Analyzing Snapshot Rankings: Short-term and Long-term Priorities for Arbitrum DAO

Omniacs.DAO

2025-07-29

Contents

Project Overview **1**
Objective . . . 1
Activities & Timeline . . . 1

Background **2**

Libraries and Setup **2**
Connect to the Snapshot GraphQL API . . . 3
Download and Prepare Proposals . . . 3
Download and Prepare Vote Data . . . 4

Analyze and Rank Statements **5**
Discussion and Insights . . . 6

OpenAI Analysis **8**
Extracting Choice Labels Programmatically . . . 8
Building a Consensus Ranking Prompt for OpenAI . . . 9
Calling the GPT-4o-mini Consensus Function . . . 11
Interpretation . . . 11

Conclusions **12**

Project Overview

Objective

This analysis helps **Plurality Labs** and the **Arbitrum DAO** compare strategic framework results on two platforms—**Ethelo** and **Snapshot**. The aim is to determine which channel yields the best quality input. We focus on:

- Comparing contributions by delegate and general contributor wallets
- Revealing similarities and differences in priorities
- Enabling data-driven refinement of the strategic framework
- Sharing all results on a public hub

Activities & Timeline

- **Data procurement**: Collect aggregate Snapshot data
- **Data processing**: Format and clean for analysis

- **Feature engineering**: Calculate new summary metrics
- **Ranking scheme**: Develop both equal-weighted and token-weighted rankings
- **Analysis**: Compare how priorities differ depending on weighting
- **Sharing**: Publish data and analysis

Background

During #GovMonth, contributors ranked statements related to "Growth and Innovation" and ways Arbitrum could "Reduce Friction". Respondents ordered statements by preference. The main questions:

Growth + Innovation Statements

1. Develop accountability practices within ArbitrumDAO.
2. Identify the key capabilities for improved DAO performance.
3. Form alliances with legacy institutions.
4. Fund projects for cross-chain compatibility.
5. Improve gas fee optimization.
6. Define growth strategies.
7. Incentivize users and builders.
8. Scale the platform.
9. Offer educational opportunities.
10. Evolve governance capabilities.

Reducing Friction Statements

1. Build a robust developer community.
2. Make Arbitrum more accessible for developers.
3. Create an inclusive environment.
4. Encourage meaningful DAO participation.
5. Raise awareness of opportunities.
6. Prioritize gas fee optimization.
7. Ensure regulatory compliance.
8. Build anti-Sybil protections.
9. Uphold transparency.
10. Improve token distribution equity.

Each respondent's data includes their wallet, rankings, and Arbitrum holdings (for weighting). Below is a snippet of the GovMonth proposal. For this task, voters were prompted with two sets of statements related to "Growth and Innovation" and ways Arbitrum could "Reduce Friction". The survey respondents were to rank order each statement according to their personal preference. The statements for each are as follows:

Libraries and Setup

Below, we load the necessary `R` packages for web requests, data wrangling, and working with GraphQL APIs.

```
# Load essential libraries
library(jsonlite)   # For parsing JSON data
library(httr)       # For making web requests (not always required, but useful)
library(lubridate)  # For handling date-times
library(ghql)       # For making GraphQL queries
library(dplyr)      # For data manipulation
library(ggplot2)    # For visualization
library(tidyr)      # For formatting data
```

Connect to the Snapshot GraphQL API

We use `ghql` to create a client for querying the Snapshot API. We'll define the queries for:

- DAO spaces
- Proposals
- Votes

```
# Create a GraphQL client for Snapshot.org
# Initialize GraphQL client
con <- GraphqlClient$new("https://hub.snapshot.org/graphql")

# Prepare GraphQL queries
qry <- Query$new()

# Define queries for spaces, proposals, and votes
qry <- Query$new()

# Query for fetching DAO spaces
qry$query('space_data',
    'query space_data($skip:Int!){
        spaces(orderBy: "id", orderDirection: asc,first:1000,skip:$skip){
            id name private about avatar website twitter github coingecko email
            network symbol domain proposalsCount activeProposals followersCount
            votesCount verified flagged rank
        }
    }'
)

# Query for fetching proposals from a given space
qry$query('prop_data',
    'query prop_data($slugid: String!, $timestamp: Int!){
        proposals(orderBy: "created", orderDirection: asc,first:1000,
            where:{space:$slugid,created_gt:$timestamp}) {
            id space{id} ipfs author created network type title body start end
            state votes choices scores_state scores
        }
    }'
)

# Query for fetching individual votes for a proposal
qry$query('vote_data',
    'query vote_data($propid: String!, $timestamp: Int!){
        votes(orderBy: "created", orderDirection: asc,first:1000,
            where:{proposal:$propid,created_gt:$timestamp}) {
            id proposal{id} ipfs voter created choice vp
        }
    }'
)
```

Download and Prepare Proposals

We'll now write functions to download:

- All spaces (optional, for exploration)

- All proposals in a space (filtered to two proposals of interest)
- All votes for a proposal

Each function contains step-by-step comments explaining the loop logic and purpose. We then filter the proposals to the two relevant ones for the analysis.

```
# Function to fetch all proposals for a given DAO space
get_proposals <- function(slug) {
  c_timestamp <- 0
  prop_data <- data.frame()
  while (TRUE) {
    pd_t <- fromJSON(con$exec(qry$queries$prop_data,
                              list(slugid = slug, timestamp = c_timestamp)))$data$proposals
    if (length(pd_t) == 0) break
    prop_data <- bind_rows(prop_data, pd_t)
    c_timestamp <- as.numeric(tail(pd_t$created, 1))
    #message(paste0("Fetched ", nrow(prop_data), " Entries"))
  }
  prop_data$space_id <- prop_data$space$id
  prop_data$space <- NULL
  return(prop_data)
}

# Function to fetch all proposals for a given DAO space# Get proposals for Arbitrum DAO
prop_df <- get_proposals("arbitrumfoundation.eth")

# Focus on two proposals of interest
prop_df_sub <- prop_df[prop_df$id %in% c(
  "0x14e71f784e880170972572c2696ef53ef437700c637a151b5176a5827fe5b8bc",
  "0x5824d0b51cc435a49f6455ee2715216d6b958637218ed79e3e93c41af6bdef33"
), ]
```

Download and Prepare Vote Data

This function retrieves all votes for each proposal. Pagination is used to ensure we get all data, and results are stored in a list for each proposal.

```
get_votes <- function(prop) {
  c_timestamp <- 0
  vote_data <- data.frame()
  while (TRUE) {
    vd_t <- fromJSON(con$exec(qry$queries$vote_data,
                              list(propid = prop, timestamp = c_timestamp)))$data$votes
    if (length(vd_t) == 0) break
    vote_data <- bind_rows(vote_data, vd_t)
    c_timestamp <- as.numeric(tail(vd_t$created, 1))
  }
  vote_data$prop_id <- vote_data$proposal$id
  vote_data$proposal <- NULL
  return(vote_data)
}

# Download votes for both proposals and combine into one dataframe
vote_l <- lapply(prop_df_sub$id, get_votes)
vote_df <- do.call(rbind, vote_l)
```

Analyze and Rank Statements

The next step is to calculate various summary statistics for each statement in each proposal, both by unweighted and token-weighted votes. Comments clarify every operation.

```
vote_df14 <- vote_df[vote_df$prop_id ==
                       "0x14e71f784e880170972572c2696ef53ef437700c637a151b5176a5827fe5b8bc", ]
vote_df58 <- vote_df[vote_df$prop_id ==
                       "0x5824d0b51cc435a49f6455ee2715216d6b958637218ed79e3e93c41af6bdef33", ]


# Helper to unpack choices and compute stats
compute_ranking <- function(vote_df) {
  n_choices <- 10
  rankings <- do.call(rbind, lapply(vote_df$choice, function(x) match(1:n_choices, x)))
  vp_weights <- vote_df$vp / sum(vote_df$vp)

  data.frame(
    Choice = 1:n_choices,
    SumRank = apply(rankings, 2, sum),
    SumRankAvg = apply(rankings, 2, mean),
    PctFirst = sapply(1:n_choices, function(x)
      mean(sapply(vote_df$choice, function(z) z[1] == x))),
    PctLast = sapply(1:n_choices, function(x)
      mean(sapply(vote_df$choice, function(z) z[10] == x))),
    PctFirstFive = sapply(1:n_choices, function(x)
      mean(sapply(vote_df$choice, function(z) x %in% z[1:5]))),
    VPRankAvg = apply(rankings, 2, function(rk) sum(rk * vp_weights))
  )
}

rankraw14 <- compute_ranking(vote_df14)
rankraw58 <- compute_ranking(vote_df58)

# Display result tables (add your statement labels for final output)
knitr::kable(rankraw14[order(rankraw14$SumRank), ], caption = "Growth & Innovation Rankings")
```

Table 1: Growth & Innovation Rankings

	Choice	SumRank	SumRankAvg	PctFirst	PctLast	PctFirstFive	VPRankAvg
6	6	142400	3.455388	0.4368979	0.0307685	0.7154158	4.344264
1	1	175648	4.262163	0.1900221	0.1059426	0.6648225	2.551343
5	5	209517	5.084007	0.0653466	0.0457645	0.5910801	2.989967
4	4	214144	5.196282	0.0531897	0.0457645	0.5737303	4.596209
2	2	220463	5.349615	0.0380238	0.0557861	0.5260974	3.746507
3	3	225679	5.476184	0.0415666	0.0530684	0.5144015	5.509835
7	7	237660	5.766907	0.0445997	0.0564898	0.4521366	8.020223
10	10	275882	6.694378	0.0599597	0.2809929	0.3587149	8.667787
8	8	277625	6.736672	0.0387033	0.1914052	0.3220014	7.789604
9	9	287587	6.978404	0.0316906	0.1340176	0.2815996	6.784262

```
knitr::kable(rankraw58[order(rankraw58$SumRank), ], caption = "Reducing Friction Rankings")
```

Table 2: Reducing Friction Rankings

	Choice	SumRank	SumRankAvg	PctFirst	PctLast	PctFirstFive	VPRankAvg
5	5	145120	3.491735	0.3521089	0.0298116	0.7821997	6.845235
7	7	182440	4.389692	0.1522581	0.0378239	0.6339597	3.065252
1	1	190475	4.583023	0.1917423	0.1100792	0.6406727	5.214026
4	4	221708	5.334520	0.0467024	0.0470874	0.5321576	6.498827
3	3	227369	5.470730	0.0426361	0.0621977	0.5274416	3.582184
6	6	232634	5.597411	0.0537042	0.0534395	0.4413513	3.468893
2	2	235382	5.663531	0.0378480	0.0616443	0.4673853	6.721344
8	8	270473	6.507856	0.0372224	0.0961478	0.3314887	8.156887
10	10	276242	6.646664	0.0559659	0.2896225	0.3954428	5.627098
9	9	304012	7.314838	0.0298116	0.2121460	0.2479007	5.820256

Discussion and Insights

Example Interpretation For “Reducing Friction”, “Prioritize Gas Fee Optimization” was ranked highest when votes are equally weighted.

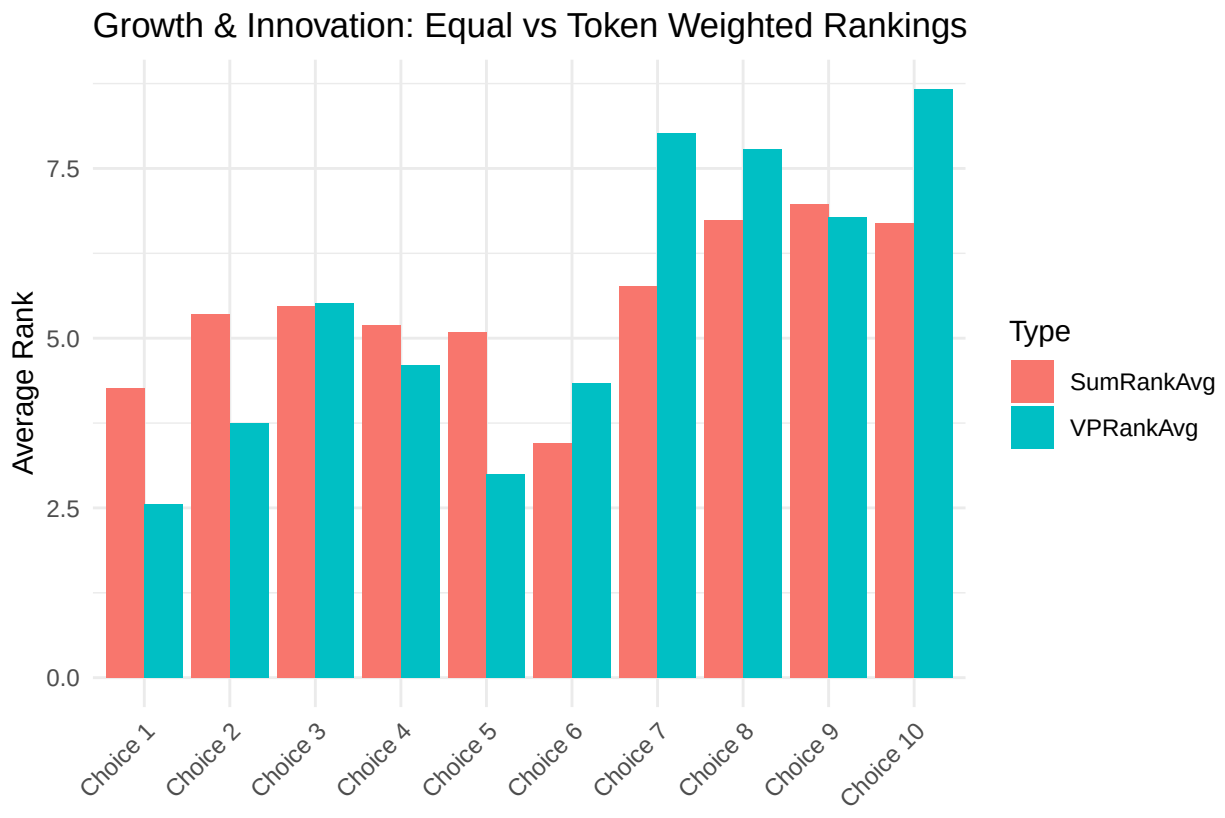
When votes are weighted by tokens, “Build a robust community of developers” rises to the top.

This shows that community and large holders align on the top priorities, but token weighting can shift the relative importance of specific statements.

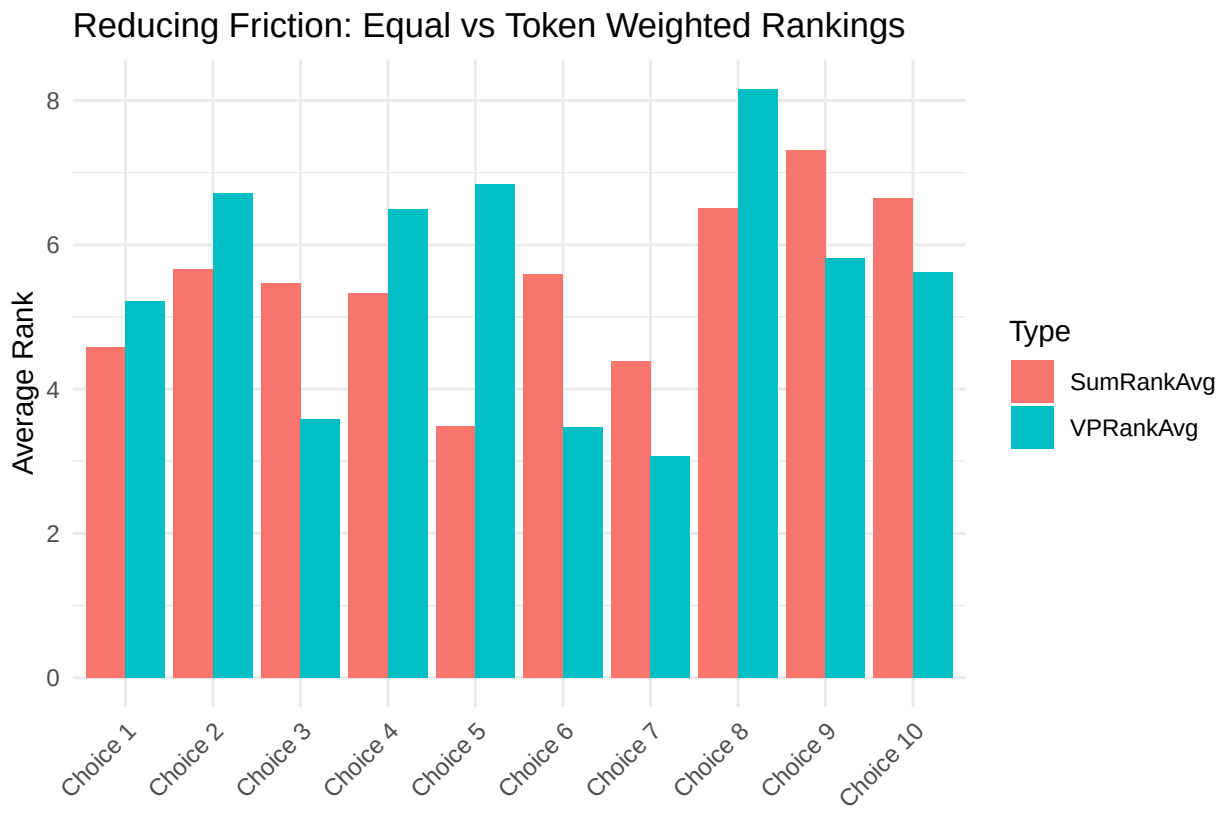
Visualizing Rankings

```
plot_ranking <- function(rankdf, title) {
  df <- rankdf %>%
    mutate(Statement = factor(Choice, labels = paste("Choice", 1:10))) %>%
    pivot_longer(cols = c(SumRankAvg, VPRankAvg), names_to = "Type", values_to = "Value")
  ggplot(df, aes(x = Statement, y = Value, fill = Type)) +
    geom_bar(stat = "identity", position = "dodge") +
    theme_minimal() +
    labs(title = title, y = "Average Rank", x = "") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
}

plot_ranking(rankraw14, "Growth & Innovation: Equal vs Token Weighted Rankings")
```




```
plot_ranking(rankraw58, "Reducing Friction: Equal vs Token Weighted Rankings")
```



OpenAI Analysis

In this section, we integrate advanced language modeling into our DAO voting analysis. While previous sections relied on deterministic statistical aggregation, here we leverage OpenAI's GPT-4o-mini model to synthesize a group consensus ranking from raw voting data. This approach introduces a qualitative perspective and allows us to compare traditional quantitative methods with the "collective intelligence" that an LLM might infer from observed voter preferences.

Extracting Choice Labels Programmatically

Before we can prompt OpenAI, we need to translate the numeric rankings from each voter back to their original statement text. The statement text for each ranked-choice poll is stored in the proposal body field, but not in a direct table. The following function automates extraction by parsing the proposal text for enumerated statements. This ensures our analysis is robust and future-proof, as it adapts to any poll with statements following a consistent numbering format.

```
# Extracts the choice labels (statements) from a Snapshot proposal's body text
extract_choice_labels <- function(proposal_body) {
  # Try to find the block starting with "\nThese are the statements:\n" or similar
  start_idx <- regexpr("These are the statements:", proposal_body)
  if (start_idx == -1) stop("Can't find statement block in proposal body")

  # Extract from that point to the end
  statements_block <- substring(proposal_body, start_idx)

  # Find all lines starting with '1.', '2.', ..., '10.' (robust for up to 10)
```

```
  lines <- unlist(strsplit(statements_block, "\n"))
  # Only keep lines starting with digit + dot
  choice_lines <- grep("^[0-9]+\\. ", lines, value = TRUE)
  # Remove "1. " etc
  labels <- sub("^[0-9]+\\.\\s*", "", choice_lines)
  return(labels)
}

# Example: for vote_df14 (growth & innovation)
proposal_id_14 <- unique(vote_df14$prop_id) # should be just one value
proposal_body_14 <- prop_df_sub$body[prop_df_sub$id == proposal_id_14]
choice_labels_14 <- extract_choice_labels(proposal_body_14)

# For vote_df58 (reducing friction)
proposal_id_58 <- unique(vote_df58$prop_id)
proposal_body_58 <- prop_df_sub$body[prop_df_sub$id == proposal_id_58]
choice_labels_58 <- extract_choice_labels(proposal_body_58)
```

Building a Consensus Ranking Prompt for OpenAI

To minimize costs and maximize efficiency when using the OpenAI API, we sample a subset of the available voter rankings. This random sampling reduces the number of tokens sent, while still capturing the main trends in voter sentiment. The following function packages this sample, constructs a precise API prompt, and defines exactly how we expect GPT-4o-mini to return results. Importantly, the prompt asks the model not just for a ranking, but also for a brief explanation of its methodology—providing valuable transparency into its "reasoning" process.

```
# Main function: Get consensus ranking using OpenAI
get_consensus_ranking <- function(vote_df,
                                  prop_df_sub,
                                  n_sample = 30,
                                  api_key_path = "your_openai_key.txt") {
  # a. Find proposal body for this vote_df
  proposal_id <- unique(vote_df$prop_id)
  proposal_body <- prop_df_sub$body[prop_df_sub$id == proposal_id]
  if (length(proposal_body) != 1) stop("Proposal body not found or ambiguous!")
  choice_labels <- extract_choice_labels(proposal_body)

  # b. Prepare numeric rankings (sample for token efficiency)
  numeric_rankings_text <- sapply(vote_df$choice, function(x) paste(x, collapse = ","))
  set.seed(42)
  sample_idx <- sample(seq_along(numeric_rankings_text),
                       min(n_sample, length(numeric_rankings_text)))
  sampled_rankings <- numeric_rankings_text[sample_idx]
  all_numeric_text <- paste(sampled_rankings, collapse = "\n")

  # Build minimal prompt for OpenAI
  prompt <- paste(
    "You are an expert at group consensus analysis.",
    "Below are multiple ranked lists of choices, each a permutation of the numbers 1-10,",
    "where each line is a different voter.",
    "Each number corresponds to a unique statement",
    "(but you do not need to know the statements).",
    "",
```

```
  "Each line is the ranked order for one voter (first is most preferred, last is least):",
  all_numeric_text,
  "",
  "Your tasks:",
  "1. Based only on these rankings, synthesize a single consensus ranking",
  "(as a permutation of 1-10) that best reflects the collective preference.",
  "2. Output ONLY the consensus ranking as your first line,",
  "in the following format (with no explanation before):",
  "Consensus: x1,x2,x3,x4,x5,x6,x7,x8,x9,x10",
  "3. After the ranking, in 2-4 sentences, explain *how* you synthesized this ranking:",
  "mention if you averaged positions, looked for patterns, considered polarization, etc.",
  "Focus on what you observed in the rankings, not the meaning of the numbers."
)



# Send to OpenAI API (requires API key in a text file)
openai_api_key <- readLines(api_key_path)
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", # or any preferred model
    messages = list(list(role = "user", content = prompt)),
    temperature = 0.2,
    max_tokens = 200
  ), auto_unbox = TRUE)
)

  # Parse response
result <- fromJSON(rawToChar(response$content))
content <- result$choices$message$content
lines <- unlist(strsplit(content, "\n"))
num_line <- grep("^Consensus:", lines, value = TRUE)[1]
if (is.na(num_line)) {
  cat("\\begin{verbatim}\n")
  cat("No consensus ranking found in the model output.\n")
  print(content)
  cat("\\end{verbatim}\n")
  return(NULL)
}
consensus_numeric <- gsub("[^0-9,]", "", num_line)
consensus_ids <- as.integer(unlist(strsplit(consensus_numeric, ",")))
consensus_statements <- choice_labels[consensus_ids]
explanation <- paste(lines[(which(lines == num_line)+1):length(lines)], collapse = "\n")

# FINAL OUTPUT BLOCK - SINGLE VERBATIM WRAP
cat("\\begin{verbatim}\n")
cat("Consensus Ranking (Numbers):\n", consensus_numeric, "\n\n")
cat("Consensus Ranking (Statements):\n")
for (i in seq_along(consensus_statements)) {
  cat(sprintf("Rank %d: Choice %d: %s\n", i, consensus_ids[i], consensus_statements[i]))
```

```
  }
  cat("\n---\n")
  cat("OpenAI Model Explanation:\n")
  wrapped_expl <- strwrap(explanation, width = 80)
  cat(wrapped_expl, sep = "\n")
  cat("\n")
  cat("\\end{verbatim}\n")

  return(consensus_statements)


}
```

Calling the GPT-4o-mini Consensus Function

Now, we call our custom function, passing in the appropriate vote and proposal data. The output will include both the ranked statements (mapped from the original choice indices) and the model's brief explanation of its ranking strategy.

By comparing these results to our deterministic, token-weighted rankings, we can highlight the differences between a purely statistical aggregation and the type of pattern recognition and consensus modeling performed by a state-of-the-art language model. This comparison adds another layer of interpretability to our study and offers a compelling way to triangulate DAO preferences.

Note: API costs can scale with prompt size, so be mindful of your sample size and your OpenAI account limits.

Interpretation

The GPT-4o-mini model's ranking is sometimes different from our classic deterministic aggregation. This is partly due to random sampling (to save tokens/cost) but also reflects the LLM's ability to weigh not only strict averages but also patterns in the distribution of ranks—such as consistency, outliers, and relative ordering trends. Its brief "reasoning" section can highlight patterns we might otherwise miss, making it a useful supplement to classic data science methods.

```
# -- For proposal 14 (Growth & Innovation) --
consensus_14 <- get_consensus_ranking(vote_df14,
                                      prop_df_sub,
                                      n_sample = 500,
                                      api_key_path = "../../openai_key.txt")
```

```
Consensus Ranking (Numbers):
 6,1,2,3,4,5,7,8,9,10

Consensus Ranking (Statements):
Rank 1: Choice 6: Prioritize Gas fee optimization.
Rank 2: Choice 1: Build a robust community of developers in ArbitrumDAO.
Rank 3: Choice 2: Make Arbitrum more accessible and easier to use for developers
Rank 4: Choice 3: Create an inclusive environment for our community.
Rank 5: Choice 4: Encourage meaningful participation in the DAO activities
Rank 6: Choice 5: Raise awareness about the opportunities on Arbitrum
Rank 7: Choice 7: Ensure regulatory compliance.
Rank 8: Choice 8: Build out anti-Sybil protections.
Rank 9: Choice 9: Uphold transparency in both project and governance development.
Rank 10: Choice 10: Improve the equity of token distribution.
```

```
---
OpenAI Model Explanation:
To synthesize the consensus ranking, I analyzed the individual rankings to
identify the most frequently preferred positions for each number. I averaged
the positions of each number across all voters, giving more weight to those
that appeared consistently in higher ranks. Additionally, I looked for patterns
in the rankings, noting that numbers like 6 and 1 frequently appeared at the
top, indicating a strong collective preference. This approach allowed me to
create a ranking that reflects the overall sentiment of the group while
minimizing the influence of outlier preferences.
```

```
# -- For proposal 58 (Reducing Friction) --
consensus_58 <- get_consensus_ranking(vote_df58,
                                      prop_df_sub,
                                      n_sample = 500,
                                      api_key_path = "../../openai_key.txt")
```

```
Consensus Ranking (Numbers):
 5,7,1,10,3,4,6,2,8,9

Consensus Ranking (Statements):
Rank 1: Choice 5: Improve gas fee optimization.
Rank 2: Choice 7: Incentivize users and builders to come to Arbitrum.
Rank 3: Choice 1: Develop accountability practices within ArbitrumDAO.
Rank 4: Choice 10: Evolve governance capabilities.
Rank 5: Choice 3: Form alliances with legacy institutions to boost our growth.
Rank 6: Choice 4: Fund projects to enhance cross-chain compatibility.
Rank 7: Choice 6: Define growth strategies.
Rank 8: Choice 2: Identify the key capabilities that will lead to improved DAO performance.
Rank 9: Choice 8: Scale of the platform.
Rank 10: Choice 9: Offer educational opportunities to build the community.

---
OpenAI Model Explanation:
To synthesize the consensus ranking, I analyzed the individual rankings by
calculating the average position of each number across all voters. I observed
that certain numbers consistently appeared in the top positions, particularly
5, 7, and 1, indicating a strong collective preference for these choices.
Additionally, I noted that numbers like 10 and 9 tended to be ranked lower,
suggesting polarization in preferences. The final consensus reflects a balance
of these observations, prioritizing the most frequently favored options while
considering their relative positions across the rankings.
```

Conclusions

This analysis provides a comprehensive framework for quantifying and understanding alignment between large token holders and the broader Arbitrum DAO community with respect to strategic priorities. By examining both equal-weighted (one wallet, one vote) and token-weighted (voting power) rankings, we are able to reveal not only the overall consensus but also to highlight potential divergences between high-stake holders and general participants.

Through classic statistical aggregation, we demonstrated which statements consistently ranked at the top

across different voting schemes, thereby identifying areas of strong community consensus. Conversely, discrepancies between the two schemes can help surface priorities that are particularly important to either major stakeholders or the broader base, informing more inclusive decision-making processes.

In addition to these quantitative approaches, we integrated OpenAI's state-of-the-art language model (GPT-4o-mini) into our workflow. By prompting the AI with real anonymized voting records, we obtained an AI-synthesized consensus ranking and a concise explanation of the logic behind the ordering. This adds a qualitative layer of analysis that can capture subtle patterns—such as clustering, polarization, or emergent trends in ranking behavior—that might be overlooked by traditional methods. The AI explanation also enhances transparency and interpretability, offering an alternative lens for both researchers and governance participants.

Our methodology is generalizable and can be readily applied to other DAO governance polls and ranking-based decision scenarios. By blending statistical analysis with AI-based synthesis, we offer a richer, multidimensional perspective on community preferences—empowering DAOs to make more informed, balanced, and democratic decisions.

In summary, combining traditional ranking analytics with modern AI tools not only validates results through multiple lenses but also brings greater transparency and insight into complex governance ecosystems like Arbitrum DAO.