Thesis Dissertation

Polarization Analysis and Detection Mobile Application

Panayiotis Liotatis

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2025

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Polarization Analysis and Detection Mobile Application

Panayiotis Liotatis

Advisor: Demetris Paschalides, PHD Candidate

Supervisor: Dr. George Pallis

Thesis submitted in partial fulfilment of the requirements for the award of Bachelor's degree in Computer Science at University of Cyprus

May 2025

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. George Pallis, and my advisor Demetris Paschalides for their continuous guidance, valuable feedback, and encouragement throughout the development of this thesis.

Abstract

Polarization is a significant issue, particularly prevalent in contemporary times due to the abundance of mass media, social media, news articles, and related sources. During periods of substantial division over critical societal issues, individuals often struggle to maintain clarity of thought, as news media may present information biased toward the political party they support or by which they are sponsored. Paschalides et al. [1] address this challenge with POLAR, a holistic framework designed for modelling polarization in news media. My thesis builds upon this framework by extending it into a mobile application for analysis and detection of polarization. This application, named *Polarization Detector*, is a first-of-its-kind, on-demand polarization analysis and detection application, giving the user the control of extracting the polarization level in the articles they consume, so that they can be better informed. The idea for this application is to allow the user to share with it any articles they browse, group the articles shared as they want, and analyse the batch of articles for polarization. The result, is a summary of polarization existence in the batch, mentioning the topics most polarized, and the opposing groups that create this polarization. The important part is that when analysing the polarization of each batch, the backend doesn't execute the whole POLAR model from start to end. Instead, POLAR executes once on a large corpus of articles (as a case study around the context of U.S. elections 2024) and stores the knowledge extracted in a graph database. When the user wants to analyse polarization in a batch, the polarization knowledge is queried from the database with the results summarised using an LLM and presented to the user.

Contents

Chapter 1

Introduction

Polarization is a significant issue, particularly prevalent in contemporary times due to the abundance of mass media, social media, news articles, and related sources. During periods of substantial division over critical societal issues, individuals often struggle to maintain clarity of thought, as news media may present information biased toward the political party they support or by which they are sponsored. In the United States specifically, partisan divisions have steadily deepened over the past several decades, with the ideological gap between the "left" and "right" continually widening [2]. This issue is considered crucial for the smooth functioning of society. As this divergence in views and beliefs increases, it can foster significant dislike – and in some instances, animosity – towards opposing groups – term known as "affective polarization" [3]. This, in turn, may lead to widespread mistrust, challenges to democratically elected authorities, and a general antipathy towards the political system and its elected officials.

The severity of this phenomenon has motivated various attempts to mitigate its effects, ranging from crowdsourcing content reviews and presenting diverse viewpoints side-by-side, to educational games designed to enhance critical understanding. However, much of the existing research has concentrated on narrow contexts, often focusing on single issues or reducing complex dynamics to a binary left-versus-right dichotomy, particularly on platforms such as Twitter. Such a restricted focus hinders a comprehensive understanding of how polarization manifests across different topics, public figures, and media outlets. An essential step towards addressing this challenge arguably involves the capacity to model the issue effectively, mapping its principal points of contention. Addressing the need for a broader mapping and understanding of polarization and its engagement with various entities, coalitions, and conflicts, Paschalides et al. introduced the POLAR framework. POLAR is an unsupervised, domain-agnostic system that processes large collections of news articles to model polarization without requiring prior domain-specific knowledge. It constructs a knowledge graph of entities (such as political

figures and organisations) by analysing their co-occurrence with supportive or oppositional language, thereby enabling the determination of inter-entity attitudes. This process helps reveal clusters of like-minded entities (termed “fellowships”) and opposing groups (“fellowship dipoles”). Consequently, the framework yields data outputs that, upon correct processing, enable the extraction of this polarization knowledge. While POLAR provides a powerful and comprehensive framework for analysing polarization at a large scale, there is no opportunity for individual users to extract such knowledge and apply it for their everyday news consumption.

This thesis, contributes by building upon the POLAR framework, utilising it as the foundational basis for a novel polarization detection application. As a case study, POLAR was applied to be executed with context related to the U.S. elections 2024 – a context recognised for its strong partisan divisions. The data extracted, following intermediate cleaning processes, were stored in a graph database to enable efficient knowledge retrieval. Subsequently, a backend system was developed. This system, by executing a minimal set of POLAR commands (specifically entity and noun phrases extraction) adapted for batch processing of smaller article sets, can leverage the previously stored knowledge to query the polarization index of a new batch. This approach is significant as it minimises the execution of the time-consuming POLAR pipeline, replacing substantial portions of it with rapid database queries. The results from these queries are not left in their raw state. They are later processed to generate an English prompt for a Large Language Model (LLM), which then provides a summarisation of the detected polarization.

To further contribute to the need of citizens better informed about polarization, the development of a mobile application was identified as a suitable approach, due to its everyday use, being the main point of consumption for news, and for its ease of sharing articles. Its main objective is to enable users browsing articles to easily share them with the application, and with a simple interaction, the application then analyses and detects polarization – using the backend system mentioned, thereby informing the user about any partisan characteristics of the content. Additionally, the application includes different features, for example organising articles easily into batches, for better user experience and control over their articles.

Chapter 2

Related Work

This chapter reviews existing literature relevant to understanding and tackling political polarization, especially concerning news media and online discussions. It starts by defining political polarization and the factors that drive it. Then, it explores the difficulties in measuring polarization and how it appears in media, including the influence of hyperpartisanship and misinformation. Finally, the chapter looks at various computational and digital methods designed to detect and reduce media polarization,

2.1 Understanding Political Polarization

Political polarization is broadly understood as the widening ideological distance or deepening social separation between different political groups [4]. This is not a single concept; researchers identify forms like *ideological polarization*, which is about differing policy views, and *affective polarization*, marked by rising dislike and distrust between opposing partisan groups [5]. Affective polarization has notably increased, especially in the U.S., where individuals often view those from other parties negatively, even if their policy disagreements are not so far apart [3]. Such animosity often stems from social identity, where party loyalty can overshadow other considerations, leading to less cross-

party cooperation and a decline in civil public discussion [6]. The widespread impact of polarization includes obstructing democratic debate, fuelling online conflicts, weakening social unity, and making it harder to reach consensus in times of crisis [7].

Certain social-cognitive processes tend to reinforce polarization. Group polarization, for example, shows how like-minded groups can become more extreme in their views after internal discussions [8]. This is supported by homophily, the tendency to associate with similar people, and confirmation bias, our inclination to favour information that confirms what we already believe [9]. These factors create echo chamber effects, especially online, where constant agreement can push moderate views to the extreme [10]. As this happens, opposing sides may see each other as threats, making compromise difficult and straining democratic values like mutual respect [11].

2.2 The Challenge of Measuring Polarization

While understanding polarization is important, accurately measuring it is a persistent challenge [12]. Polarization is complex, appearing differently across various contexts, such as social media discussions which are influenced by platform algorithms, user actions, and network effects [12].

Much of the work in political science has used bimodality – the presence of two distinct peaks in how beliefs are distributed – as a key sign of polarization [12]. Various statistical methods try to measure how far a distribution deviates from having a single peak (unimodality) to quantify polarization [12]. A comparative study by Di Martino et al. reviewed five common measures, including the Bimodality Coefficient and Hartigan's Dip Test, noting their individual strengths and weaknesses [12]. For instance, some measures might misinterpret skewed distributions or be overly sensitive to the distance between opinion clusters [12].

This review of measurement techniques shows that no single metric works perfectly for all situations. Applying concepts like bimodality accurately can be tricky with large datasets where manual checks are not feasible, requiring dependable automated methods. This underscores the ongoing need for better tools to effectively analyse complex patterns of polarization.

2.3 Polarization in Media and Online Discourse

Political polarization is a significant aspect of modern public discussion, especially in countries like the United States where partisan gaps have widened noticeably [13] [14] [15]. This is driven by factors like ideological sorting in political parties and the strong influence of today's media.

A key factor in media polarization is hyperpartisan news, which involves articles with strong, often extreme, biases supporting a particular political stance or party [16]. This type of reporting often uses sensationalized, one-sided language, valuing ideological loyalty over objective reporting. The spread of such news can worsen polarization among readers and reduce trust in established government and news sources. Hyperpartisan news is generally considered a form of misinformation and often shares traits with fake news. Linguistically, these articles might use more adjectives and adverbs, emotionally charged words, and a sensational style in both their content and titles. Hyperpartisanship essentially involves an exaggerated use of various biases – like spin, ad hominem attacks, or framing – to aggressively push one viewpoint [16].

The digital environment has further complicated this landscape. Traditional partisan media (like certain news channels or websites) can reinforce polarization by exposing audiences to one-sided content [17]. Social media platforms add another layer, raising concerns about echo chambers and filter bubbles that might limit exposure to diverse viewpoints [18]. While the exact impact of filter bubbles is still debated [19], social media undoubtedly helps spread hyperpartisan content and misinformation quickly. Misleading information often thrives in polarized settings, where people are more likely to believe and share content that confirms their biases, leading to a cycle that deepens these divides [20]. Interestingly, efforts to counter this by showing people opposing views on social media have sometimes backfired, causing individuals to become even more set in their original opinions [21]. This shows that digital media's role is complex, with algorithms, misinformation, and online interaction styles all contributing to current levels of polarization. Studies on platforms like YouTube have also noted ideological segregation, though some centrally-positioned content might encourage more interaction between different viewpoints [12].

2.4 Automated Detection and Mitigation Strategies

The issues of media polarization and hyperpartisanship have led to significant research into automated methods for detecting them and potentially reducing their impact. These tools often aim to predict a text's political leaning, identify if it's hyperpartisan [16], or pinpoint specific topics that are highly polarized [22].

Maggini et al. offer a systematic review of automated text-based methods for finding hyperpartisan news articles [16]. These methods often analyse linguistic cues (like n-grams or sentiment), semantic information from word embeddings or topic models, stylistic features like readability, and sometimes metadata like publisher details. A variety of machine learning models have been used, from traditional ones like SVMs and Random Forests to advanced deep learning models like CNNs, RNNs, and Transformers (e.g. BERT). While Transformers perform well, the use of very large language models (LLMs) for this specific task is still an emerging area [16].

Beyond classifying whole articles, some research focuses on identifying specific polarized topics within news. He et al. proposed PaCTE (Partisanship-aware Contextualised Topic Embeddings), a method to find such topics in partisan news [22]. PaCTE involves using Latent Dirichlet Allocation (LDA) for initial topic discovery, then fine-tuning a language model like BERT to recognise partisanship. It represents a news source's ideological stance on a topic using specialised "corpus-contextualised topic embeddings" and measures polarization by the cosine distance between these embeddings from different sources. This technique aims for a detailed, topic-level understanding of polarization using modern NLP.

Other digital solutions also try to address these issues. Some use crowdsourcing to identify misleading content or rate news source trustworthiness [23]. Platforms like AllSides present news from multiple perspectives to help users see media bias [24]. Educational games such as "Bad News" use "inoculation theory" to help users spot misinformation tactics [25]. There are also efforts to create dialogues between opposing groups, though these have had varied success [26].

2.4.1 Identifying the Gap: The Need for User-Centric Real-Time Analysis

The existing research shows a strong and developing field focused on understanding, measuring, and detecting political polarization and hyperpartisanship. Quantitative measures [12] help define polarized distributions, and many automated methods, including advanced deep learning techniques like PaCTE [22], work to identify biased content or polarized topics. Digital tools also offer media literacy education and exposure to different views.

However, many advanced analytical systems are primarily for researchers or large-scale operations, often needing significant technical resources or expert handling. While they improve our understanding, there's a clear need for tools that directly empower individual news readers. Specifically, accessible, user-driven applications are lacking that can offer on-demand insights into the polarization of news content people encounter daily. Most current solutions don't provide a mobile-friendly, real-time way for users to submit articles, group them as desired, and get an immediate, summarised analysis of polarization regarding specific topics and entities. This thesis seeks to fill this gap by using a powerful, pre-calculated model of polarization from a comprehensive framework to provide such a tool.

2.5 The POLAR Framework

Addressing the limitations of existing tools in comprehensively mapping and understanding polarization across various topics, entities, coalitions, and diverse conflicts, Paschalides et al. [1] developed POLAR. POLAR is described as a holistic pipeline designed for modelling polarization within collections of news articles and for identifying the most polarized topics within a given discourse. In essence, the framework automates a form of content analysis to delineate opposing factions and the core issues that divide them. The data model and definitions established by POLAR are central to its methodology. Key concepts defined by Paschalides et al. include:

i. *Polarization*: Drawing from social science, as a societal process where a group splits into two or more sub-groups with opposing and conflicting beliefs. Within their

model, this manifests when entity attitudes towards specific topics shift to more extreme positions, leading to this group division.

ii. *Entity*: Any real-world subject of interest, such as a person, organisation, location, political group, or specific event, each uniquely identified and classified by type.

iii. *Sentiment Attitude Graph (SAG)*: In this graph, entities are depicted as nodes, and the connections (edges) between them are weighted by the sentiment (e.g. positive, neutral, or negative) expressed in their interactions.

iv. *Fellowship*: Identified within the SAG as a distinct cluster of entities predominantly interconnected by positive sentiment and relationships. It signifies a group of like-minded entities that generally support one another or share common stances.

v. *Fellowship Dipole*: Models two opposing fellowships and the predominantly negative or conflictual relationships that exist between these two groups. It is a core structure used to represent instances of polarization and conflict.

vi. *Polarizing Topic*: An issue or theme of discussion that evokes strong disagreement and opposing attitudes from the entities within a fellowship dipole. The degree to which a topic is polarizing is quantified by POLAR using a "*polarization index*", which measures the extent of this attitudinal opposition.

vii. *Polarization Index (pi)* [27]: Quantifies the degree to which a topic is polarized**.** This index operates on a scale from 0 (no polarization) to 1 (perfect polarization). The underlying principle is that a topic is perfectly polarized when the opinions expressed about it are split into two distinct groups of roughly equal size that hold strongly opposing average views. Consequently, the calculation of this index for a given topic primarily considers two factors derived from the set of sentiment attitudes expressed by entities within the dipole's fellowships: first, the balance in the number of positive versus negative attitudes, and second, the magnitude of the difference between the average positive attitude and the average negative attitude. A high polarization index value therefore indicates that the topic sharply divides the relevant entities into two substantial,

opposing camps. This allows POLAR to generate a ranked list of topics based on their degree of divisiveness within the analysed discourse.

These definitions and the underlying data model from the POLAR framework are of fundamental importance to this thesis. The entire methodology for storing the extracted polarization knowledge, the process of identifying polarization within new data from the batch, and specifically, the criteria used to define a “polarized batch” of articles, are directly built upon these foundational principles.

Chapter 3

Methodology

3.1 Overall Research Approach

This thesis adopts a Design Science Research (DSR) methodology to address the identified challenges of media polarization for individual news consumers. DSR is an appropriate approach as the primary focus of this research is the creation and evaluation of a novel IT artifact – the "Polarization Detector" mobile application – designed to solve a practical problem – people struggling to maintain clarity of thought, consuming articles with polarized topics – by extending existing technological capabilities. The core of this DSR process involves the design, development, and demonstration of this application to provide users with accessible, real-time insights into the polarization present in the news content they consume.

The primary research goal, therefore, is to design, develop, and demonstrate the utility of a mobile application that effectively extends the POLAR framework. This extension aims to enable real-time polarization detection for end-users by implementing an efficient backend architecture that leverages pre-computed knowledge modelled by POLAR. This approach seeks to empower users with greater awareness and critical understanding of the media landscape, thereby contributing a practical solution to a pressing societal issue.

3.2 Leveraging the POLAR Framework: Foundation and Initial Setup

3.2.1 POLAR Framework Foundation and Case Study Definition

The POLAR framework, developed by Paschalides et al. [1], is fundamental to this project. The initial execution of POLAR to generate its comprehensive polarization model represents the first crucial step in the knowledge gathering and construction process for this thesis. As detailed in Chapter 2, this model encompasses key elements such as entities (e.g. people, organisations, places), the Sentiment Attitude Graph (SAG) representing their relationships, fellowships (clusters of entities with predominantly positive internal attitudes), and dipoles (two opposing fellowships whose entities exhibit predominantly negative attitudes towards each other, often concerning specific topics). Topics, particularly "polarizing topics", are central to the formation of these fellowships and dipoles within POLAR's graph-based model, where connections are weighted by expressed attitudes. The execution of the POLAR pipeline yields structured data files containing this critical information – including entities, the SAG, fellowships, dipoles, and polarizing topics with their associated indices – which serves as the foundational knowledge base for the system developed in this research.

The research for this thesis commenced in the summer of 2024. The period encompassing the U.S. presidential elections of November 5th, 2024, was selected as a highly relevant and prominent case study for generating the foundational polarization model with POLAR. This electoral context was chosen due to its inherent propensity for strong polarization, typically characterized by contentious debate topics (e.g. immigration and border control, abortion rights, war conflicts) that receive significant media coverage and often provoke strong public disagreement. Such periods are generally marked by heightened public sentiment and political alignment, alongside a high volume of news

article publication, with many outlets presenting distinct partisan perspectives and extensive coverage of major electoral events.

3.2.2 Data Acquisition and Initial Corpus Generation

Following the selection of this case study context, the initial data acquisition phase of the POLAR framework was undertaken. POLAR's data acquisition process begins by fetching articles from the GDELT Project[1], a large, open database of global news. The selection of articles to be fetched is guided by user-defined keywords and a specified date range, which are provided as initial parameters to POLAR's NewsCorpusCollector module. This module then retrieves article archives from GDELT that fall within the given date range and contain any of the specified keywords in their URLs.

The selection of keywords and the date range required careful consideration to maximize the retrieval of relevant articles, thereby ensuring a rich dataset for constructing a comprehensive polarization model for the chosen context. The chosen keywords included prominent political figures central to the election (e.g. Donald Trump, Kamala Harris, Joe Biden), the primary political parties (Democratic and Republican), and terms specific to the election period (e.g. "running mate", "super Tuesday"). The keyword selection also considered typical URL formatting conventions, leading to the use of lowercase terms with phrases separated by dashes (e.g. "running-mate"). Furthermore, variations in how terms might appear in URLs were accounted for (e.g. "JD Vance" could be represented as both "j-d-vance" and "jd-vance"). A full list of the keywords used is provided in "Appendix A" under the keyword_list parameter.

The selected date range needed to be sufficiently broad to capture adequate contextual information and generate substantial knowledge yet narrow enough to remain focused on the U.S. elections 2024 case study. Therefore, the start date was set to June 1st, 2024, accounting for events such as the first presidential debate between Joe Biden and Donald Trump on June 27th. The end date was established as January 31st, 2025, allowing for the inclusion of post-election analyses and coverage of the presidential inauguration on January 20th, 2025.

[1] https://www.gdeltproject.org/

With these parameters defined (including an output directory named outputJuneToJanuary), the NewsCorpusCollector module of polarlib was executed. This module performed a sequence of operations including downloading relevant daily GDELT archives, filtering articles based on keywords and other criteria (such as actor country codes, with "USA" kept as default, and processing up to 128 articles per day), fetching the HTML content of selected articles, parsing these articles to extract text and metadata into structured JSON files, and applying an initial text-cleaning pipeline. This process yielded a total of 7,696 unique news articles, averaging approximately 31.5 articles per day over the defined collection period.

3.2.3 Addressing Entity Extraction Challenges and Data Preprocessing

Following the initial data collection and preprocessing, the subsequent crucial step was entity extraction using polarlib. However, initial attempts at this stage revealed significant noise and misclassification of entities within the U.S. election context. For instance, "Joe DiMaggio" (the baseball player) was frequently extracted from mentions of "Joe Biden" due to the shared first name. Similarly, mentions of "J.D. Vance" were often misclassified as "Juris Doctor" or linked to "Zebulon Baird Vance". A particularly notable issue was the underrepresentation of Kamala Harris compared to other key figures like Donald Trump and Joe Biden; this was often because references to "Harris" alone were ambiguously linked to other entities (e.g. "Harris County") instead of the Presidential candidate. Other observed inaccuracies included "Democratic Party" being mapped to the "Democratic Party of Korea" and mentions of "swing state" being misinterpreted as "Swing Music".

To address these challenges and enhance the clarity and accuracy of the knowledge base, a degree of supervised data cleaning was deemed necessary, despite POLAR's generally unsupervised design. Three custom Python scripts were developed specifically for this U.S. elections 2024 case study. Two of these scripts were designed to run before polarlib's EntityExtractor module to preemptively correct or guide the extraction process, while the third script was applied after entity extraction for post-correction tasks.

The first pre-extraction script performed targeted string manipulations on the raw article texts. Its primary purpose was to remove common sources of noise that could lead to incorrect entity extractions, such as ambiguous standalone first names (e.g. deleting "Joe" when immediately preceding "Biden"), and to eliminate irrelevant boilerplate content like

advertising phrases (e.g. sentences starting with “Click here...”). The second pre-extraction script, the kamalaHarrisCorrector, specifically aimed to improve the recognition of Kamala Harris by disambiguating standalone mentions of “Harris”. This script analysed the current and up to nine preceding sentences, using predefined lists of contextual keywords (both positive indicators for Kamala Harris and negative indicators for other entities named Harris) and checking for titles, to determine if a “Harris” mention likely referred to Kamala Harris and, if so, replaced it with her full name, making sure the entity extractor would correctly identify it.

3.2.4 Full POLAR Pipeline Execution for Knowledge Generation

After these pre-extraction cleaning steps, the orchestrated polarlib pipeline, proceeded with the main analysis:

1. *Entity Extraction:* Performed using polarlib’s EntityExtractor on the refined article texts.

2. *Post-Extraction Entity Correction:* Following the initial entity extraction by polarlib, the third custom script, entitiesCorrector, was executed. This script refined the generated entity list by applying a set of custom rules defined in an external JSON mapping file (entity_mappings_corrections.json). For each extracted entity, it checked against these rules – which specified the original entity URI, its textual mention, and optional sentence-level conditions – to decide whether to retain the entity, replace it with a corrected canonical form, or delete it if deemed irrelevant or invalid, performing this correction process in two passes over the data (first pass could create conditions for corrections on the second pass).

3. *Noun Phrase Extraction:* Executed using NounPhraseExtractor to identify key noun phrases from the cleaned and entity-corrected texts.

4. *Topic Identification:* Utilised TopicIdentifier to encode the extracted noun phrases and then cluster semantically similar phrases into topics (using a clustering threshold of 0.8).

5. *Sentiment Attitude Calculation:* The SyntacticalSentimentAttitudePipeline was employed (configured with a Spacy en_core_web_sm model and the MPQA subjectivity lexicon[2]) to determine attitudes between entities and topics.

6. *Sentiment Attitude Graph (SAG) Construction:* The SAGGenerator constructed the SAG. This involved calculating attitude buckets and converting attitude signs based on specific bins (Negative: [-1.00, -0.02], Neutral: [-0.02, 0.10], Positive: [0.10, 1.00]) and a minimum frequency threshold determined empirically from the 99.2nd percentile of attitude pair counts for this dataset. This resulted in a SAG with 138 nodes and 256 edges.

7. *Fellowship and Dipole Extraction:* FellowshipExtractor identified fellowships with parameters set to n_iter = 25, resolution = 0.7, and merge_iter = 20, values found suitable for this dataset after experimentation. Subsequently, DipoleGenerator generated the dipoles representing conflicts between these fellowships.

8. *Topic Polarization Calculation:* Finally, the TopicAttitudeCalculator was used to load sentiment attitudes, identify topics relevant to the dipoles, and calculate the polarization index for these topics, thus concluding the generation of the foundational polarization model.

3.2.5 Technical Environment and Dependencies

This initial execution of the full POLAR pipeline was conducted using the polarlib library[3] within an isolated conda environment managed via Anaconda[4]. The process was run on a 2024 MacBook Pro with M4 Pro chip, 24GB Unified RAM, operating with Python 3.12.9. The standard dependencies, as listed in the polarlib requirements.txt file, were primarily specified for a Windows OS and an NVIDIA CUDA-enabled environment. Consequently, several modifications were necessary to ensure compatibility and functionality on macOS:

---

[2] https://github.com/beefoo/text-analysis/blob/master/lexicons_external/subjectivity_clues_hltemnlp05/subjclueslen1-HLTEMNLP05.tff [Accessed September 2024]. Needs to be downloaded, persistently stored and correctly reference its path.

[3] https://github.com/dpasch01/polarlib [Accessed September 2025]

[4] https://www.anaconda.com/docs/tools/working-with-conda/environments

(i) All NVIDIA-specific dependencies for CUDA processing were removed, as macOS utilises its Metal Performance Shaders (MPS) for GPU acceleration via PyTorch.

(ii) The pickle5 dependency was removed from the requirements, and any import pickle5 statements within the polarlib codebase were replaced with import pickle. This change was made because pickle5's functionalities are integrated into the standard pickle library in Python 3.8 and later versions, including the Python 3.12.9 environment used.

Furthermore, a necessary code modification was made within a polarlib component responsible for topic identification: the device (dev) specification inside the encode_noun_phrases function was changed from "cuda" to "mps" to enable GPU-accelerated processing on the MacBook Pro. These modifications enabled the successful execution of the POLAR pipeline within this macOS environment.

It is important to note that the entity extraction component depends on DBpedia Spotlight[5] for named entity linking [28]. This was run as a Docker container using the dbpedia/dbpedia-spotlight image (specifically for English, en), exposing port 2222, as per the polarlib documentation. This container needs to be active during the entity extraction phase. Additional note is that for calculations related to structural polarization (e.g. frustration index), it requires the Gurobi Optimizer[6]. This was installed, and a free academic license was obtained and activated according to Gurobi's guidelines.

These environmental configurations and modifications enabled the successful execution of the complete polarlib pipeline, yielding the structured data files that form the basis for the knowledge graph.

3.3 System Architecture

This system is designed with a client-server architecture, comprising a mobile application (frontend) and a sophisticated backend responsible for processing and analysis. The backend integrates several key services, including adjusted polarlib functionalities, a graph database for knowledge storage, and a Large Language Model (LLM) for summarising results. Figure 1 provides a high-level illustration of these components and

[5] https://www.dbpedia-spotlight.org/

[6] https://www.gurobi.com/downloads/gurobi-software/ [Accessed May 2025]

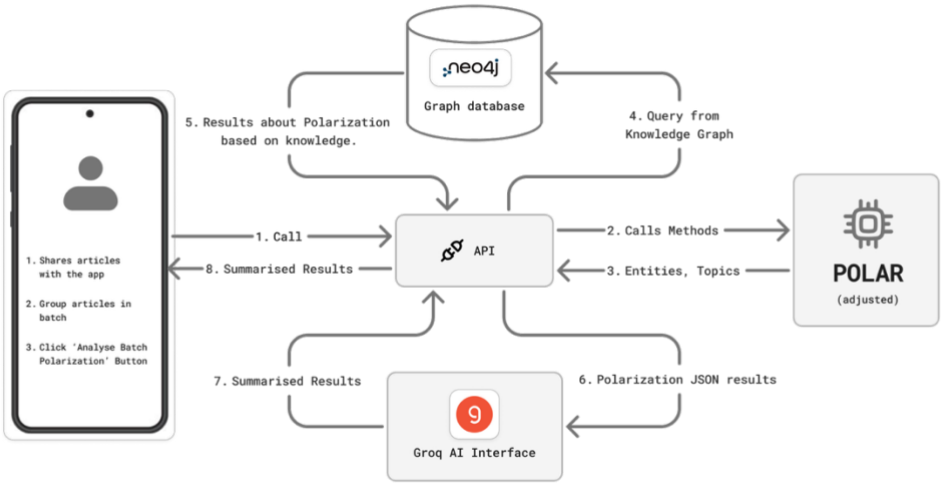

Figure 1 – Illustration of primary data flow during polarization analysis of a batch, from mobile app, using backend API to orchistrate the flow to POLAR, then to Neo4j graph database, then LLM summarisation, and finally the return of results to user

their interactions. The architecture is modular, with key backend components containerized using Docker, to separate concerns and facilitate efficient real-time polarization analysis.

3.3.1 Frontend: "Polarization Detector" Mobile Application

This Android mobile application serves as the primary user interface for the **"**Polarization Detector**"** system. Its main objective is to empower users by allowing them to easily submit articles for polarization analysis and to receive clear, understandable results. The application facilitates this through several key features:

Users can share article URLs directly from their mobile browser or other compatible applications, as the **"**Polarization Detector**"** is registered as a system share target. Once articles are shared with the app, users can **organis**e them into custom groups **(**batches**).** This feature provides flexibility, enabling users to group content by source, topic, date, or any other criteria they find relevant to their analysis needs.

To enhance user control and experience, the application also incorporates features for managing submitted content. This includes the ability to delete individual articles or entire batches, with options to either "unbatch" articles or permanently remove them.

Users can also remove specific articles from an existing batch. For efficient operation and offline accessibility of user-created batches, the application caches data locally using a Room database (an abstraction layer over SQLite).

For any created batch, the user can initiate a comprehensive polarization analysis with a simple action. This triggers a request to the backend system, which then performs the necessary processing to determine the polarization characteristics of the articles within that batch. Upon completion, the backend returns the analysis results, which the application then displays clearly to the user. The User Interface (UI) **adopts** a formal theme**,** designed with simplicity and ease of use as primary goals, to ensure an intuitive experience for a general audience.

3.3.2 Backend System

The backend system orchestrates all data processing and analysis tasks, minimizing the computational load on the user's device. It is comprised of several key services that work together:

1. *API Gateway:* A Flask-based API serves as the central communication hub, managing requests from the mobile application and coordinating interactions between the various backend services. It exposes specific endpoints for tasks such as initial article processing and batched polarization analysis.
2. *Adjusted POLAR Processing Module:* This module contains adapted functionalities from the polarlib library, specifically optimized for processing individual articles or small batches as they are received from the user. Key adjustments include fetching article content from user-provided URLs (rather than GDELT) and modifying polarlib functions to operate efficiently on these smaller inputs for tasks like entity and noun phrases extraction. This module, along with the API, runs within a Docker container, configured with the necessary dependencies (the requirements.txt file of polarlib, modified to include Flask, Groq and Neo4j client).

This *polar_api* docker container, utilises a file system, with similar structure as the one imagined by polarlib. It mounts a volume to an API_OUTPUT directory, where subdirectories and files generated during any processing in the backend are stored, to achieve persistence in generated data.

3. *Knowledge Base (Neo4j Graph Database):* The system utilises a Neo4j graph database (Community Edition), deployed as a separate Docker container, to store the extensive polarization knowledge generated from the initial, large-scale POLAR run (detailed in Chapter 3.2).

This database is deployed as a separate Docker container. Its role is to persist the pre-computed SAG, fellowships, dipoles, and polarizing topics for efficient querying during real-time analysis.

4. *LLM Summarisation Service (Groq AI Interface):* To provide users with a concise and understandable summary of the polarization analysis, the backend integrates with a Large Language Model (LLM) via the Groq AI API. This service takes the structured polarization data retrieved from the Neo4j database and, using a carefully crafted prompt, instructs the LLM to generate a textual summary of the findings, including key polarizing topics and involved entities/groups.

3.3.3 Primary Data Flow

Figure 1 illustrates the primary data flow when a user initiates a polarization analysis for a batch of articles:

1. The Mobile Application sends the selected batch of article, with identifier the name of the batch, to the Backend API.
2. The backend's Adjusted POLAR processing module performs lightweight entity and noun phrases extraction[7] on the articles in the user's batch. To optimize response times, some initial processing (individual article preprocessing), occurs when articles are first shared with the app, leaving just aggregations and filtering of entities and noun phrases at the time of polarization analysis. Filtering of noun phrases includes keeping only the ones that are also mentioned in a curated list of 51 manually selected meaningful topics (details in subsection 3.4.6). The intermediate and final outputs of these executions are stored permanently to the referencing batch directories.
3. These extracted entities and filtered noun phrases are then used to query the Neo4j Graph Database to retrieve the relevant pre-computed polarization information, which are the dipoles and polarization topics that appear in this batch.

[7] The reason why it stops there and not further (i.e. topic extraction) is explained in Chapter 3.5.2

4. The retrieved structured data is then passed to the LLM Summarisation Service (Groq AI Interface), which generates a textual summary.
5. This summary is returned via the API to the Mobile Application for display to the user, along with a list of key entities and topics identified in their batch.

3.4 Knowledge Base Construction and Management

Following the initial execution of the polarlib pipeline (detailed in Section 3.2), which generated a comprehensive model of the polarization landscape from the U.S. elections case study, the next critical step was to store this information in a persistent and query-efficient manner. This subchapter describes the construction and management of this knowledge base, which serves as the foundation for the real-time analysis performed.

3.4.1 Purpose and Technology Choice

To enable fast and complex queries for real-time polarization analysis – without re-running the computationally intensive polarlib pipeline for each user request – a dedicated, persistent knowledge base was essential. Various database models were considered. Traditional SQL databases, while structured, are not optimized for traversing the complex, interconnected relationships inherent in polarization data. Document-oriented databases were also deemed unsuitable for effectively representing the network structure of this information. While Vector Databases offered a strong option for semantic retrieval [29], this project required a more explicit representation of entities and their relationships rather than purely similarity-based lookups.

Ultimately, a graph database model was selected due to its natural ability to represent and efficiently query interconnected data, such as the entities, relationships, and group dynamics central to polarization models. Neo4j (Community Edition) was chosen as the specific graph database technology, primarily for its robust support for graph traversals, its expressive Cypher query language, and its ease of deployment [30]. As detailed in

Section 3.3.2, the Neo4j database was deployed as a Docker container for consistent setup and operational management.

3.4.2 Exporting Knowledge to CSV

The output from the polarlib pipeline (described in Chapter 3.2) consists of various Python pickle (.pckl) and JSON files that store the Sentiment Attitude Graph (SAG), entity mappings, fellowship lists, topic dictionaries, dipole lists, and attitude information. To prepare this data for ingestion into Neo4j, an augmented version of polarlib's PolarizationKnowledgeGraph class was utilised.

This class was first used to load and consolidate all the relevant data from the output files. Subsequently, custom export functions (e.g. export_entity_member_of_fellowship, export_dipole_polarization_to_topic, etc.) were implemented or leveraged.

These functions transformed the internal graph representation and associated data within the PolarizationKnowledgeGraph object into a set of structured CSV files. Key CSV files generated through this process included:

i. entities.csv (listing unique entities – DBpedia URL as Id – and their names)
ii. topics.csv (listing unique topics and their labels)
iii. dipoles.csv (listing unique dipole identifiers)
iv. entity_fellowship.csv (mapping entities to their respective fellowships)
v. fellowship_dipole.csv (linking fellowships to the dipoles they are part of)
vi. entity_entity_attitudes.csv (detailing direct sentiment relationships between entities)
vii. entity_topic_attitudes.csv (detailing entity attitudes towards specific topics)
viii. fellowship_topic_attitudes.csv (detailing collective fellowship attitudes towards topics)
ix. dipole_topic_polarization.csv (detailing the polarization of topics within specific dipoles, including the polarization index)

This intermediate CSV format provided a clean structure ready for batch ingestion into Neo4j. Figure 2 illustrates this process from data extraction of the initial execution of POLAR commands pipeline, to the organisation into the CSV files ready for ingestion to the Neo4j graph database.

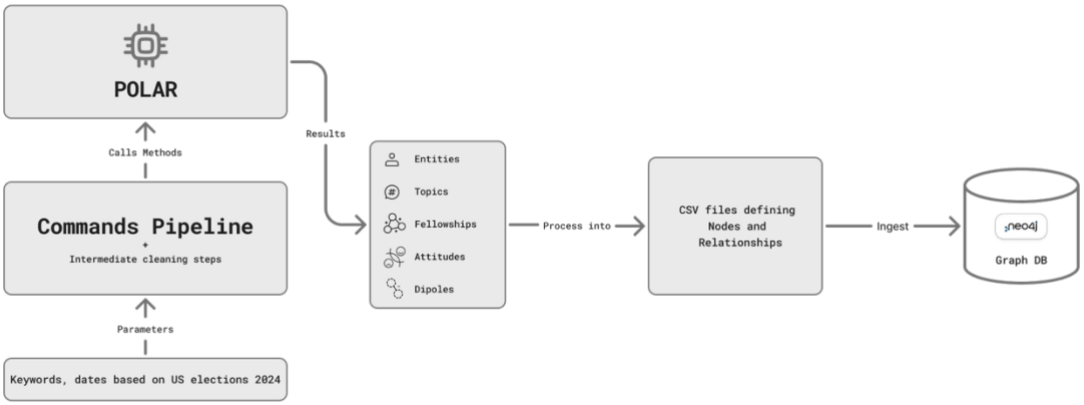

Figure 2 - Flow from initial POLAR exesution, with its output organised into CSV files for ingestion to Neo4j graph database

3.4.3 Neo4j Graph Schema Design

The Neo4j knowledge base was designed with a schema that mirrors the core components of the POLAR framework. The primary node labels defined are:

a) Entity: Represents individuals, organisations, locations, etc. Key properties include id (DBpedia URL) and name.
b) Topic: Represents discussion topics extracted from the news corpus. Key properties include id and name (topic label).
c) Fellowship: Represents clusters of like-minded entities. Key properties include id (e.g. F0, F1).
d) Dipole: Represents a conflict between two fellowships. Key properties include id (e.g. D0_1).

These nodes are interconnected through the following relationship types, which capture the structure and dynamics of the polarization model:

i. MEMBER_OF (Entity → Fellowship)
ii. PART_OF (Fellowship → Dipole)
iii. HAS_ATTITUDE_TO_ENTITY (Entity → Entity), including weight property with domain of either -1.0 for hostility or +1.0 for friendliness.
iv. HAS_ATTITUDE_TO_TOPIC (Entity → Topic), with properties of weight – domain of [-1.0, 1.0] – and observations.

v. HAS_COLLECTIVE_ATTITUDE_TO_TOPIC (Fellowship → Topic), with weight property on domain [-1.0, 1.0].

vi. HAS_POLARIZATION_TOWARDS (Dipole → Topic), with properties of weight – domain of [0.0, 1.0] from none to high polarization index – and observations.

Figure 3 illustrates this graph schema.

3.4.4 Data Ingestion into Neo4j

The process of populating the Neo4j database with data from the generated CSV files was managed by a custom Python class, Neo4jIngestor. This process involved:

i. Node Creation: For each type of primary data element (Entities, Topics, and Dipoles), – the fellowships were created at the same time as the relationship – the corresponding CSV file was processed. The injector efficiently created a distinct node in the graph for each unique item, assigning it the appropriate label and setting its properties, such as its unique identifier and name. Care was taken to avoid duplicating nodes if they already existed from previous processing steps.

ii. Relationship Establishment: Once the primary nodes were established in the database, the relationships that connecting them were created. For each type of connection, the relevant CSV data was read. The injector then identified the corresponding source and target nodes in the graph and established the defined relationship between them, assigning appropriately properties for that relationship (weight and/or observation).

3.4.5 Final Populated Knowledge Base

Upon completion of the ingestion process, the Neo4j instance contained a fully populated graph database. This knowledge base represents a structured and ready-to-be-queried

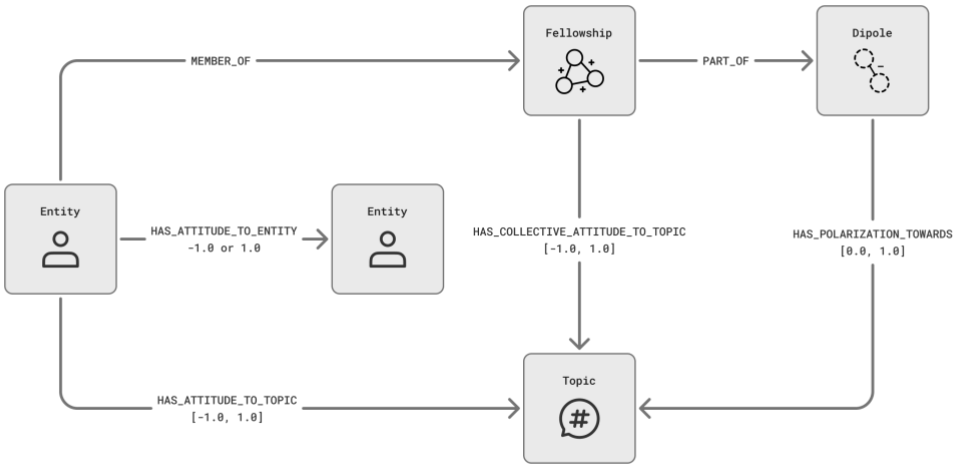

Figure 3 - Ilustration of the nodes and relationships of the data model of the graph database, including the relationship weights and value domains

snapshot of the polarization landscape of this case study. Figure 4 shows the size of the populated neo4j graph database.

Figure 5 provides a visual snapshot of a selected portion of this knowledge base, illustrating the connections of dipoles, their component fellowships, member entities, and associated polarizing topics.
This visualisation was generated using a custom script that first queries the Neo4j database to retrieve information about the top 5 most populous dipoles. The query, fetches

details for each selected dipole, including its member fellowships and their entities, its top 3 most polarizing topics (filtered from the curated list of meaningful topics), and examples of entity-to-entity, entity-to-topic, and fellowship-to-topic attitudes within the context of that dipole. For instance, this visualised example (Figure 5) highlights among others a notable dipole. This dipole features a fellowship centred around the “Kamala Harris” entity (associated with entities such as “Tim Walz” and the “California Democratic Party”), in opposition to a singleton fellowship representing the “Donald Trump 2024 campaign”. Significantly, among the polarizing topics identified for this specific dipole are “border” and “health”. This finding corresponds closely with the prominent real-world political discourse observed during the U.S. 2024 election period,

Nodes		
Topics	→	5,489
Entities	→	138
Fellowships	→	119
Dipoles	→	72
	Total nodes	5,818
Relationships		
MEMBER_OF	→	137
PART_OF	→	144
HAS_POLARIZATION_TOWARDS	→	10,634
HAS_ATTITUDE_TO_ENTITY	→	512
HAS_ATTITUDE_TO_TOPIC	→	5,297
HAS_COLLECTIVE_ATTITUDE_TO_TOPIC	→	8,552
	Total relationships	25,276

Figure 4 - Number of nodes and relationships of each type of the populated neo4j

where issues of border control and healthcare policy were indeed central and highly contested subjects of debate [31].

3.4.6 Curated List of Meaningful topics

To ensure the application provides focused and impactful insights from the many topics generated by the initial polarlib run (detailed in Chapter 3.2), a curated list of 51 meaningful and highly polarized topics was developed. This step was crucial for concentrating the analysis on distinct and relevant issues within the U.S. elections 2024 case study, effectively filtering out overly broad, granular, or less pertinent topics.

The curation process began with a quantitative assessment using polarlib's TopicLevelPolarizationAnalyser. This tool calculated a "global polarization score" for every topic, reflecting its overall divisiveness across the dataset. Topics that met a minimum score threshold of 0.5 were identified as "highly polarized". This filtered subset of 581 topics, was then manually and carefully reviewed. During this review, topics were selected for the final list based on their clarity, direct relevance to major election

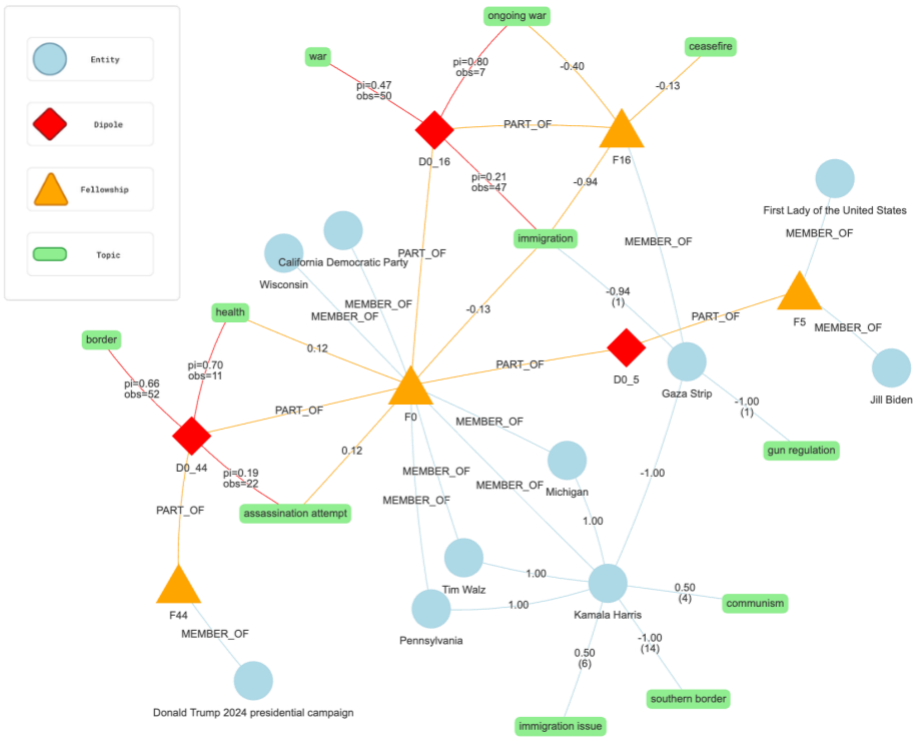

Figure 5 - Visually Display of a snapshot of the populated neo4j database showing the 3 dipoles, with the 3 most polarizing topics and the component entities and fellowships

narratives and key campaign events, and their potential interest to an end-user seeking to understand media polarization, resulting in the final list of 51 topics. This curated list is important to the system, as it is subsequently used by the backend system to refine the focus of real-time polarization analysis. The full list of these topics is displayed in "Appendix D".

3.5 Real-time Polarization Analysis for User Batches

This section details the step-by-step process undertaken by the backend system when a user first shares the articles and places them in a proper batch and then initiates a polarization analysis request for said batch. The pipeline is designed for efficiency, leveraging pre-computed knowledge and lightweight processing to deliver timely results to the mobile application.

3.5.1 Individual Article Pre-processing

A key strategy for ensuring system responsiveness is to perform significant groundwork when individual articles are first shared by the user, rather than deferring all computation until a full batch analysis is requested. This amortization of processing helps minimize the final delay perceived by the user, leading to a smoother experience. When a user shares an article with the application, the following backend operations are triggered:

i. *Article Reception and Content Fetching:* The mobile application sends a POST request to the backend API endpoint (/article/preProcess), containing as parameter the URL of the shared article. The backend then fetches the article's content from this URL using a standard HTTP library (python's "requests").

ii. *Content Parsing and Initial Preprocessing:* To handle individual articles effectively, a custom python class, ArticleCollector, was developed. This class is distinct from polarlib's NewsCorpusCollector (which is designed for large-scale GDELT processing). The ArticleCollector is initialized with three parameters, an output directory, a batch identifier – set to a temporary (tmp) directory – for the incoming article, and a URL. It then utilises adapted versions of parse_articles and pre_process_articles functions, driven from polarlib's logic, to parse the fetched HTML content and extract the raw text. This process is streamlined to operate on the single provided URL.

iii. *Custom Pre-Entity Extraction Cleaning:* The initially processed text of the article is then passed through the first two custom cleaning scripts – cleanFiles and kamalaHarrisCorrector – as mentioned in Chapter 3.2, with minor changes to clean only the file of the specified URL.

iv. *Entity Extraction:* EntityExtractor class of polarlib **was a**dapted to work on the single article**'**s content. A crucial configuration detail for this stage, given the containerized backend environment, involves setting the DBpedia Spotlight URL. When services run in separate Docker containers (as is the case for the backend API/polarlib processing and the DBpedia Spotlight service), they typically communicate using their defined service names on the Docker network. Therefore, the Spotlight URL parameter for the EntityExtractor was configured to point to the DBpedia Spotlight container**'**s service address (http://spotlight:80/rest/annotate**,** replacing localhost or 127.0.0.1).

v. *Post-Entity Extraction Cleaning:* The third custom script, entitiesCorrector (also described in Chapter 3.2), is then applied to the entities extracted from this single article**.**

vi. *Noun Phrase Extraction:* Finally, polarlib's NounPhraseExtractor is executed on the article's text to identify and extract relevant noun phrases, similarly adjusted for single URL processing. This initial pre-processing deliberately concludes at NP extraction due to efficiency considerations grounded in POLAR's methodology. According to Paschalides et al., NPs are the fundamental elements that **get** subsequently cluster**ed** to form formal topics – a process that is computationally intensive. By extracting only the NPs at this stage, which can be seen as the unclustered **"**raw**"** topics, the system efficiently gathers essential topical indicators for individual articles. The more demanding task of semantic clustering and full topic identification is thereby deferred until a user initiates a complete batch analysis, significantly reducing the processing time for each shared article. This optimization yields substantial performance benefits; for instance, an earlier implementation that included full topic identification and clustering required approximately **1-2** minutes for a small batch of 7 articles, on the contrary however**,** the current approach focused on NP extraction reduces this initial processing to a few seconds. Another major issue with that approach was that the clustering had to be executed on the whole batch of articles, on the polarization extraction initiation.

vii. *Output and Storage of Features**:*** During this article preprocessing, the extracted entities and noun phrases, and the rest intermediate generated files – are stored as part of the temporary (tmp) batch.

At the end of this backend processing, a success code (200) is sent to the frontend, indicating successful fetching (otherwise an error code with explanatory message). The mobile application then is free to store the new article persistently (using Room [32], layer over SQLite) to an "Article" table, with batch set to the temporary (further structure explanation in Section 3.6). This change updates the User Interface automatically.

This process described before, hides the heavy analysis processing by eliminating a big work part with every article shared. Consequently, when the user groups the articles to a batch, a simple – almost instant – moving the files to the new batch is enough.

3.5.2 Batch Analysis Initiation

This subsection details the sequence of events and backend processes that are triggered when a user explicitly requests a polarization analysis for a previously defined batch of

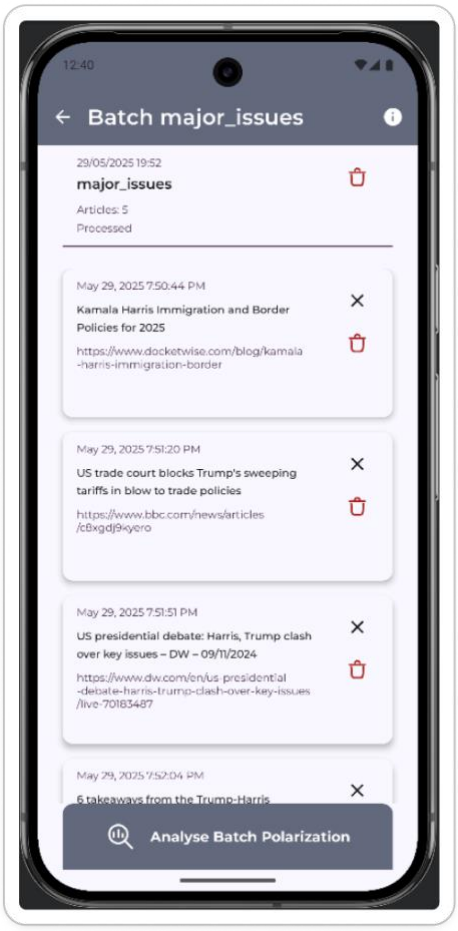

Figure 6 - BatchDetailsScreen for a batch named "major_issues", with a header, the list of articles placed in this batch, and at the bottom the "Analyse Batch Polarization" button

articles. As illustrated in Figure 6, the user, having shared articles and organised them into a batch, can initiate this analysis with a "Analyse Batch Polarization" button within the application's BatchDetailsScreen (Figure 20).

3.5.2.1 First Communication to Backend

The initiation for analysis, starts a sequence of distinct requests to the backend API. While these could be consolidated into a single request, they were separated during development for ease of testing and to allow for a more progressive display of results to the user (e.g. showing entities and topics before the final polarization summary). These requests include the following functionalities:

1. Batch Entity Aggregation (/batch/entities endpoint): Retrieves a consolidated list of all unique entities present across the articles in the specified batch, along with their frequency of mention.

2. Batch Topic Indicator Aggregation (/batch/topics endpoint): Retrieves a list of the meaningful topics (derived from noun phrases) of the batch, along with number of mentions.
3. Polarization Analysis and Summarisation (/batch/polarization endpoint): This is the main analysis call which, if necessary, can internally trigger the entity and topic aggregation steps. It then queries the knowledge base and returns the LLM-generated polarization summary and supporting data – the results from the query themselves, but removing abstract content (e.g. fellowship or dipole ids, etc).

For all three calls, the primary parameter is the unique batch name. The backend functions handling entity and topic aggregation are designed to first check for previously computed and cached results for that batch, and if found, these cached results are returned immediately to avoid redundant processing.

3.5.2.2 Aggregating Entities of Batch

As detailed in Chapter 3.5.1, entity extraction is performed for each individual article when it is first shared, resulting in JSON files stored in a directory structure like API_OUTPUT/entities/<batch_name>/<article_identifier>.json. When the /batch/entities API endpoint is called (or when the main /batch/polarization endpoint triggers this step), the backend aggregates these individual entity lists for the specified batch.

This is achieved by the entities_mapping() method within the adapted EntityExtractor class. For each entities file for each article, it parses the JSON, extracts all identified entities – using their DBpedia URI as the unique identifier – and accumulates a total count for each unique entity across all articles in the batch. The aggregated results sorted by entity frequency are then saved as an entities_mapping.json file within the batch's entity directory for caching and is also returned to the mobile application.

3.5.2.3 Aggregating and Filtering Noun Phrases as Topic Indicators of Batch

Similarly, noun phrases are extracted for each individual article during the initial pre-processing (Section 3.5.1), with results stored in a directory like API_OUTPUT/noun_phrases/<batch_name>/<article_identifier>.json. When the /batch/topics API endpoint is called (or this step is triggered by the main polarization

analysis function), these individually extracted noun phrases are aggregated and refined for the entire batch.

This process is handled by the calculate_batch_meaningful_topics_from_noun_phrases() function, which internally uses a noun_phrases_mapping() method (placed inside the NounPhraseExtractor class).

This mapping method first iterates through all individual article noun phrase JSON files for the given batch. It extracts each noun phrase and accumulates a total count for every unique noun phrase across all articles in the batch. This raw mapping of all noun phrases to their frequencies is saved to a file (e.g. topics_name_mapping.json) for caching.

The system filters the noun phrases by loading the curated list of 51 meaningful topic names (as described in Section 3.4.6). Only the noun phrases from the batch that exactly match an entry in this curated list are retained.

The final refined list is saved to a separate file – topics_name_mapping_cleaned.json – and returned to the mobile application.

The output of these aggregation and refinement stages is a list of unique entities present in the batch (with their frequencies) and a refined list of key topics in the batch (with their frequencies). These aggregated features are now ready to be used for querying the Neo4j knowledge base, as detailed in the following section.

3.5.3 Querying the Neo4j Knowledge Base

Once the aggregated entities and meaningful topics for the batch are prepared, the backend system queries the Neo4j graph database. This step aims to retrieve relevant, pre-computed polarization information associated with the batch's content. The primary function responsible for this is fetch_batch_polarization_data_strict, which takes the list of entity IDs (Dbpedia URLs) and topic names from the batch as input, along with a minimum polarization threshold (min_pol = 0.2).

The querying process executed by this function involves several key stages:

1. *Identifying Relevant Dipole – Topic Polarizations:* A Cypher query is first executed to find Dipole – Topic pairs where:

a. At least one Entity from the user's batch is a member of a Fellowship involved in the Dipole.
b. The Topic name is present in the batch's topic names.
c. The HAS_POLARIZATION_TOWARDS relationship between the Dipole and Topic has a weight (polarization index) greater than the specified min_pol threshold.

2. *Strict Filtering:* A dipole – topic polarization instance is only retained if both fellowships of the dipole have at least one entity member that is also present in the user's current batch of articles. This ensures that the reported polarization is directly relevant and represented by entities from the user's specific input. This can, of course, be adjusted according to the needs and personal definition of polarization for a batch. In this case, we could define a batch being polarized on a topic *t*, iff:
   i. *t* exists in batch
   ii. There is a dipole *d polarized towards t* with two fellowships *f1* and *f2 part of d*.
   ii. There is at least one entity *e1 member of f1 part of d* and at least one entity *e2 member of f2 part of d, f1 ≠ f2*, with both *e1* and *e2* existing in the batch

The final output of this querying stage is a structured JSON object. This object primarily contains a list under dipole_topic_analysis, where each entry represents a highly relevant dipole – topic polarization instance. An example structure of the resulting JSON is displayed in Figure 7.

3.5.4 LLM-based Summarisation

While the Neo4j queries retrieve structured data identifying polarized topics and involved groups, this raw output can be complex for direct user interpretation. To provide a more accessible, narrative understanding, the system utilises a Large Language Model (LLM) to translate these findings into a concise summary. The rapid advancements in LLMs [33], particularly their ability to understand context and generate coherent text (generative AI), make them well-suited for transforming structured polarization data into an easily digestible paragraph.

Figure 7 - Example Json result from Neo4j query regarding polariation analysis. Shows a dipolewith polarization index 0.72 on topic "immigration" and the two fellowships considting the dipole with their entities and collective attitude to the topic

3.5.4.1 Decision on Interface and LLM

For the decision involved to decision on the LLM technology to use, I needed an LLM with high accuracy for cost ratio. The Groq AI interface emerged as a strong candidate, offering rapid inference speeds. Meta's "llama-3.3-70b-versatile" model (as available via Groq) was selected as a very good option, providing a powerful balance of contextual understanding and generation capabilities [34]. As of the time of development, Groq allows for a generous number of requests and tokens within its free tier[8] (e.g. 30 requests per minute, 1,000 requests per day, 12,000 tokens per minute, and 100,000 tokens per day), which is suitable for the application's expected load. As the task for the LLM would

[8] https://console.groq.com/docs/rate-limits

be a delicate summarisation of complex relational data, an accurate and capable LLM like a 70B parameter model was deemed appropriate. Of course, different interfaces or LLMs could potentially provide similar results, and this choice reflects the tools and performance characteristics prioritised for this project.

3.5.4.2 Preprocessing results for Prompt Generation

The structured JSON data retrieved from Neo4j, while rich in detail, is not in an optimal format for direct ingestion by an LLM if the goal is a high-quality narrative summary. Therefore, a significant preprocessing step is undertaken by the prepare_polarization_prompt method to transform this data into a more descriptive and organised textual format, which then forms the core of the prompt.

Algorithmically, this preprocessing involves the following key operations:

1. *Iterate and Aggregate by Topic:* The function iterates through each entry in the dipole_topic_analysis list from the Neo4j results.
2. *Track Maximum Polarization:* For each unique topic_name, it initialises or updates a record. It keeps track of the max_pi (maximum polarization index observed for that topic across different dipoles) and the set of source_dipoles contributing to its polarization.
3. *Consolidate Fellowships by Stance:* Within each topic, it organises information about the involved fellowships based on its collective stance (positive, negative, or neutral) towards that topic.
    a. *Unique Entity Group Identification:* To avoid redundantly listing the same set of entities if they appear in multiple technically distinct fellowship objects, an entity_key is created for each group of entities. This key is a tuple of sorted entity names found within a fellowship in the current batch.
    b. *Stance Categorisation:* The stance is determined by the attitude_score of the fellowship towards the topic, using predefined thresholds (strongly negative [-1.0, -0.6], negative (-0.6, -0.2), neutral [-0.2, 0.2], positive (0.2, 0.6], strongly positive (0.6 – 1.0)).
    c. *Conflict Resolution for Duplicate Groups:* If the same unique group of entities (identified by entity_key) appears multiple times under the same topic and stance category (e.g. if different dipoles identify the same group with slightly different

attitude scores), the system retains the instance where the group exhibits the highest absolute attitude.

4. *Sort Topics:* After processing all entries, the aggregated topics are sorted in descending based on their polarization index.
5. Final Formatting: The resulting data is then systematically formatted into a series of textual lines that will constitute the main informational part of the prompt fed to the LLM. Helper functions are used to convert polarization index and attitude scores into qualitative descriptions (e.g. for polarization index: (high: (0.65, 1.0], moderate: (0.35, 0.65], low [0.0, 0.35]) for better readability within the prompt itself.

This careful preprocessing ensures that the LLM receives a clear, organised, and non-redundant representation of the polarization data, ready for a summarisation task.

3.5.4.3 Prompt Generation:

The construction of the prompt itself is a critical aspect of obtaining the desired summary from the LLM. The prepare_polarization_prompt function, after preprocessing the data as described before, assembles the final prompt using several prompt engineering techniques [35]:

1. Role Assignment (Persona): The prompt begins by assigning a role to the LLM: "You are an expert in analysing and summarising political polarization from structured data." This helps set the context and primes the LLM to use appropriate style and knowledge.
2. Clear Task Definition: The task is explicitly stated: "Your task is to generate a concise, human-readable summary of polarization from the data below."
3. Contextual Definitions: To ensure the LLM correctly interprets the provided data, the key terms of "Polarization index (pi)" and stance (the attitude) are given a definition.
4. Structured Output Guidance (Few-shot learning) [36]: The prompt provides explicit instructions on how the summary should be structured:
    a. Start with an overall assessment.
    b. Identify the most polarized topics.
    c. For each key topic, state its name, polarization level, and describe the groups holding positive and negative stances, including their representative entities and interpreted stance. An example output format for this description is provided (few-shot)

5. Stylistic and Content Constraints: The prompt guides the LLM on the desired style and content:
    a. “Use clear, narrative language. Synthesize the information rather than just listing data points.”
    b. “Keep the summary concise yet informative. Do not include any assumptions, only knowledge provided here.”
6. Data Presentation Format: The pre-processed polarization data is clearly sectioned under a heading like “Aggregated Polarization Data by Topic (Groups per Stance):”. Each topic’s information is then laid out with its overall polarization and lists of groups under positive and negative stances, including their entities and interpreted attitudes (e.g. Topic: “Topic Name”, Overall Topic Polarization: High (Max PI: 0.85)).
7. Handling Empty or Insufficient Data: The prompt generation logic includes a provision for cases where no significant polarization data is found after aggregation, ensuring a graceful message can be formed.
8. Input Token Limit Awareness: The final prompt string is truncated to 5000 characters if it exceeds this limit, preventing errors from too long inputs to the LLM API.

This prompt is also saved by the backend (e.g. prompt_<timestamp>.json) for logging and debugging purposes. See the full prompt instruction in “**Appendix E**”

3.5.4.4 LLM Configurations

The interaction with the Groq AI API is configured using specific parameters to control the LLM’s generation process, as implemented in the get_polarization_analysis_summary method. The chosen model is *llama-3.3-70b-versatile*, and the API call uses a messages-based structure, including a system message that provides a high-level instruction or persona, reinforcing the main prompt (“You are a concise expert in summarising political polarization based on knowledge provided.”) and a user message containing the detailed prompt generated.

A max_tokens parameter is set to 500 to limit the maximum length of the generated summary, ensuring conciseness and managing token consumption. The temperature of the LLM is set to 0.4 (which is considered low) for a more deterministic, focused, and

less random output [37], which is generally preferred for summarisation tasks that need to stay close to the source data, balancing factual grounding with readability.

3.5.4.5 Output Handling

Once the LLM generates the summary, the backend performs a few final steps of whitespace stripping, storing the prompt and summary persistently as files for logging purposes in the API_OUTPUT/polarization_results/<batch_name> as prompt_<timestamp>.json and <timestamp>.json accordingly.

3.5.5 Response to Mobile Application

The results of the polarization analysis process that are sent to the mobile application are:
(i) success code (200), otherwise error code + message
(ii) summary generated from LLM
(iii) A simplified version of the structured polarization data queried from the Neo4j – removed the IDs of dipoles, fellowships and entities. This way allows the application (and any frontend) to display, along with the summary, the raw data in any other way.
(iv) Remember, from the batch/entities and batch/topics calls, a list of entities with their frequencies and a list of filtered topics with their frequencies are returned to the app before everything else. If these exclusive calls would be removed, the lists could be returned from the main analysis function as well.
The benefit of returning all these different data to the frontend is that it can display whatever and however it deems appropriate, with easily manageable refinements.

3.6 Mobile Application Development

The “Polarization Detector” mobile application, developed as a proof-of-concept, serves as the primary interface for end-user interaction with the system. Key objectives guiding its development included ensuring internal scalability, manageability, efficiency, and data persistence. Concurrently, the user interface (UI) and user experience (UX) were designed to adhere to a simple, clean, and minimalistic pattern, emphasising formal presentation, ease of use, and clarity. The core functionalities addressed by the application are: (i) seamless sharing of articles by the user, (ii) organisation of shared articles into

user-defined batches, (iii) straightforward review of articles within each batch, and (iv) clear and appropriate presentation of the polarization analysis results.

This section details the mobile application's development, focusing on the technologies employed, mechanisms for persistent and efficient local data storage, strategies for reliable background execution of long-running tasks, and the design and organisation of the application's screens. Emphasis is placed on the AnalysisResultsScreen, which is responsible for displaying the outcomes of the polarization analysis.

3.6.1 Environment and Technologies

The Android mobile application was natively developed within Android Studio. Kotlin was chosen as the primary programming language, for its modern features and conciseness. The user interface (UI) was constructed using Jetpack Compose, Google's contemporary declarative UI toolkit for Android[9], which facilitates the creation of dynamic and responsive user interfaces.
The application could have been built using any other environment and target device group (e.g. IOS, multiplatform), but Android development was chosen due to some prior experience and consequently helped with faster development.
Material Design 3 guidelines ensured that the application incorporates best practices in UI/UX design, promoting consistency and a high-quality user experience.

For local data persistence, Room Persistence Library was integrated. Room provides an abstraction layer over SQLite, simplifying database operations and offering compile-time verification of SQL queries [32]. Network communication with the backend API was managed using Retrofit, a type-safe HTTP client for Android and Java, in conjunction with OkHttp3 as the underlying HTTP engine for efficient request handling and customization [38]. For managing long-running background tasks, such as article preprocessing and the final polarization analysis requests, WorkManager was employed to ensure these operations are persistent and handled reliably, even if the application is closed or the device restarts [39].

[9] https://developer.android.com/compose

The application follows the Model-View-ViewModel (MVVM) architectural pattern. This architectural choice is recommended by Google for Android development as it promotes a separation of concerns, making the codebase more modular, testable, and maintainable. In MVVM, the ViewModel prepares and manages data for the UI, the View (Composable functions in Compose) observes the ViewModel for data changes and updates the UI, and the Model represents the data and business logic [40].

3.6.2 Enabling Article Sharing

Seamless article sharing is a pivotal feature enabling the "Polarization Detector" to function as an on-demand analysis tool. To facilitate direct sharing from a user's mobile browser or other news applications, the application was registered as a system share target [41].

This involved configuring an IntentFilter within the AndroidManifest.xml file for the Main Activity. This filter specifies that the Activity can handle ACTION_SEND intents with a data type of text/plain, which is standard for sharing URLs or plain text.

The application's main activity was then designed to handle *two distinct launch modes*, a normal launch – when user opens the app from their phone Home Screen – which navigates the user to the application's HomeScreen, and a share intent launch – when the app opens after a user selects from their browser share menu – which navigates the user to the UrlSharedScreen having as parameter the URL just shared.
This implementation ensures that users can easily direct articles to the **"**Polarization Detector**"** from various sources within the**ir** Android device.

3.6.3 Persistent Local Data Storage

To provide a robust user experience, maintain user-created data across application sessions, and enable offline access to article batches, the Room Persistence Library was implemented. Room acts as an Object Relational Mapper (ORM) over SQLite, simplifying database interactions and providing compile-time checks for SQL queries [32].

Figure 8 - Local Traditional SQL scema of Local Room Database of Android application showing four tables (Batch, Article, Entity, Topic) with their columns.

The local database schema includes the following primary tables (Figure 8):

(i) *Batch Table:* Stores information about user-created batches, as well as a flag and the batch polarization results. Specifically, a batch contains a mandatory unique name, a flag indicating whether the batch has been analysed, for better UI presentation, the polarization summary of the LLM after the batch analysation, and the raw JSON results (see Section 3.5.5, (iii) on the list) which is converted and stored as a list of DipoleTopics. A DipoleTopic represents an item in this list of the JSON, with the topic name, polarization index, the two fellowships with their entities and collective attitudes. The storage of a list of objects inside a single table was possible with the use of a TypeConverter that "JSONifies" the object on insertion to the table.

(ii) *Article Table:* Stores details about individual articles shared by the user. These include the URL, the article title (which is fetched by the mobile app for better user experience), and a batch id as a foreign key to the associated batch.

(iii) *Topic Table & Entity Table*: These tables are used to cache the resulting lists of articles and topics with their number of mentions, and they also have a reference to the batch.

A significant advantage of using Room with Jetpack Compose and modern Android architecture components (like Kotlin Flows) is that changes to the database can be observed directly. This allows the UI to update automatically when data is modified (e.g. an article is added to a batch, a batch is deleted), leading to a responsive and up-to-date user interface. Using this simple SQL schema, results into an effective local storage of the data benefiting the user experience and data persistence.

3.6.4 Remote Calls to Backend

The backend serves as the analytical core of the system, offloading computationally intensive processing from the mobile device. Consequently, robust and efficient communication between the mobile application and the backend is essential for effective data flow and reliable operation, particularly in handling potential network failures.

3.6.4.1 Establishing the Connections

All network requests to the backend are managed using Retrofit. An EndpointsService interface defines all the necessary API endpoints (e.g. /article/preProcess, /batch/entities, /batch/topics, /batch/polarization, /batch/create, etc.), corresponding to the backend's API structure. This interface uses annotations to describe HTTP operations, request parameters, and response types.

Kotlin data classes are utilised to model the structure of request bodies and JSON responses for each endpoint. Retrofit, configured with a Gson converter, automatically handles the serialization of outgoing data and deserialization of incoming responses.

The underlying OkHttp3 client, used by Retrofit, is configured with appropriate parameters, such as read/write timeouts. The base URL for API calls is set to the backend service address. When the backend is deployed in a Docker container on the same host machine as the Android Emulator, this URL is typically http://10.0.2.2:8080/, where 8080 is the port exposed by the polar_api Docker container.

A repository file abstracts each of the ports as functions that use the service, call the backend, and check for successful or error results and acting accordingly – either forwards the retrieved data or an error message.

3.6.4.2 Persistent Backend calls

Certain backend operations, notably article preprocessing and the multi-stage batch polarization analysis, can be relatively time-consuming and are susceptible to interruptions from network instability or application lifecycle changes. Executing such tasks directly within a ViewModel's lifecycle could lead to their premature termination if, for instance, the user navigates away from the app or network connectivity is lost.

To mitigate these issues and ensure the reliable execution of critical background tasks, WorkManager is used. WorkManager is an Android Jetpack library specifically designed

for deferrable, asynchronous tasks that require guaranteed completion [42]. In this application, WorkManager is utilised for the four most significant backend interactions: the initial article preprocessing call and the three distinct API calls involved in the full batch analysis process. This approach ensures that these operations persist and complete successfully, even in the face of application closure or transient failures.

3.6.5 Architecture and file organisation

The codebase is structured according to the MVVM (Model-View-ViewModel) pattern. This architecture was used to separate the presentation logic from the UI, making the app easier to maintain and extend. For each screen, there are four separate files orchestrated for its complete functionality.

i. Composable file: This file contains the Jetpack Compose functions responsible for rendering the user interface of a specific screen. It includes a main @Composable annotated function representing the screen itself, along with any helper composable functions used to build smaller UI elements within that screen. This layer observes data exposed by the ViewModel and reacts to changes by recomposing the UI.
ii. ViewModel class: Responsible for holding and managing the UI state for that screen, fetching data from the Model (e.g. repositories), processing user inputs, and exposing observable data (e.g. using Kotlin Flows) that the Composable UI can collect and act accordingly.
iii. State data class: Kotlin data class that defined to represent the complete UI state for a screen. This immutable state object holds all the information the UI needs to render itself at any given moment (e.g. lists of articles, loading indicators, error messages).
iv. Event sealed interface: Declares the different types of events or actions that can originate from the UI (e.g. a button click, a list item selection). The ViewModel then implements an “onEvent” method to handle these specific events, triggering appropriate business logic or state updates.

The above structure creates a separation of concern, following the MVVM architecture and best practises, for maintainable and scalable code.

3.6.6 User Interface – Mobile Application Screens

The “Polarization Detector” mobile application is designed as the primary, user-facing component of the system, responsible for enabling interaction and delivering polarization insights. A core principle in its design was the adoption of a simple, clear, and minimalistic UI. This approach aims to provide an intuitive user experience, ensuring that the application is easy to navigate and the presented information is easily understandable, aligning with the formal nature of the analysis provided. The next sections will detail the key screens of the application, outlining their specific purpose, functionality, and how they contribute to the user’s journey in obtaining and interpreting polarization data. The figures for these screens are placed together at “Appendix F”.

The application primarily facilitates user interaction through four main screens: the HomeScreen, the UrlSharedScreen, the BatchDetailsScreen, and the critically important AnalysisResultsScreen. A detailed explanation of each screen’s presentation and purpose follows.

3.6.6.1 Home Screen (Figure 15)

The HomeScreen serves as the initial entry point when the application is launched directly by the user (e.g. from the device’s main application menu). This screen functions as a central portal, providing navigation to other key sections of the application. It features a button labelled “Unbatched Articles”, which directs the user to the UrlShared Screen. In this context, the UrlShared Screen is presented without any pre-loaded URL parameter, enabling users to view and manage articles that have been shared with the application but not yet assigned to a specific batch.

Below the “Unbatched Articles” button, a “Batches” section is displayed. This area includes a “Create new batch” button and a list of previously created batches. Clicking on an existing batch from this list navigates the user to the BatchDetailsScreen corresponding to that batch.

- *Create Batch Dialog* (Figure 16): This dialog is invoked when the user taps the “Create New Batch” button, either on the HomeScreen or as an option within the UrlSharedScreen (detailed subsequently). Its purpose is to allow the user to specify and submit a unique name for the new batch, ensuring distinct identification.

- *Delete Batch Dialog* (Figure 17): For each batch listed, an option for deletion is provided. When the delete action is initiated by clicking the delete icon, a confirmation dialog appears. This dialog presents the user with two choices for handling the articles within the batch, in addition to an option to cancel the deletion. The user can decide to either "unbatch" the articles, which reassigns them to a default temporary (tmp), or to delete both the batch and all its contained articles permanently both from local and backend storage. This approach gives the user more control over their data management.

3.6.6.2 Url Shared Screen (Figure 18)

The UrlSharedScreen is the application's entry point when the user shares an article. This screen is parameterised with the incoming URL. Upon initialisation with a shared URL, the application initiates the article pre-processing execution (Section 3.5.1). Following a successful response from the backend, which signifies the completion of initial preprocessing, the article's metadata is persisted to the local Room database. The UI then automatically updates to display the newly shared article.

In addition to the received article, this screen presents a list of all previously shared articles that are currently unbatched, sorted in descending order by their creation (share) date. Each article is rendered as a card element, displaying its creation date and time, its fetched title, and its URL. Users are provided with options to delete individual articles from this list or to select multiple articles for batch assignment.

A pinned "Add to batch" button is positioned at the bottom of this screen, below the list of articles. After selecting one or more articles, the user can click this button, which subsequently presents a bottom sheet dialog (Figure 19). This dialog offers two primary actions: (i) An "Add to new batch" button, that will show the "Create Batch" dialog, and allows the user to simultaneously create the batch and populate it with the selected articles, (ii) A list of the existing batches, to add the articles to a previously created batch.

3.6.6.3 BatchDetails Screen (Figure 20)

The BatchDetailsScreen serves as an intermediary stage, allowing users to review and manage the contents of a specific batch before initiating a polarization analysis. The user

can reach this screen when they select a batch from the list presented on the HomeScreen. The screen at the top displays a header containing details of the selected batch, such as its name, creation date, and number of articles it contains. Below this header, a list of all articles that populate this batch is presented. For each article in this list, options are available to either unbatch (an X icon) or to delete it permanently (a trash bin icon).

At the bottom of this screen, a large, screen-wide "Analyse Batch Polarization" button is featured. Pressing this button is the trigger to start the polarization analysis sequence (Section 3.5.2) for the current batch, and navigates the user to the AnalysisResultsScreen.

3.6.6.4 AnalysisResults Screen (Figure 21 – Figure 23)

This is the most important screen of the app, that displays the results of the whole processing. Its information it displays is as follows:

(i) A batch header with its basic info

(ii) *List of Polarizing Topics* (Figure 21): This list contains card elements, for each distinct polarizing topic identified in the batch. Each card displays the topic's name and an "average polarization score" (scaled from 0 to 100). This score is derived by averaging the polarization index values (retrieved from the backend, originating from POLAR's HAS_POLARIZATION_TOWARDS relationships) associated with that specific topic across all relevant dipoles identified in the batch analysis. The resulting average is then multiplied by 100 to enhance user readability and provide an intuitive measure of polarization intensity.

When the user taps on one of these topic cards, the card expands to reveal a more detailed breakdown. This expanded view presents entities associated with that topic, categorized into three groups based on their respective fellowship's collective attitude (positive, neutral, or negative) towards the topic. These groups are distinguished through color-coding (e.g. green for positive, black or grey for neutral, and red for negative), and arranged in a columnar layout for easy comparison.

(iii) *Summary* (Figure 22): Positioned after the list of polarizing topics, the LLM generated summary (as detailed in Section 3.5.4) is presented. Initially, this summary is displayed in a collapsed state, showing only the first two lines, and a "show more" text. Clicking on the summary expands the card, revealing the full text. This feature provides a human-readable narrative that explains the primary polarization findings.

(iv) *Key Topics and Entities* (Figure 23): At the bottom of the screen, the lists of "Key Topics" and "Key Entities" (returned from the /batch/entities and /batch/topics calls) identified across the batch are displayed. Each item in these lists is accompanied by its number of mentions within the batch. These lists are initially configured to show the top five mentioned items and are designed to be expandable, allowing the user to view a more comprehensive list if more items are available.

The structured presentation on the AnalysisResultsScreen is intentionally designed to offer a clear and progressive disclosure of information. Initially, presenting a more compact and visually appealing display of the polarizing topics and their associated score, then a summary of the findings in a human-readable way for extra explanation, and concludes with some extra details that the user might be interested in. This layered approach empowers the user to readily understand the polarization context of the articles they have chosen to analyse.

3.7 Tools and Technologies Summary

This section provides a consolidated summary of the primary tools, libraries, frameworks, and services utilised throughout the research and development of the "Polarization Detector" system.

3.7.1 Foundational Polarization Analysis (Backend Knowledge Generation)

- *Core Framework*: polarlib (Python library for the POLAR framework)
- *Data Source*: GDELT Project (for initial news article corpus)
- *Entity Linking*: DBpedia Spotlight (run via Docker: dbpedia/dbpedia-spotlight:en)
- *Optimization (Structural Polarization)*: Gurobi Optimizer
- *Development Environment*: Python 3.12.9, Anaconda (for environment management)
- *Operating System*: macOS (with M4 Pro chip)

3.7.2 Backend System (API and Real-time Analysis)

- *Programming Language*: Python
- *Web Framework* (API): Flask

- *Graph Database*: Neo4j Community Edition (run via Docker)
- *LLM Service Integration*: Groq AI API
- *LLM Model*: Meta's "llama-3.3-70b-versatile"
- *Containerization*: Docker (for Neo4j, DBpedia Spotlight, and the main backend API/polarlib processing module)

3.7.3 Frontend Mobile Application "Polarization Detector"

- *Platform*: Android (Native Development)
- *IDE*: Android Studio
- *Programming Language*: Kotlin
- *UI Toolkit*: Jetpack Compose
- *Design Guidelines*: Material Design 3
- *Architecture Pattern*: Model-View-ViewModel (MVVM)
- *Local Database*: Room Persistence Library (SQLite abstraction)
- *Networking*: Retrofit with OkHttp3
- *Background Processing*: WorkManager

Chapter 4

System Evaluation and Limitations

A controlled, manual evaluation was conducted to assess the system's core functionality and the quality of its polarization detection. This involved the creation of 30 distinct batches of articles.

These batches were specifically created and populated to include articles with content that based on the knowledge previously extracted by the full polarlib pipeline, was expected to indicate clear polarization. This allowed for a targeted assessment of whether the application could identify and represent these known polarization dynamics.

The evaluation focused on the system's primary goals:

Efficiency: Achieving near real-time analysis for typical batch sizes.

Accuracy: Ensuring the polarization insights (identified topics, opposing groups, and overall sentiment) align with real-world facts and the knowledge of the graph database.

User Experience: Presenting this complex information in a clear, user-friendly manner within the mobile application.

4.1 Performance and Efficiency Evaluation

A key design goal for the "Polarization Detector" was operational efficiency, aiming for near real-time analysis that would be practical for users. It is imagined that the expected use of the application, would involve analysing batches of no more than 20 articles.
Performance tests were conducted on batches of varying sizes. For instance, the maximum processing time observed for a batch containing 20 articles was approximately 4 seconds. This included backend processing for entity and noun phrase aggregation from pre-processed individual articles, querying the Neo4j knowledge base, and generating the LLM summary.
This response time is considered within acceptable limits for a real-time user experience. This efficiency represents a significant improvement over initial development attempts that integrated full topic identification and clustering (as per the complete polarlib pipeline) for each batch, instead of terminating at the noun phrases extraction, like implemented now. Those earlier approaches required 1-2 minutes for a batch of only ~7 articles. The current architecture, which leverages pre-processing of individual articles and efficient querying of the pre-computed knowledge base, demonstrates a significant enhancement in processing speed, successfully achieving the goal of rapid analysis.

4.2 Qualitative Accuracy Assessment

Assessing the accuracy of detecting political polarization is inherently complex and often qualitative. For this study, a thorough quantitative validation was beyond scope, however, a qualitative assessment was performed on the outputs from the 30 test batches. The manual review of these batches indicated that the system generally produced plausible and contextually relevant polarization analyses. The entities, key topics, and the LLM-generated summaries often aligned with expected real-world political discourse and the known characteristics of the articles within the test batches.
Instances where the output seemed less aligned with real-world facts or expectations were observed. These "mistakes" often appeared to originate from the noise within the initial knowledge base generated by polarlib from the news corpus, rather than fundamental flaws in the system's knowledge extraction and summarisation logic.

Nevertheless, the core proof-of-concept – extracting relevant polarization insights from the structured knowledge base based on user-defined batches – was demonstrated to be functioning effectively.

4.3 UI/UX Considerations

While extensive formal usability studies were not conducted as part of this thesis, considerations for user interface (UI) design and user experience (UX) were addressed during development. To enhance the application's intuitiveness and overall usability, advice was taken from a UI/UX design professional regarding screen layout, navigation, and information presentation. The aim was to create an interface that is both easy to navigate and allows users to clearly understand the polarization analysis results.

4.4 Limitations

While the "Polarization Detector" system successfully demonstrates a novel approach to real-time polarization analysis, several limitations in the current implementation and the broader study should be acknowledged:

i. Static Knowledge base Representation: The foundational polarization knowledge base, generated by the initial polarlib execution, is static. The process of creating this knowledge base from the over 7500 article (covering an months-long date range) was computationally intensive requires many hours to complete. With this implementation, the current system does not dynamically learn from new articles shared by users or update its understanding of the polarization landscape over time. This limits its ability to adapt to evolving discourse without a complete regeneration of the base model.

ii. Restricted Article Accessibility: Some articles cannot be fetched programmatically due to website restrictions or paywalls. Different libraries were tested to fetch them but with no success. This constraint limits the set of articles users can effectively share with the application, resulting to reducing the practical utility and comprehensiveness of the polarization analysis.

iii. Dependency on Upstream Data Quality: The quality of the polarization insights provided by the application is inherently dependent on the accuracy and

comprehensiveness of the knowledge generated by the underlying polarlib framework and the quality of the news corpus it processes. Any noise or biases present in the initial polarlib output will inevitably propagate to the application's analysis. An attempt to minimise the noisy results for the topics is the creation of the meaningful topics list, but this approach can also be considered too restrictive, raising the need for a better, more unsupervised motion of good quality topics, in agreement with POLAR's initial intentions.

Chapter 5

Conlusion

The “Polarization Detector” system described in this thesis tries to address challenge of the growing media polarization, utilizing an existing polarization modeling framework. The system aims to empower individual users with a tool of real-time polarization index indications, on the articles of their everyday consumption. As a proof-of-concept, focusing on the U.S. elections as the knowledge context, a server-client architecture is created, that leverages the polarlib library for the polarization knowledge modelling, stores generated data in a structured graph database (Neo4j), and in real-time can handle articles analysis requests, by fetching the polarization knowledge from the database for insights of polarization within the user-given articles, and in the end display a LLM-generated summary, along with compact, visually appealing data to the user. This approach contributes in fighting the expading partisan polarization by enforcing the user with the ability to detect this polarization, easily from the comfort of their smartphone. The evaluation demonstrated that this system can process typical batches of articles in near real-time, a significant improvement over power-intensive methods requiring full pipeline execution per batch. This work underscores the potential of initially executing time-consuming work and persisting the results in a way that only a lightweight processing is necessary in real-time, output accurate insights.

Bibliography

[1] D. Paschalides, G. Pallis and M. Dikaiakos, “POLAR: A Holistic Framework for the Modelling of Polarization and Identification of Polarizing Topics in News Media”, in *Proc. IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining (ASONAM)*, Virtual Event, Netherlands, 2021.

[2] Facing History & Ourselves, “Political Polarization in the United States”, 2024.

[3] J. Druckman and J. Levy, “Affective Polarization in the American Public”, Institute For Policy Research, Northwestern, 2021.

[4] K. Churcher, “Political polarization”, 2025.

[5] A. Leininger, F. Grünewaldand and N. Buntfuß, “Ideological and affective polarization in multiparty systems”, 2023. [Online]. Available: osf.io/preprints/socarxiv/mz6rs_v1. [Accessed 21 May 2025].

[6] J. K. Kobellarz, M. Brocic, D. Silver and t. H. Dilva, “Bubble reachers and uncivil discourse in polarized online public sphere”, *PloS one,* vol. 19, no. 6, 2024.

[7] F. Milačić, “THE NEGATIVE IMPACT OF POLARIZATION ON DEMOCRACY”, Friedrich Ebert Stiftung, 2021.

[8] R. Henderson, “The Science Behind Why People Follow the Crowd”, Psychology Today, 2017. [Online]. Available: https://www.psychologytoday.com/us/blog/after-service/201705/the-science-behind-why-people-follow-the-crowd. [Accessed 21 May 2025].

[9] A. Patov and ., “Homophily Bias: Preference for Similar Others”, 2024. [Online]. Available: https://www.renascence.io/journal/homophily-bias-preference-for-similar-others. [Accessed May 2025].

[10] B. Baumgaertner and F. Justwan, “The preference for belief, issue polarization, and echo chambers”, *Springer Nature Link,* vol. 200, no. 412, 2022.

[11] t. Carothers and A. O’donohue, Democracies Divided: The Global Challenge of Political Polarization, Brookings Institution Press, 2019.

[12] E. D. Martino, M. Cinelli, R. Cerqueti and W. Quattrociocchi, “Quantifying Polarization: A Comparative Study of Measures and Methods”, 2025.

[13] C. Doherty, J. Kiley and B. Jameson, “Partisanship and Political Animosity in 2016”, 2016.

[14] L. Boxell and M. S. M. J. Gentzkow, “Cross-Country Trends in Affective Polarization”, 2021.

[15] J. A. Piazza, “Drivers of Political Violence in the United States”, *Sage Journalsc,* vol. 42, no. 1, 2022.

[16] M. J. Maggini, D. Bassi, P. Piot, G. Dias and P. Gamallo, “A systematic review of automated hyperpartisan news detection”, *PLoS ONE,* vol. 20, no. 2, 2025.

[17] F. Roscini, “How The American Media Landscape is Polarizing the Country”, Boston University , 2025.

[18] A. R. Argueda, C. T. Robertson, R. Fletcher and R. K. Nielsen, “Echo Chambers, Filter Bubbles, and Polarization: A Literature Review”, *Royal Society Open Science,* vol. 7, no. 4, 2022.

[19] J. Stray, R. Iyer and H. Larrauri, “The algorithmic management of polarization and violence on social media”, Knight First Amendment Institute, Columbia University, 2023.

[20] E. B. Marino, J. M. Benitez-Baleato and A. S. Ribeiro, “The Polarization Loop: How Emotions Drive Propagation of Disinformation in Online Media–The Case of Conspiracy Theories and Extreme Right Movements in Southern Europe”, *Social Sciences,* vol. 13, 2024.

[21] C. Bail, “Exposure to opposing views on social media can increase political polarization”, *Proc. Natl. Acad. Sci. USA ,* vol. 115, no. 37, pp. 9216-9221, 2018.

[22] Z. He, N. Mokhberian, A. Câmara, A. Abeliuk and K. Lerman, “Detecting Polarized Topics Using Partisanship-aware Contextualized Topic Embeddings”, arXiv, 2021.

[23] G. Pennycook and D. Rand, “Fighting misinformation on social media using crowdsourced judgments of news source quality”, *Proc. Natl. Acad. Sci. U.S.A.,* vol. 116, no. 7, pp. 2521-2526, 2019.

[24] AllSides, “About AllSides”, [Online]. Available: https://www.allsides.com/about. [Accessed 21 May 2025].

[25] D. Golay, “The bad news game: a defense against fake news”, *XRDS: Crossroads, The ACM Magazine for Students,* vol. 27, no. 1, pp. 32-33, 2020.

[26] S. Balietti, L. Getoor, D. Goldstein and D. Watts, “Reducing opinion polarization: Effects of exposure to similar people with differing political views”, vol. 118, no. 52, 2021.

[27] A. Morales, J. Losada, R. M. Benito and J. Borondo, “Measuring Political Polarization: Twitter shows the two sides of Venezuela”, *ResearchGate,* vol. 25, 2015.

[28] A. Olieman, H. Azarbonyad, M. Dehghani, J. Kamps and M. Marx , “Entity Linking by Focusing DBpedia Candidate Entities”, in *ERD ‘14 Proceedings of the first international workshop on Entity recognition & disambiguation*, Gold Coast, Australia, 2014.

[29] C. Y. Wijaya, “Semantic Search with Vector Databases”, KDnuggets, 2024.

[30] Neo4j, “Why Neo4j? Top Ten Reasons”, [Online]. Available: https://neo4j.com/top-ten-reasons/. [Accessed 21 May 2025].

[31] “In Tied Presidential Race, Harris and Trump Have Contrasting Strengths, Weaknesses”, Pew Research Center, 2024.

[32] Google, “Room Persistence Library”, 2024. [Online]. Available: https://developer.android.com/training/data-storage/room. [Accessed 21 May 2025].

[33] D. H. Hagos, R. Battle and R. D. B., “Recent Advances in Generative AI and Large Language Models: Current Status, Challenges, and Perspectives”, arXiv, Washington DC, 2024.

[34] Groq, “A New Scaling Paradigm: Meta’s Llama 3.3 70B Challenges ‘Death of Scaling Law’”, 6 Dec 2024. [Online]. Available: https://groq.com/a-new-scaling-paradigm-metas-llama-3-3-70b-challenges-death-of-scaling-law/. [Accessed 21 May 2025].

[35] P. e. a. Sahoo, “A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications”, arXiv, 2024.

[36] B. T. B. e. al., “Language Models are Few-Shot Learners”, 2020.

[37] E. Hellstrom, “Temperature Setting in LLMs: A Comprehensive Guide”, *PromptLayer Blog,* 2025.

[38] E. Koc, “Retrofit And OkHttp”, 2023. [Online]. Available: https://medium.com/@erdi.koc/retrofit-and-okhttp-675d34eb7458. [Accessed 21 May 2025].

[39] Google, “WorkManager Overview”, 2024. [Online]. Available: https://developer.android.com/develop/background-work/background-tasks/persistent/getting-started. [Accessed 21 May 2025].

[40] O. C. Işık, “Introduction to MVVM Architecture”, 2023. [Online]. Available: https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679. [Accessed 21 May 2025].

[41] Android Developers, “Receive simple data from other apps”, [Online]. Available: https://developer.android.com/training/sharing/receive. [Accessed 21 May 2025].

[42] Google Developers, “Schedule tasks with WorkManager”, 2024. [Online]. Available: https://developer.android.com/courses/pathways/android-basics-compose-unit-7-pathway-1. [Accessed 21 May 2025].

Appendix A

Knowledge Gathering and Storage Pipeline

Includes initial pipeline of POLAR, cleaning scripts placement, data preparing for ingestion and ingestion

```
keyword_list = [
    'harris', 'biden', 'trump', 'kamala-harris', 'donald-trump',
    'joe-biden', 'jd-vance', 'j-d-vance', 'tim-walz', 'rfk-jr', 'robert-f-kennedy-jr',
    'presidential-candidate', 'running-mate', 'candidacy', 'vice-president', 'inauguration',
    'presidential-debate', 'endorsement', 'presidential-election', 'usa-election', 'us-election',
    'electoral', 'democrat', 'republican', 'democratic-party', 'republican-party', 'party-convention',
    'presidential-campaign', 'election-campaign', 'nomination-speech','press-conference', 'swing-voter',
    'campaign-rally', 'swing-state', 'president', 'debate',
    'super-tuesday','convention', 'nomination', 'conference'
]


corpus_collector = NewsCorpusCollector(
    output_dir   = OUTPUT_DIR,
    from_date    = date(year = 2024, month = 6, day = 1),
    to_date      = date(year = 2025, month = 1, day = 31),
    keywords     = keyword_list,
    domains      = None
)
```

```
corpus_collector.collect_archives()

corpus_collector.collect_articles(n_threads_fetching=16, n_articles=128)

corpus_collector.parse_articles(n_threads_parsing = 8)

corpus_collector.pre_process_articles()


cleanFiles(OUTPUT_DIR)

kamalaHarrisCorrector(OUTPUT_DIR, n_threads=10)


nlp = spacy.load("en_core_web_sm")

entity_extractor = EntityExtractor(output_dir=OUTPUT_DIR)

entity_extractor.extract_entities(n_processes = 5)

entitiesCorrector(OUTPUT_DIR, "entity_mappings_corrections.json")


noun_phrase_extractor = NounPhraseExtractor(output_dir=OUTPUT_DIR)

noun_phrase_extractor.extract_noun_phrases(n_workers = 8)


topic_identifier = TopicIdentifier(output_dir=OUTPUT_DIR)

topic_identifier.encode_noun_phrases()

topic_identifier.noun_phrase_clustering()


sentiment_attitude_pipeline = SyntacticalSentimentAttitudePipeline(

    output_dir  = OUTPUT_DIR,

    nlp         = nlp,

    mpqa_path   = "subjectivity_clues_hltemnlp05/subjclueslen1-HLTEMNLP05.tff"

)
```

```
sentiment_attitude_pipeline.calculate_sentiment_attitudes(n_threads = 10)


sag_generator = SAGGenerator(OUTPUT_DIR)

sag_generator.load_sentiment_attitudes()

bins = sag_generator.calculate_attitude_buckets(verbose=True)


plt.hist([len(v) for v in sag_generator.pair_sentiment_attitude_dict.values()], log=True)

min_freq = sag_generator.percentile(percentile = 99.2)


sag_generator.convert_attitude_signs(

   bin_category_mapping = {

      “NEGATIVE”:  [(-1.00, -0.02)],

      “NEUTRAL”:   [(-0.02, 0.10)],

      “POSITIVE”:  [(0.10, 1.00)]

   },

   minimum_frequency = min_freq,

   verbose             = True

)


G, node_to_int, int_to_node = sag_generator.construct_sag()

print(f”SAG constructed with {G.number_of_nodes()} nodes (entities) and {G.number_of_edges()} edges.”)


fellowship_extractor = FellowshipExtractor(OUTPUT_DIR)

fellowships          = fellowship_extractor.extract_fellowships(

   n_iter     = 25,

   resolution = 0.7,
```

```
    merge_iter = 20,
    jar_path   ='polarlib/',
    verbose    = True
)

topic_attitude_calculator = TopicAttitudeCalculator(OUTPUT_DIR)
topic_attitude_calculator.load_sentiment_attitudes()
topic_attitude_calculator.get_polarization_topics()
topic_attitude_calculator.get_topic_attitudes()

pkg = PolarizationKnowledgeGraph("outputJuneToJanuary")
pkg.construct()
# Prepare CSVs for ingestion
pkg.export_all("my_exports")

ing = Neo4jIngestor()
# Ingest all data from CSVs to neo4j
ing.ingest_all(import_dir="my_exports")
ing.close()

# By this step we have gathered and stored knowledge in Neo4j
```

Appendix B

PreProcessArticle Implementation Backend

(Simplified Implementation, without loggings or error checks)

```
# polar_api flask endpoint

@app.route('/article/preProcess', methods=['POST'])

def preProcessArticle():

    data = request.get_json() or {}

    url = data.get('url')

    status_code, response = preProcessArticleImpl(url=url, batch_name=TEMP_BATCH)

    return jsonify({'message': response}), status_code




# polar_api_impl

def preProcessArticleImpl(url, batch_name):

    article_collector = ArticleCollector(OUTPUT_DIR, urls=[url], batchName=batch_name)

    article_collector.collect_articles(n_threads_fetching=1)

    article_collector.parse_articles(n_threads_parsing=1)

    article_collector.pre_process_articles(n_threads=1)


    cleanFiles(output_dir=OUTPUT_DIR, urls=[url], batch_name=batch_name)

    kamalaHarrisCorrector(output_dir=OUTPUT_DIR, urls=[url], batch_name=batch_name)
```

```
extractor = EntityExtractor(
    output_dir=OUTPUT_DIR,
    batch_name=batch_name,
    urls=[url],
    spotlight_url='http://spotlight:80/rest/annotate'
)
extractor.extract_entities(n_processes=1)

entitiesCorrector(output_dir=OUTPUT_DIR, urls=[url],
    batch_name=batch_name,
      mappings_file_path="entity_mappings_corrections.json"
)
np_extractor = NounPhraseExtractor(
    output_dir=OUTPUT_DIR,
    urls=[url],
    batch_name=batch_name
)
np_extractor.extract_noun_phrases(n_workers=1, aggregate=False)

return 200, ""
```

Appendix C

Polarization Analysis Backend

(Simplified Implementation, without loggings or error checks)

```
def analyse_batch_polarization_simplified(batch_name: str):
    """
    Highly simplified pipeline for analysing polarization in a batch of articles.
    1. Ensures aggregated entities and topics for the batch are calculated/loaded.
    2. Fetches polarization data from Neo4j ('strict' query).
    3. Builds a prompt from the Neo4j data.
    4. Calls the Groq API to generate a summary.
    5. Returns the summary and the raw data from the Neo4j query.
    """
    calculate_batch_entities(batch_name)
    calculate_batch_meaningful_topics_from_noun_phrases(batch_name)

    with open(ENTITIES_MAPPING_CLEANED_PATH(batch_name), 'r', encoding='utf-8') as f:
        emap = json.load(f)
    with open(TOPICS_MAPPING_CLEANED_PATH(batch_name), 'r', encoding='utf-8') as f:
        tmap = json.load(f)

    batch_entity_ids = list(emap.keys())
    batch_topic_names = list(tmap.keys())

    # Query to neo4j
    query_results = fetch_batch_polarization_data_strict(
        batch_entity_ids,
        batch_topic_names
    )
    ## Save results in a file

    groq_api_client = GroqAPI()
    prompt_text = groq_api_client.prepare_polarization_prompt(query_results)
    ## Save prompt in a file
```

```
    summary = groq_api_client.get_polarization_analysis_summary(batch_name, prompt_text)
    ## Save summary in a file

    return 200, summary, query_results

# Neo4j query for knowledge extraction
def fetch_batch_polarization_data_strict(
    batch_entity_ids,
    batch_topic_names,
    min_pol= 0.2,
) -> dict:
    """
    Strict mode: skip any dipole-topic if either fellowship
    has zero entities existing in the batch.
    """
    driver  = GraphDatabase.driver(NEO4J_URI, auth=basic_auth(NEO4J_USER, NEO4J_PASS))
    session = driver.session()
    try:
        # Fetch dipole–topic polarization above threshold
        q = """
        MATCH (e:Entity)-[:MEMBER_OF]->()-->(d:Dipole)-[r:HAS_POLARIZATION_TOWARDS]->(t:Topic)
          WHERE e.id IN $eids
            AND t.name IN $topics
            AND r.weight > $min_pol
        // only keep each (dipole,topic,pol) once
        RETURN DISTINCT
          d.id     AS dipole,
          t.name   AS topic,
          r.weight AS polarization
        ORDER BY dipole, polarization DESC
        """
        recs = session.run(q, {
            "eids":    batch_entity_ids,
            "topics":  batch_topic_names,
            "min_pol": min_pol
        })

        # Group by dipole
        dipole_map = {}
```

```
        for r in recs:
            dipole_map.setdefault(r["dipole"], []).append({
                "topic":        r["topic"],
                "polarization": r["polarization"]
            })

        output = []
        # For each dipole fetch its two fellowships
        for did, tlist in dipole_map.items():
            for entry in tlist:
                topic = entry["topic"]
                pol   = entry["polarization"]


# Pull fellowships + their batch-entities + collective attitude
                q2 = """
                MATCH (d:Dipole {id:$did})<-[:PART_OF]-(f:Fellowship)
                OPTIONAL MATCH (f)<-[:MEMBER_OF]-(e:Entity)
                  WHERE e.id IN $eids
                OPTIONAL MATCH (f)-[r:HAS_COLLECTIVE_ATTITUDE_TO_TOPIC]->(t:Topic {name:$topic})
                RETURN
                  f.id            AS fellowship,
                  collect(DISTINCT {id:e.id, name:e.name}) AS entities,
                  coalesce(r.weight, 0.0)                  AS attitude
                """
                fellows = []
                for f2 in session.run(q2, {
                    "did":    did,
                    "eids":   batch_entity_ids,
                    "topic":  topic
                }):
                    ents = [m for m in f2["entities"] if m["id"]]
                    fellows.append({
                        "fellowship": f2["fellowship"],
                        "entities":   ents,
                        "attitude":   f2["attitude"]
                    })

                # 5) Strict: skip if either fellowship has zero matched entities
```

```
            if len(fellows) == 2 and all(f["entities"] for f in fellows):
                output.append({
                    "dipole":       did,
                    "topic":        topic,
                    "polarization": pol,
                    "fellowships":  fellows
                })

    return {"dipole_topic_analysis": output}

finally:
    session.close()
    driver.close()
```

Appendix D

List of 51 meaningful topics (> 0.5 polarization index + real-life meaning/value)

gun regulation	birthright citizenship
transportation	mass deportation
personal profit	economy
arrest	ceasefire
climate	impeachment process
climate change	genocide
fentanyl human trafficking	possible replacement
national debt	military aid
election denialism	immigration issue
communism	immigration reform
medication abortion	immigration
migration	illegal immigration
tax evasion	abortion right
criminal activity	southern border
bipartisan border security package	tariff

political violence	covid-19 pandemic
health	health care
artificial intelligence	illegals
border	iran
ongoing war	drug trafficking
war	nuclear power
another trade war	poverty
government efficiency	vaccine
endless war	vaccine skeptic
military aid	assassination attempt
military	

Appendix E

Prompt Generated instruction for LLM

lines = [

“You are an expert in analysing and summarising political polarization from structured data.”,

“Your task is to generate a concise, human-readable summary of polarization from the data below.”,

“This data is aggregated by topic. For each topic, groups of entities are listed under their respective stances (positive, negative, neutral), reflecting their most pronounced attitude if multiple instances of the same group were found.”,

““,

“Key Definitions:”,

“ - **Polarization Index (PI):** For each topic, this is the maximum observed polarization score (0.0 low to 1.0 high), indicating how contested it is.”,

“ - **Stance:** Describes a group’s sentiment towards a topic (score from -1.0 strongly negative to +1.0 strongly positive). Groups of entities are categorized by their most definitive positive, negative, or neutral stance found in the data.”,

““,

“Summary Structure Guidance:”,

“1. Start with an overall assessment of polarization in the batch.”,

“2. Identify the most polarized topics (if there are many, mention them all, but focus on 2-3).”,

“3. For each key topic:”,

“ - State its name and overall polarization level (e.g. ‘high’, ‘moderate’).”,

“ - Describe the groups of entities holding positive stances. List them with their representative entities (as provided) and their interpreted stance.”,

“ - Similarly, describe the groups of entities holding negative stances.”,

“ - Example: ‘The topic of \”Healthcare Reform\” shows high polarization. Groups with a positive stance include: Group (entities: Patients United, Doctors for Reform and others; stance: strongly positive). Groups with a negative stance include: Group (entities: Insurers Alliance, Free Market Medical; stance: negative), Group (entities: Taxpayers Watchdog; stance: strongly negative).’”,

“4. Use clear, narrative language. Synthesize the information rather than just listing data points.”,

“5. Keep the summary concise yet informative. Do not include any assumptions, only knowledge provided here”,

““,

“Aggregated Polarization Data by Topic (Groups per Stance):”,

]

Appendix F

Polarization Detector Application Screenshots

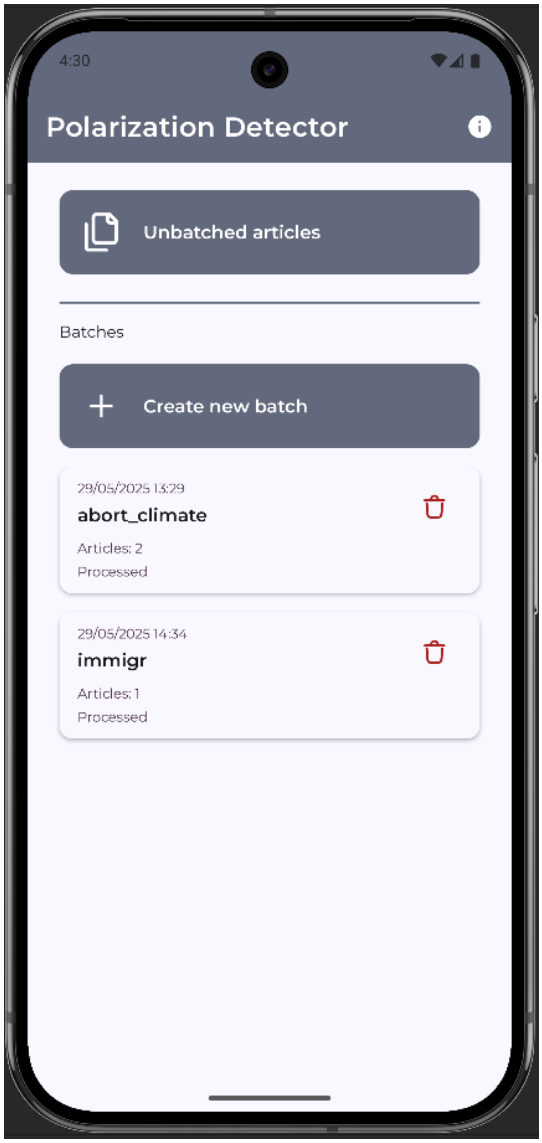

Figure 10 - Home Screen

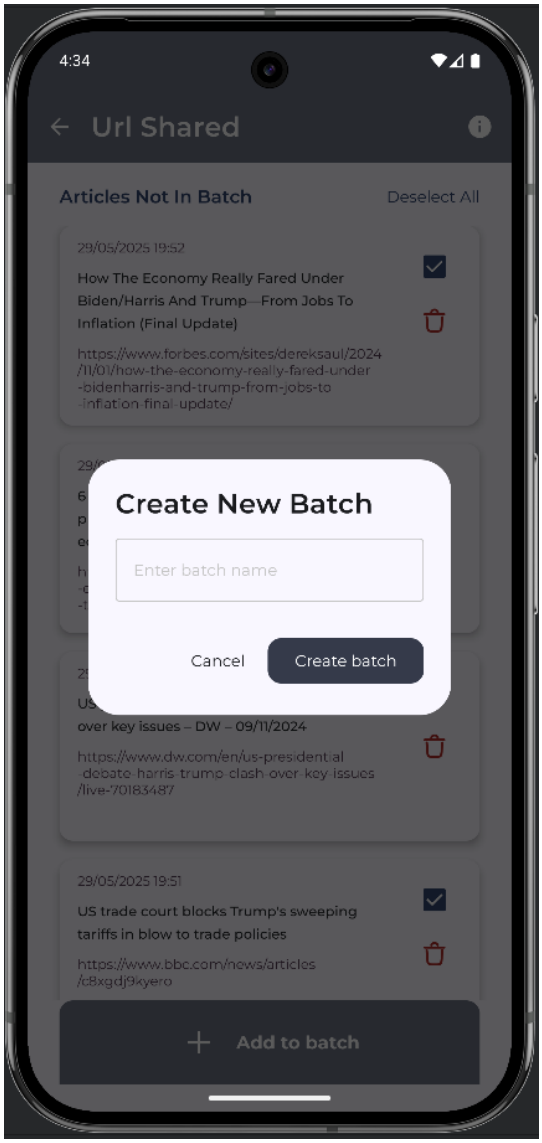

Figure 9 - Create Batch Dialog

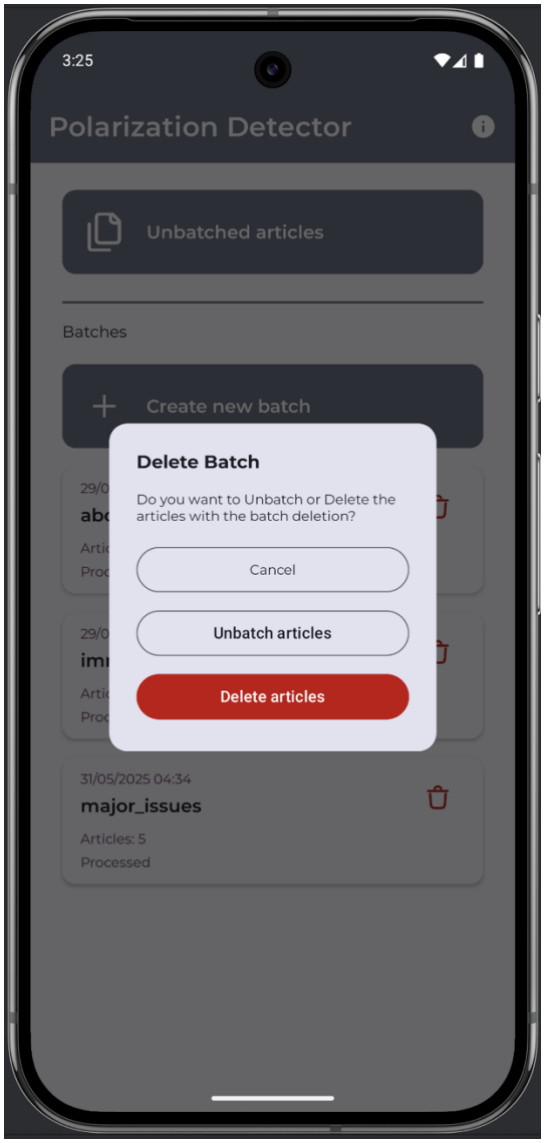

Figure 12 - Delete Batch Dialog

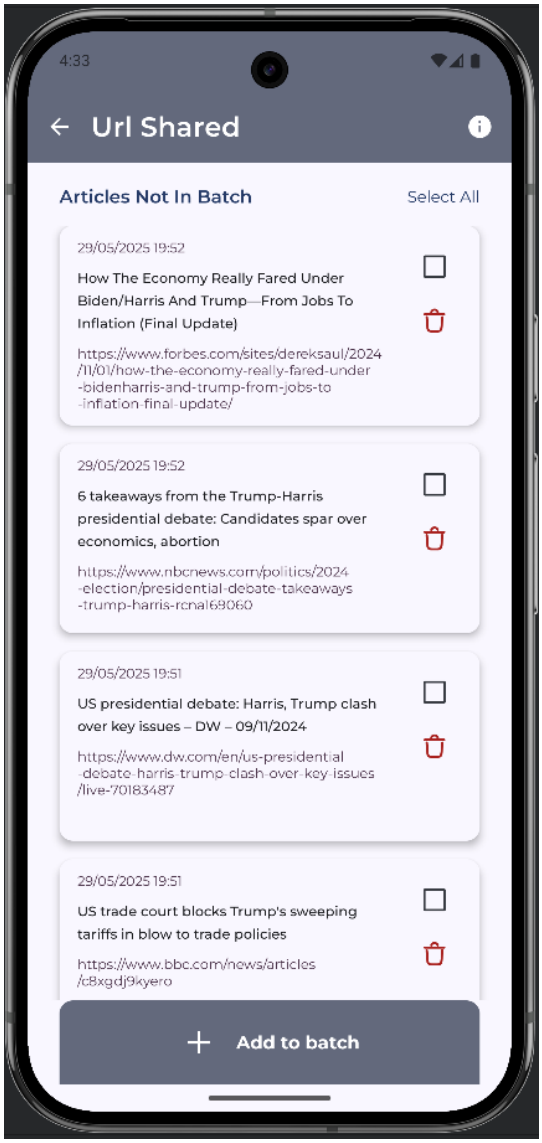

Figure 11 - UrlShared Screen

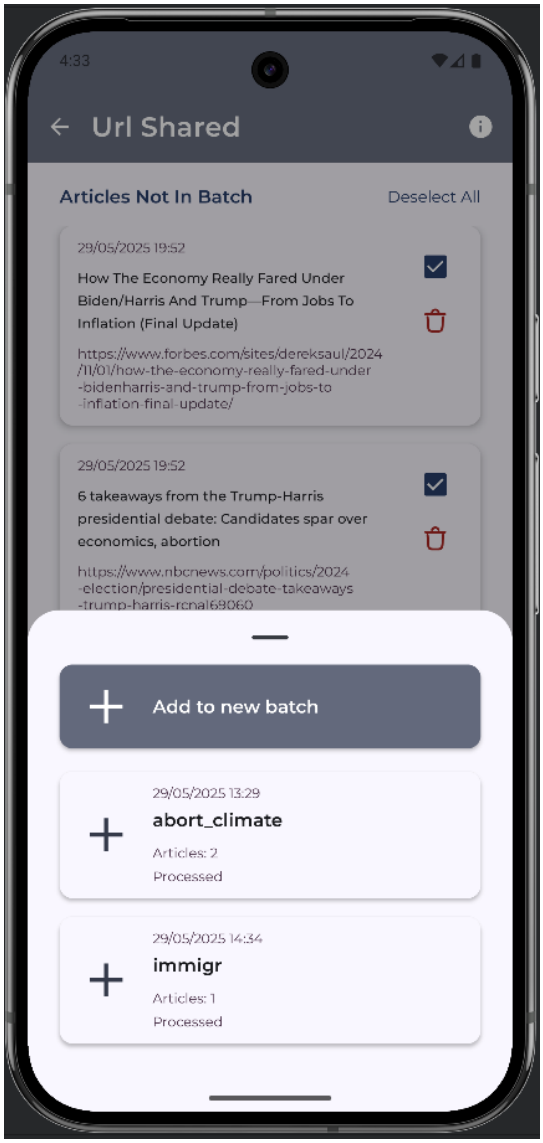

Figure 14 - Add Articles to Batch Bottom Sheet

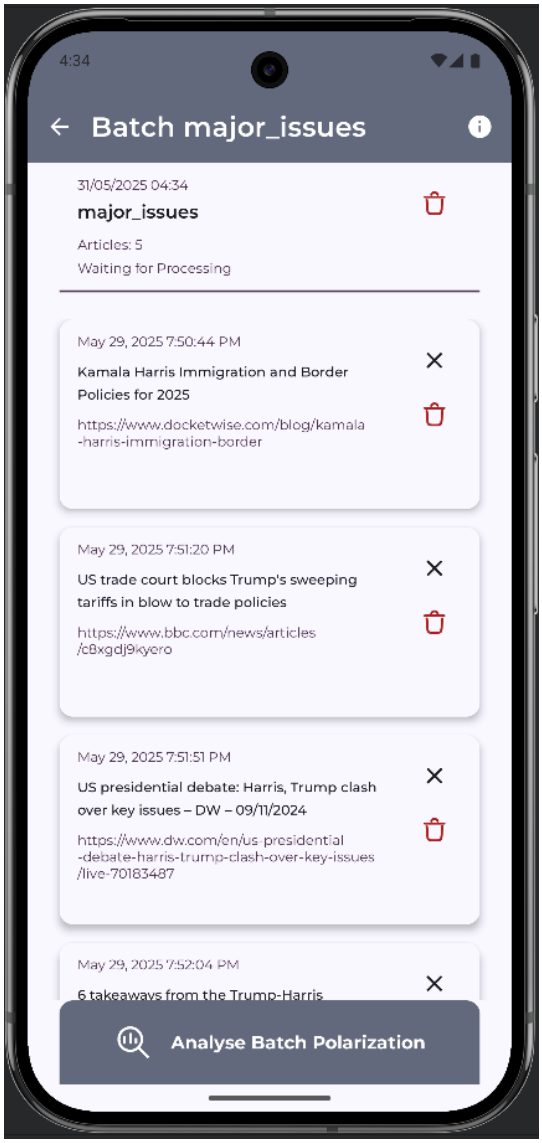

Figure 13 - BatchDetails Screen

Figure 16- AnalysisResults Screen, top part showing list with polarized topics, one of them expanded to display the entities and their stance

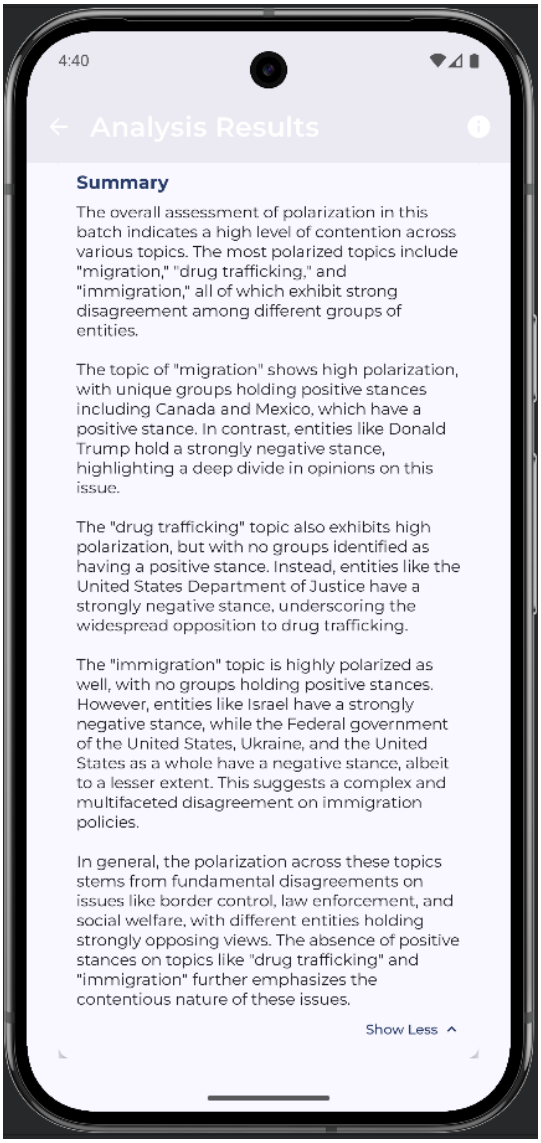

Figure 15 - AnalysisResults Screen showing the expanded summary generated by LLM

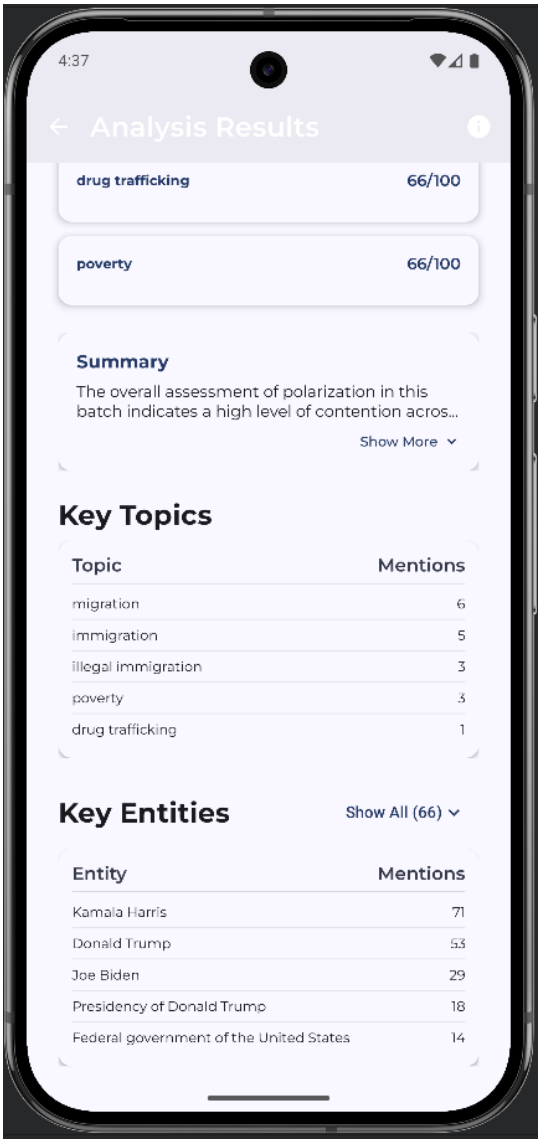

Figure 17 - AnalysisResults Screen, bottom part showing list of topics and entities, with their frequency in the batch