Analyzing Jokerace Submissions: Vision, Mission, and Strategic Priorities

Omniacs.DAO

2025-07-29

Case Study: From Raw Text to Actionable Insights with R and OpenAI

How can you make sense of thousands of free-text submissions from the Arbitrum community? This is a common challenge for decentralized autonomous organizations (DAOs), companies, and projects that rely on community feedback to guide their strategy. Raw text is messy, unstructured, and time-consuming to analyze manually.

In this case study, we'll walk through a complete, real-world workflow for analyzing community proposal submissions from the Arbitrum DAO's Jokerace contests. We will use a powerful combination of classic NLP techniques and modern AI to extract meaningful themes and generate a high-level summary.

You will learn how to:

1. **Preprocess and Clean** raw text data for analysis.
2. Use **N-gram Analysis** to find common multi-word phrases.
3. Apply **Topic Modeling (LDA)** to discover latent themes in the submissions.
4. Leverage the **OpenAI API** to synthesize your findings into a concise, human-readable report.

1. Project Overview and Setup

Objective

Plurality Labs, founded by @DisruptionJoe and supported by his teammates @prose11 | GovAlpha and @radioflyerMQ were given the task to setup experimental governance mechanisms that would allow for formalized "sense-making" that will help the DAO come up with a strong mission and vision to guide their decisions as they then follow up with initiating the capital allocation process where grants will be given out to support the ARB ecosystem.

One of those experiments was an online incentivized survey hosted by "JokerRace" as part of the ThanksARB initiative of Arbitrum's "#GOVMonth". The Jokeraces consisted of 4 separate on-chain surveys where users were prompted to provide feedback on:

- ArbitrumDAO Short-term Strategic Priorities (Reduce Friction)
  - 0xbf47BDA4b172daf321148197700cBED04dbe0D58
- ArbitrumDAO Long-term Strategic Priorities (Growth and Innovation)
  - 0x5D4e25fA847430Bf1974637f5bA8CB09D0B94ec7
- ArbitrumDAO Northstar Strategic Framework: Vision
  - 0x0d4C05e4BaE5eE625aADC35479Cc0b140DDF95D4
- ArbitrumDAO Northstar Strategic Framework: Mission
  - 0x5a207fA8e1136303Fd5232E200Ca30042c45c3B6

The goal of this case-study is to analyze submissions across four Jokerace contests—Vision, Mission, Short-term priorities, and Long-term priorities—to:

1. Identify key themes, concerns, and strategic ideas proposed by the community.

2. Develop a reusable methodology for future text analysis.
3. Create a summarized report of the findings.

R Environment Setup First, let's set up our R environment by loading the necessary libraries. These packages cover everything from data manipulation (`dplyr`, `tidyr`) and text mining (`tm`, `tidytext`) to table formatting (`kableExtra`) and API communication (`httr`).

```
# --- Initial Setup ---
# Set options for the R session
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE)
options(stringsAsFactors = FALSE, scipen = 100, digits = 4)

# --- Load Libraries ---
# For data manipulation
library(dplyr)
library(tidyr)
library(stringr)
library(readr)

# For text mining and NLP
library(tidytext)
library(tm)
library(SnowballC)
library(textclean)
library(topicmodels)

# For tables and visualizations
library(DT)
library(kableExtra)

# For interacting with APIs
library(httr)
library(jsonlite)
```

Loading the Data We'll use one main dataset:

`propfinal`: Contains the core proposal submissions, including the text content, author, and timestamp from JokerAce.

```
# Load Data
propfinal <- readRDS("../data/DataProposalsJokerAce.RDS")

# Output the table within a horizontally scrollable div
# tbl_html <- kable(head(propfinal), "html", escape = FALSE) %>%
#   kable_styling(full_width = FALSE, font_size = 12)

#cat('<div style="overflow-x:auto; max-width:100%;">', as.character(tbl_html), '</div>')
knitr::kable(head(propfinal), format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

2. Data Cleaning and Preprocessing

Text data from the wild is messy. It's filled with HTML tags, punctuation, and inconsistent capitalization. To prepare it for analysis, we need to standardize it through a process called preprocessing. Our pipeline will:

1. Remove HTML tags.

BlockTime	Contract	Address
2023-09-06 09:12:57	0xbf47bda4b172daf321148197700cbed04dbe0d58	0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
2023-09-06 09:13:36	0x5d4e25fa847430bf1974637f5ba8cb09d0b94ec7	0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
2023-09-06 09:14:02	0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4	0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
2023-09-06 09:14:34	0x5a207fa8e1136303fd5232e200ca30042c45c3b6	0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
2023-09-09 11:34:17	0x5a207fa8e1136303fd5232e200ca30042c45c3b6	0x88496a9b5d7543148d9f8aecc9f69cb8437a4bae
2023-09-09 11:34:40	0x5a207fa8e1136303fd5232e200ca30042c45c3b6	0x70f826a221519bd1146901ff973753dd03cf97e0

2. Convert all text to lowercase.
3. Remove punctuation.
4. Stem words to their root form (e.g., "running", "ran", and "runs" all become "run"). This helps group related words.
5. Trim excess whitespace.

```
# --- Clean and Preprocess Text ---

# Apply a series of cleaning functions
propfinal_clean <- sapply(propfinal$Content, function(text) {
  text %>%
    textclean::replace_html() %>%      # Remove HTML tags
    tolower() %>%                      # Convert to lowercase
    removePunctuation() %>%            # Remove punctuation marks
    wordStem(language = "en") %>%      # Stem words to their root
    str_trim()                         # Remove leading/trailing whitespace
})

# Display a before-and-after comparison
df_preview <- data.frame(
  Raw = propfinal$Content[1:5],
  Cleaned = propfinal_clean[1:5],
  stringsAsFactors = FALSE
)
# knitr::kable(df_preview, format = "html", escape = FALSE, row.names = FALSE) %>%
#   kable_styling(full_width = FALSE)

knitr::kable(df_preview, format = "latex", booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Arbitrum DAO's short term is to enhance accessibility, user experience, and community support
Arbitrum DAO's long term is to establish a robust, secure, and globally recognized blockchain ecosystem
Arbitrum DAO's vision is to revolutionize blockchain adoption and transform global finance through decentralized sol
Arbitrum DAO's mission is to drive innovation, foster collaboration, and empower individuals in the decentralized lan
cxvxcvfds

This clean, stemmed text is now ready for more advanced analysis.

3. Exploratory Analysis: N-grams

What are the most common phrases in the submissions? While looking at single words (unigrams) is useful, n-grams (sequences of n words) give us more context. For example, the trigram "reduce transaction fees" is far more insightful than the individual words "reduce," "transaction," and "fees."

Let's find the most common trigrams (3-word phrases), excluding common "stop words" like "the," "a," and "is". We'll also add custom stop words specific to our dataset, like "arbitrum" and "dao," to filter out noise.

```
# --- N-gram Analysis ---

# Create a dataframe from the cleaned text
df <- data.frame(text = propfinal_clean, stringsAsFactors = FALSE)

# Load a standard list of English stopwords
english_stopwords <- readLines("https://slcladal.github.io/resources/stopwords_en.txt",
                               encoding = "UTF-8")
# Add custom stopwords relevant to this specific dataset
custom_stopwords <- c("arbitrumdao", "arbitrum", "project", "arb", "dao")
english_stopwords <- c(english_stopwords, custom_stopwords)

# Generate a table of trigrams
trigram_table <- df %>%
  # Tokenize the text into trigrams (sequences of 3 words)
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  # Separate the trigram into three distinct columns
  separate(trigram, into = c("word1", "word2", "word3"), sep = " ") %>%
  # Remove rows where any of the three words is a stopword
  filter(
    !word1 %in% english_stopwords,
    !word2 %in% english_stopwords,
    !word3 %in% english_stopwords
  ) %>%
  # Remove rows with missing values
  drop_na() %>%
  # Count the occurrences of each unique trigram
  count(word1, word2, word3, sort = TRUE)

# Display the top 15 most frequent trigrams
cat("Top 15 Trigrams:")
```

```
## Top 15 Trigrams:
```

```
knitr::kable(head(trigram_table, 30))
```

word1	word2	word3	n
make	wise	decisions	212
agree	positive	things	135
layer	2	scaling	132
layer	2	solutions	105
shortterm	strategic	priorities	104
longterm	strategic	priorities	89
decentralized	finance	defi	75
regular	security	audits	74
great	potentialthe	teams	69

word1	word2	word3	n
potentialthe	teams	dedication	69
excellent	fan	support	65
2	scaling	solutions	64
2	scaling	solution	62
creating	anti	sybils	58
decentralized	applications	dapps	58
fairly	thriving	community	58
friction	pain	points	54
top	10	finishers	53
make	transparant	activity	51
revolutionize	blockchain	adoption	51
transform	global	finance	51
includes	users	developers	46
kulechov	founder	lensprotocol	44
stani	kulechov	founder	44
decentralized	autonomous	organization	42
btc	touch	30k	40
community	members	developers	37
sybil	security	system	37
target	audience	includes	37
anti	sybil	security	36

This output immediately gives us a feel for recurring suggestions and ideas within the community.

4. Uncovering Themes with Topic Modeling (LDA)

While n-grams show us popular phrases, topic modeling helps us discover the underlying, latent themes across all submissions. We'll use Latent Dirichlet Allocation (LDA), a popular unsupervised algorithm that works by:

Assuming each document is a mix of topics. Assuming each topic is a mix of words. Figuring out the "topics" (which are just clusters of words) that best explain the collection of documents.

The LDA Workflow We'll create a repeatable workflow using a series of functions and a loop to process each contest's submissions.

Step 1: Define Helper Functions We'll start with functions to preprocess text, create a Document-Term Matrix (DTM), and run the LDA model. A DTM is a simple matrix where rows represent documents (our proposals), columns represent terms (words), and the cells contain the word counts.

```
# --- LDA Helper Functions ---

# Function to preprocess a vector of texts for topic modeling
preprocess_text <- function(texts, stopwords) {
  corpus <- Corpus(VectorSource(texts)) # Create a corpus object
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removeWords, stopwords)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, stemDocument, language = "en")
  corpus <- tm_map(corpus, stripWhitespace)
  return(corpus)
}
```

```
# Function to create a Document-Term Matrix (DTM)
get_DTM <- function(corpus, min_freq = 5) {
  DTM <- DocumentTermMatrix(corpus, control = list(bounds = list(global = c(min_freq, Inf))))
  # Remove empty rows (documents with no terms after cleaning)
  sel_idx <- slam::row_sums(DTM) > 0
  DTM <- DTM[sel_idx, ]
  return(list(DTM = DTM, sel_idx = sel_idx))
}

# Function to run the LDA algorithm
run_lda <- function(DTM, K, iter = 5) { # Increased iterations for better convergence
  set.seed(1234) # for reproducibility
  topicModel <- LDA(DTM, k = K, method = "Gibbs", control = list(iter = iter, verbose = 0))
  return(topicModel)
}

# Function to extract top terms from the model
get_top_terms <- function(model, n = 20) {
  as.data.frame(terms(model, n))
}
```

Step 2: Set Up and Run the Analysis Loop Now, we define the contests we want to analyze and the number of topics (K) we want to find for each. Choosing K is a mix of art and science; it often requires experimentation to find a number that produces coherent, distinct topics.

The loop will perform the following for each contest:

1. Subset the proposals for that contest.
2. Perform final cleaning and filtering.
3. Run the preprocess_text and get_DTM functions.
4. Run the LDA model using our run_lda function.
5. Assign the most likely topic to each proposal.
6. Extract the top words for each discovered topic.

```
# --- Define Contests and K (Number of Topics) ---
contestdf <- do.call(rbind, list(
  data.frame(Address = "0xbf47bda4b172daf321148197700cbed04dbe0d58",
             Name = "Reduce Friction",
             Slug = "RF4",
             Topics = 4),
  data.frame(Address = "0x5d4e25fa847430bf1974637f5ba8cb09d0b94ec7",
             Name = "Growth and Innovation",
             Slug = "GI5",
             Topics = 5),
  data.frame(Address = "0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4",
             Name = "Vision",
             Slug = "V7",
             Topics = 7),
  data.frame(Address = "0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4",
             Name = "Vision",
             Slug = "V10",
             Topics = 10),
  data.frame(Address = "0x5a207fa8e1136303fd5232e200ca30042c45c3b6",
```

```
            Name = "Mission",
            Slug = "M11",
            Topics = 11),
  data.frame(Address = "0x5a207fa8e1136303fd5232e200ca30042c45c3b6",
            Name = "Mission",
            Slug = "M15",
            Topics = 15)
))


# Create a directory to store outputs
sapply(file.path("../project1/temp/txts", contestdf$Slug), dir.create, recursive = TRUE, showWarnings =
```

```
## ../project1/temp/txts/RF4 ../project1/temp/txts/GI5  ../project1/temp/txts/V7
##                     FALSE                     FALSE                     FALSE
## ../project1/temp/txts/V10 ../project1/temp/txts/M11 ../project1/temp/txts/M15
##                     FALSE                     FALSE                     FALSE
```

```
# --- Main LDA Analysis Loop ---
results <- tibble(TermIdx = 1:50)
propout <- propfinal

for (idx in seq_len(nrow(contestdf))) {
  # --- 1. Subset text for this contest address ---
  address_mask <- propfinal$Contract == contestdf$Address[idx]
  textdata <- propfinal$ContentParsed[address_mask]
  original_idx <- which(address_mask)

  # --- 2. Clean encoding ---
  textdata <- iconv(textdata, from = "", to = "UTF-8", sub = "")  # Fix encoding
  long_mask <- nchar(textdata) > 40
  textdata <- textdata[long_mask]
  original_idx <- original_idx[long_mask]
  # Remove duplicates (keep first occurrence)
  dup_mask <- !duplicated(textdata)
  textdata <- textdata[dup_mask]
  original_idx <- original_idx[dup_mask]

  if (length(textdata) < contestdf$Topics[idx]) next # Skip if not enough docs

  docdf <- tibble(
    Contest = contestdf$Name[idx],
    doc_id = seq_along(textdata),
    text = textdata,
    original_idx = original_idx
  )

  # --- 3. Preprocess and DTM ---
  corpus <- preprocess_text(docdf$text, english_stopwords)
  dtm_out <- get_DTM(corpus, min_freq = 5)
  DTM <- dtm_out$DTM
  docdf <- docdf[dtm_out$sel_idx, ]

  if (nrow(docdf) < contestdf$Topics[idx]) next # Skip if too few docs after pruning
```

```
  # --- 4. Topic modeling ---
  K <- contestdf$Topics[idx]
  topicModel <- run_lda(DTM, K)
  tmResult <- posterior(topicModel)
  theta <- tmResult$topics
  docdf$Topic <- apply(theta, 1, which.max)

  # --- 5. Annotate main dataframe with topic assignment (use original_idx!) ---
  propout[docdf$original_idx, contestdf$Slug[idx]] <- docdf$Topic

  # --- 6. Save per-topic submissions as text files ---
  for (tidx in unique(docdf$Topic)) {
    tempd <- docdf[docdf$Topic == tidx, ]
    outdir <- file.path("../project1/temp/txts", contestdf$Slug[idx])
    dir.create(outdir, recursive = TRUE, showWarnings = FALSE)
    writeLines(
      paste0("Submission : ", 1:nrow(tempd), "\n", tempd$text, collapse = "\n\n\n"),
      file.path(outdir, paste0("Topic-", tidx, ".txt"))
    )
  }

  # --- 7. Save top terms ---
  result_t <- get_top_terms(topicModel, 50)
  names(result_t) <- paste0(contestdf$Slug[idx], "_Topic_", seq_len(K))
  results <- bind_cols(results, result_t)
  message("Finished: ", contestdf$Name[idx])
}
```

TermIdx	RF4_Topic_1	RF4_Topic_2	RF4_Topic_3	RF4_Topic_4	GI5_Topic_1	GI5_Topic_2	GI5_Top
1	communiti	user	develop	communiti	develop	develop	develop
2	user	develop	user	user	secur	communiti	blockchai
3	blockchain	make	communiti	develop	blockchain	project	user
4	develop	communiti	secur	govern	communiti	ecosystem	communit
5	secur	address	ensur	ecosystem	enhanc	user	ecosystem
6	support	secur	blockchain	ensur	fund	research	network

By looking at the word clusters for each topic (e.g., RF4_Topic_1), we can assign a human-readable label. For instance, a topic with words like "communiti," "blockchain," "develop," "secur," and "support" is possibly about Governance.

5. Synthesizing Insights with the OpenAI API

We now have structured data: n-grams and topics. But we still need to produce a high-level summary. This is where an LLM like GPT-4o mini shines. We will ask the AI to act as an analyst, providing it with our topic keywords and the raw submissions, and instructing it to generate a synthesized report.

Step 1: Prepare the Prompt The key to getting a good result from an LLM is a well-structured prompt. Our prompt will:

1. Assign a role: "You are an expert blockchain community analyst."
2. Provide context: Give it the top keywords from our LDA model.
3. Provide the data: Give it the raw text submissions.

4. Give clear instructions: Ask it to identify themes and find standout submissions.
5. Specify the output format: Request clear headings for easy reading.

```
# --- OpenAI Summarization ---

# Gather the data for the prompt
# Let's focus on the "Reduce Friction" contest (RF4)
# Note: In a real scenario, you'd load the text files you saved during the LDA loop.
# For this example, we will use a placeholder.

# Extract top words for the "Reduce Friction" contest
top_words <- unlist(results[, grep("^RF4_Topic_", colnames(results))])
top_words <- unique(top_words[!is.na(top_words)])

# Load all submissions for 'Reduce Friction'
submissions <- c(
  readLines("../project1/temp/txts/RF4/Topic-1.txt"),
  readLines("../project1/temp/txts/RF4/Topic-2.txt"),
  readLines("../project1/temp/txts/RF4/Topic-3.txt"),
  readLines("../project1/temp/txts/RF4/Topic-4.txt")
)
# Remove blank lines
submissions <- submissions[nchar(submissions) > 0]

# Craft the prompt
prompt <- paste(
  "You are an expert blockchain community analyst. Given the following user submissions",
  "and most common keywords,",
  "create a detailed summary for each of these categories:",
  "1. Reduce Friction (overall)\n2. Growth and Innovation\n3. Vision\n4. Mission",
  "\n\nFor each category, do the following:\n- Identify and describe the main themes,",
  "concerns, and opportunities found in the submissions.",
  "- Provide at least 2-3 representative or standout submissions (either as direct quotes",
  "or short paraphrases) that best capture the spirit or key insights of that category.",
  "- Where relevant, highlight points of consensus, strong opinions, or",
  "recurring challenges and suggestions.",
  "\n\nMost common keywords:\n", paste(top_words, collapse = ", "),
  "\n\nUser submissions:\n", paste(submissions, collapse = "\n"),
  "\n\nFormat your output with clear headings for each category, and",
  "organize within each section as:",
  "- Themes:\n  [Detailed synthesis]\n- Standout submissions:",
  "\n  - [Quote or paraphrase 1]\n  - [Quote or paraphrase 2]\n  - [Quote or paraphrase 3]",
  "\nIf a submission is relevant to more than one category, you may include it in multiple sections."
)
```

Step 2: Make the API Call Next, we send our prompt to the OpenAI API using the `httr` package. For this exercise, you will need to aquire an API key from OpenAI. Prepaying for API credits is required. We are using `gpt-4o-mini`, but OpenAI has many available options.

Important: Never hardcode your API key directly in your script. For ease of use, load it from a separate, secure file (that you don't share or commit to version control). For a more advanced secure approach, store your API key as a system environment variable.

```
# --- Call OpenAI API ---
```

```
# Load your API key securely from a file
openai_api_key <- readLines("../../openai_key.txt", warn = FALSE)

# Make the POST request to the OpenAI API
response <- POST(
  url = "https://api.openai.com/v1/chat/completions",
  add_headers(
    Authorization = paste("Bearer", openai_api_key)
  ),
  content_type_json(),
  body = toJSON(list(
    model = "gpt-4o-mini", # A powerful and cost-effective model
    messages = list(list(role = "user", content = prompt)),
    temperature = 0.5, # Lower temperature for more focused, less creative output
    max_tokens = 1500
  ), auto_unbox = TRUE)
)

# Parse and display the response
if (http_status(response)$category == "Success") {
  result <- fromJSON(rawToChar(response$content))
  # wrap in our CSS box
  cat("<div class='output-box'>\n")
  cat(result$choices$message$content, "\n")
  cat("</div>\n")
} else {
  cat("<div class='output-box'>\n")
  cat("Error calling OpenAI API:\n")
  cat(rawToChar(response$content), "\n")
  cat("</div>\n")
}
```

1. Reduce Friction (Overall)

Themes:

The submissions highlight a strong consensus on the need to improve user experience and accessibility within the Arbitrum ecosystem. Key areas of focus include simplifying the onboarding process, reducing gas fees, enhancing developer support, and improving communication channels. The community emphasizes the importance of user-friendly interfaces and comprehensive documentation to lower barriers for new users and developers. Additionally, there is a call for increased transparency in governance processes to foster trust and engagement among community members.

Standout submissions:

- "User onboarding complexity is a significant barrier. ArbitrumDAO should simplify the onboarding process for new users by developing user-friendly guides, tutorials, and documentation."
- "High gas fees on the Ethereum network can deter users. ArbitrumDAO should subsidize gas fees or explore Layer 2 solutions to mitigate this issue."
- "Improving the user experience is crucial for widespread adoption. ArbitrumDAO should invest in developing user-friendly interfaces, documentation, and educational resources to lower the barrier to entry."

2. Growth and Innovation

Themes:

Submissions reflect a strong desire for innovation and growth within the Arbitrum ecosystem. There is a clear call for strategic partnerships and collaborations with other blockchain projects to enhance interoperability and expand use cases. Many submissions advocate for funding developer initiatives, including grants and hackathons, to stimulate innovation. The community also emphasizes the importance of promoting educational initiatives to raise awareness about Arbitrum's advantages, thereby driving user adoption.

Standout submissions:

- "Fostering a vibrant developer community is essential. ArbitrumDAO should allocate resources to support developers, such as providing comprehensive developer documentation, offering grants or funding for innovative projects."
- "Establish partnerships with other projects and ecosystems to enhance interoperability and expand the utility of Arbitrum."
- "Invest in educational initiatives to raise awareness about the benefits of Arbitrum and how to use it effectively."

3. Vision

Themes:

The vision articulated by the community emphasizes creating a decentralized and user-centric ecosystem that prioritizes transparency, inclusivity, and innovation. Submissions highlight the importance of community engagement and the need for a robust governance framework that reflects the values of the DAO. There is a strong belief that by aligning actions with the DAO's vision, Arbitrum can foster a lasting impact and legacy within the blockchain space.

Standout submissions:

- "Our vision for ArbitrumDAO is clear: to become a beacon of decentralized innovation and collaboration in the blockchain ecosystem."
- "Keeping our Strategic Framework: Vision, Mission, and Values alive becomes even more critical as we continue to grow."
- "When our actions reflect our values, we build trust with our members, builders, end-users, and partners."

4. Mission

Themes:

The mission articulated in the submissions focuses on empowering users and developers through education, support, and community engagement. There is a shared commitment to ensuring the security and efficiency of the Arbitrum network while promoting decentralization and inclusivity. The community emphasizes the importance of creating a sustainable ecosystem that supports innovation and addresses the evolving needs of its users.

Standout submissions:

- "Our mission is to empower the global community to shape the future of decentralized finance, governance, and beyond."
- "To ensure long-term success, ArbitrumDAO should address immediate friction points, pain points, and barriers affecting our community members, developers, and critical stakeholders."

- "Community engagement is pivotal for long-term success. ArbitrumDAO should allocate resources towards organizing events, hackathons, meetups, and educational initiatives to foster collaboration."

By synthesizing these themes and standout submissions, we can see a clear direction for ArbitrumDAO that prioritizes user experience, community engagement, and innovative growth strategies.

6. Conclusion and Recommendations

By combining traditional NLP techniques with the power of modern LLMs, we have moved from thousands of raw text submissions to a concise, actionable summary of community feedback.

Key Findings

- Identified Core Themes: Using LDA, we successfully identified and categorized the main topics of conversation, such as governance, onboarding, and developer tooling.
- Pinpointed Specific Suggestions: N-gram analysis helped highlight specific, recurring phrases and proposals.
- Synthesized Actionable Insights: The OpenAI API provided a high-quality narrative summary, saving hours of manual reading and interpretation.

Recommendations for Future Analysis

- Improve Prompting: To get more structured data from the community, design submission forms with clearer, more focused prompts.
- Automate Filtering: Use regular expressions and readability metrics to automatically filter out low-quality or spam submissions.
- Integrate Voting Data: Correlate the topics of proposals with their voting outcomes to see which ideas gained the most traction.