relevant content after multiple trials, it returns `[No relevant response found]`.

This auto-assessment cycle enhances accuracy and focus, particularly in situations where first-time retrievals are sub-optimal.

3. Evaluation

Our evaluation is inspired by the RAGAS framework, which provides five main metrics—faithfulness, relevance, precision, recall, and correctness (when golden answers are available) to evaluate the model performance comprehensively[1]. However, RAGAS has some limitations when applied to Dialogue2Data, which we aim to address using strategies proposed in Daedalus v4[2].

- **Limitation in assessing answer style**: Our partner expects consistent response styles for downstream analytics, but current metrics lack assessment of sentence completeness, length, and clarity. We will add a consistency metric to measure stylistic alignment.

- **Bias introduced by model preferences**: RAGAS relies on LLMs, whose preference for templated answers may introduce bias and affect scoring objectivity. We apply prompt engineering to reduce the bias.
- **Lack of feedback mechanism**: RAGAS provides scores without supporting information, reducing interpretability and making it harder for quick manual validation. We will introduce a feedback module that presents the supporting information behind scoring.
- **Limited golden samples**: Correctness evaluation requires golden answers, which are limited. We adopt stratified sampling to create a small but high-quality golden sample set that includes fewer than 25 examples but covers a broad range of topics and linguistic styles.

Table 2: Summary of our evaluation framework based on RAGAS with enhancement of 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	To create a small but diverse golden sample set

Timeline

The detailed timeline is included in Appendix A.

Week 1: Apr 28 – May 4, 2025

- Set up repository, deliver proposal presentation, draft 50% of proposal report.

Week 2: May 5 – 11

- Finalize proposal report. Split into RAG and Evaluation groups.

Week 3: May 12 – 18

- RAG: Test Self-RAG script, document.
- Evaluation: Extend module, add 10 golden samples.

Week 4: May 19 – 25

- Integrate efforts, evaluate approaches, optimize, select approach.

Week 5: May 26 – Jun 1

- Finalize output, CLI, documentation, pass end-to-end tests.

Week 6: Jun 2 – 8

- Complete runnable draft, CLI.

Final: Jun 9 – 12

- Submit draft (Jun 9), present (Jun 12), submit final report, package (Jun 12).

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

Appendix A: Detailed Timeline

Weekly Milestones

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

- Set up repository with initial files and documentation.
- Deliver proposal presentation.
- Complete 50% of proposal report draft.

Week 2: May 5 – May 11, 2025 (Proposal Report and Parallel Development)

- Submit proposal report draft and final.
- Split team into two groups: **RAG Development** and **Evaluation Development**.
- **Group 1: RAG Development**
  - Complete and test RAG approach script.
  - Pass unit tests with 80% coverage.
- **Group 2: Evaluation Development**
  - Develop basic evaluation module using RAGAS.
  - Create 2 golden samples for evaluation.

Week 3: May 12 – May 18, 2025 (Continued Parallel Development)

- **Group 1: RAG Development**
  - Complete and test Self-RAG approach script.
  - Complete documentation with Self-RAG details.
  - Prepare evaluation with baseline approach.
- **Group 2: Evaluation Development**
  - Extend evaluation module using Daedalus system.
  - Create 10 more golden samples for evaluation.

Week 4: May 19 – May 25, 2025 (Evaluation and Approach Selection)

- Reunite team to integrate RAG and evaluation efforts.
- Complete evaluation on both approaches.
- Optimize hyperparameters and models for both approaches.
- Decide which approach to develop further.
- Update documentation with metrics for authentic and synthetic data.

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

- Finalize structured output.
- Finalize documentation for output and testing.
- Develop and test Command-Line Interface (CLI).
- Pass end-to-end tests, including CLI interactions.
- Complete 50% of final presentation slides.

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

- Reduce bugs.
- Complete runnable draft and CLI.
- Complete 50% of final report.
- Complete final presentation slides.

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

- Submit runnable draft on June 9, 16:00.
- Deliver final presentation on June 12 or 13 (TBC).
- Complete 50% of draft report.
- Reduce bugs to fewer than 5 critical issues.

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

- Submit final report draft on June 18, 16:00.
- Finalize package with passing tests.
- Complete documentation.

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

- Submit final report on June 25, 12:00.
- Submit package on June 25, 12:00.