# Supplementary Information

## Supplementary Information

Paper: *The late Quaternary megafauna extinction debate: a systematic review of the literature*

### Bibliographic Analyses

In order to improve our understanding of the structure of the megafauna extinction debate across disciplines, we conducted a series of quantitative analyses involving both citation data (with bibliometrics) and labels/tags derived from the literature review. Using bibliographic data, we constructed a citation network, modelling each paper as a node. The nodes were connected if one paper (node) cited another. The structure of this citation network, we argued, could be used to better understand how the debate(s) about megafauna extinction have unfolded, whether the conversation among experts was highly siloed (into academic "camps"), and whether idea lineages could be traced. We also used standard cluster detection methods to identify clusters of papers and then examined correlations between the citation clusters and tagged themes to see whether prominent ideas in those papers corresponded to the identified network clusters — e.g., whether the extinction cause promoted in a given paper predicted its membership in an empirically identifiable cluster. Lastly, we created a co-authorship network where each node represented a scholar identified by last name in the bibliographic sample. Authors (nodes) in the co-authorship network shared an edge if the relevant authors ever co-authored a paper together (appeared together on the authorship list of a given paper in our sample). We used this network to identify prominent (highly connected) authors and to see whether co-authorship clustering corresponded to clusters in the citation network — which would, we reasoned, indicate scientific "camps" in the broader literature that tended to cite themselves and their co-authors, contributing to clumpy structure in the citation network.

- Citation Network: papers are nodes, edges indicate citations
- Co-Authorship Network: authors are nodes, edges indicate coauthorship

## Replication

The rest of this notebook can be used to fully replicate our analysis in a Python environment.

### Load Libraries

```
# basic data science
import pandas as pd
import numpy as np
from scipy.stats import spearmanr
import hashlib
import random
import re

# network tools
import networkx as nx
from networkx.algorithms.dag import dag_longest_path
from networkx.algorithms.dag import dag_longest_path_length
import igraph as ig
import leidenalg

# plotting
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from networkx.drawing.nx_agraph import graphviz_layout
from matplotlib.cm import get_cmap
import arviz as az

# PYMC and Modelling
import pymc as pm
import pytensor.tensor as pt
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from pytensor.tensor.special import softmax

# collections and combinations
from collections import Counter
from itertools import combinations

# hotfix deprecation warnings
import warnings
from matplotlib import MatplotlibDeprecationWarning
```

```
warnings.filterwarnings("ignore", category=MatplotlibDeprecationWarning)

# misc
import tabulate
```

### Load Bibliometric Data

We begin by loading the bibliometric and review data from a spreadsheet in order to create a bibliometric dataframe for further analyses. We will use the DOI of each paper as a unique identifier.

```
# Load the original bibliometric spreadsheet
df_original = pd.read_excel("../data/Papers_list_final_V2.xlsx")

# Load the new spreadsheet with disciplinary and extinction tags
df_new_tags = pd.read_excel("../data/mf_review_papers_final.xlsx")

# Check for consistent column name for DOI (adjust if needed)
doi_col_original = 'DOI'
doi_col_new = 'DOI'

# Standardize the DOI format (e.g., strip, lowercase)
df_original[doi_col_original] = (
    df_original[doi_col_original]
    .str.strip()
    .str.lower()
)

df_new_tags[doi_col_new] = (
    df_new_tags[doi_col_new]
    .str.strip()
    .str.lower()
)


# Merge on DOI
df_merged = df_original.merge(
    df_new_tags,
    left_on=doi_col_original,
    right_on=doi_col_new,
    how='left',  # keep all original rows
```

```
    suffixes=('', '_new')  # avoid column name clashes
)

# Set index for later use
df_indexed = df_merged.set_index('DOI')
```

Some DOIs are contained in a list in a single spreadsheet cell, which means they come into the Python environment as a text string that needs to be parsed.

```
# Regex to match DOIs (robust version)
doi_regex = re.compile(r'(10\.\d{4,9}/[-._;()/:A-Z0-9]+)', re.IGNORECASE)

def extract_dois(cited_block):
    if pd.isna(cited_block):
        return []

    # Split by semicolon to get individual references
    refs = cited_block.split(';')
    dois = []

    for ref in refs:
        found = doi_regex.findall(ref)
        for doi in found:
            clean = doi.strip().rstrip('.,;)]').lstrip('[')
            if clean not in dois:
                dois.append(clean)

    return dois

# Apply to dataframe
df_indexed['Cited_DOIs'] = df_indexed['Cited_references'].apply(extract_dois)
```

### Create Graphs

The next step is to use standard tools in Python (from the NetworkX library) to construct network data structures based on the references (and DOIs). Since the bibliographic data include citations to papers not in the review sample, there could be tens of thousands of nodes, ultimately, in a citation graph, but we have only review data (labels, themes, and so on) for the papers in our specific sample. As a result, we will limit the networks to just the papers inside the sample we reviewed.

```
# Create a directed graph
G = nx.DiGraph()

# Add nodes and edges
for _, row in df_indexed.iterrows():
    citing_doi = row.name  # DOI is now the index
    cited_dois = row['Cited_DOIs']

    if not citing_doi or pd.isna(citing_doi):
        continue

    G.add_node(citing_doi)  # Add the focal paper

    for cited_doi in cited_dois:
        if cited_doi and isinstance(cited_doi, str):
            G.add_node(cited_doi)
            G.add_edge(citing_doi, cited_doi)

# Internal DOIs are now the index
internal_dois = set(df_indexed.index)

# Keep only edges where both source and target are internal
internal_edges = [
    (row.name, cited)
    for _, row in df_indexed.iterrows()
    for cited in row['Cited_DOIs']
    if row.name in internal_dois and cited in internal_dois
]

G_internal = nx.DiGraph()
G_internal.add_edges_from(internal_edges)

print(
    f"Internal network: {G_internal.number_of_nodes()} nodes, "
    f"{G_internal.number_of_edges()} edges"
)
```

```
Internal network: 328 nodes, 1816 edges
```

```
print(f"Total nodes: {G.number_of_nodes()}")
print(f"Total edges: {G.number_of_edges()}")
```

```
# Most cited papers (highest in-degree)
top_cited = sorted(G.in_degree(), key=lambda x: x[1], reverse=True)[:10]
print("Top cited DOIs:")
for doi, count in top_cited:
    print(f"{doi} : {count} citations")
```

```
Total nodes: 11472
Total edges: 20217
Top cited DOIs:
10.1146/annurev.ecolsys.34.011802.132415 : 117 citations
10.1126/science.1101476 : 112 citations
10.1038/nature10574 : 62 citations
10.1098/rspb.2013.3254 : 56 citations
10.1126/science.1059342 : 55 citations
10.1126/science.1060264 : 54 citations
10.1016/j.quaint.2009.11.017 : 50 citations
10.1126/science.1179504 : 47 citations
10.1073/pnas.1502540113 : 43 citations
10.1016/S0305-4403(02)00205-4 : 43 citations
```

Using *in-degree* (the number of edges coming into a node), we can approximate the citation count for a given paper and then compare that to the counts in other databases, like the Web of Science. If we find strong correlations, then we can assume the relative prominence of the papers in our sample is well represented by in-degree.

```
# Compute in-degree (citations received from others in the dataset)
in_degree = dict(G.in_degree())
df_indexed['In_degree'] = df_indexed.index.map(in_degree).fillna(0).astype(int)
```

```
comparison_df = df_indexed[
    ['In_degree', 'Times_cited_WoS',
     'Times_cited_all_database']
]

# Ensure types are numeric
comparison_df = comparison_df.apply(pd.to_numeric, errors='coerce').fillna(0)

# Shorten column names
renamed = {
    'In_degree': 'InDeg',
    'Times_cited_WoS': 'WoS',
```

```
    'Times_cited_all_database': 'AllDB'
}

df_short = comparison_df.rename(columns=renamed)
correlation_matrix = df_short.corr().round(2)
```

```
correlation_matrix
```

Table 1: Pearson correlation matrix for citation metrics

| | InDeg | WoS | AllDB |
|---|---|---|---|
| InDeg | 1.00 | 0.78 | 0.79 |
| WoS | 0.78 | 1.00 | 1.00 |
| AllDB | 0.79 | 1.00 | 1.00 |

We found strong correlations (see table Table 1 above) across citation counts for the papers in our sample, which means we can assume our internal network metrics reflect (in relative terms) the wider prominence of the papers within the overall literature.

```
df_indexed['Parsed_reference_count'] = df_indexed['Cited_DOIs'].apply(
    lambda x: len(x) if isinstance(x, list) else 0
)
```

::: {#tbl-reference-summary .cell tbl-cap=' "Summary statistics for reference counts: ParsedRefs = Number of parsed references, CitedRefs = Number of cited references." ' execution_count=10}

```
# Rename and round
comparison = (
    df_indexed[['Parsed_reference_count', 'Cited_reference_count']]
    .rename(columns={
        'Parsed_reference_count': 'ParsedRefs',
        'Cited_reference_count': 'CitedRefs'
    })
)


comparison = comparison.apply(pd.to_numeric, errors='coerce').fillna(0)

summary_table = comparison.describe().round(0).astype(int)
summary_table
```

| | ParsedRefs | CitedRefs |
|---|---|---|
| count | 360 | 360 |
| mean | 56 | 81 |
| std | 38 | 51 |
| min | 0 | 0 |
| 25% | 33 | 51 |
| 50% | 51 | 73 |
| 75% | 71 | 101 |
| max | 335 | 499 |

:::

```
correlation = comparison.corr().round(2)
correlation
```

| | ParsedRefs | CitedRefs |
|---|---|---|
| ParsedRefs | 1.00 | 0.88 |
| CitedRefs | 0.88 | 1.00 |

### Analyze Graphs

We conducted a number of analyses of the citation network. First, we looked at the number of strongly and weakly connected components, which are used to describe the network's connectivity structure. A strongly connected component is a maximal subgraph in which every node can reach every other node via directed paths, while a weakly connected component is a maximal subgraph where all nodes are connected if edge directions are ignored. Maximal, in this context, means that no additional nodes can be included in the relevant subgraph without breaking the strongly or weakly connected property — so, adding another node from the total graph to a weakly connected subgraph would mean that not every node in the subgraph could reach the new node (ignoring edge direction).

*The number of strongly connected components (SCC) indicates how many maximal subgraphs exist in which every node is reachable from every other node via directed paths.*

*The number of weakly connected components (WCC) shows how many maximal subgraphs remain connected if edge directions are ignored.*

```
num_scc = nx.number_strongly_connected_components(G)
num_wcc = nx.number_weakly_connected_components(G)

num_scc_internal = nx.number_strongly_connected_components(G_internal)
num_wcc_internal = nx.number_weakly_connected_components(G_internal)

print(f"Strongly connected components (total graph): {num_scc}")
print(f"Weakly connected components (total graph): {num_wcc}")

print(f"Strongly connected components (internal graph): {num_scc_internal}")
print(f"Weakly connected components (internal graph): {num_wcc_internal}")
```

```
Strongly connected components (total graph): 11472
Weakly connected components (total graph): 10
Strongly connected components (internal graph): 328
Weakly connected components (internal graph): 1
```

The algorithms used to detect strongly and weakly connected components revealed distinct structures in the total and internal citation graphs (see above). In the total graph (which includes all incoming citations from papers that cite any paper in our review sample), we find 11,472 strongly connected components and 10 weakly connected components. In the internal graph (which considers only papers and citation edges within our specific sample), we find 328 strongly connected components but a single weakly connected component.

In citation networks like the ones analyzed here, strongly connected components typically reduce to individual papers, since reciprocal citations are rare — papers are published in sequence, not in parallel, and thus cannot usually cite each other simultaneously. As a result, each strongly connected component often corresponds to a single node (a given paper with directed edges pointing backward in time to the papers it cites).

In contrast, the presence of a single weakly connected component in the internal graph indicates that all papers are connected through at least indirect citation chains when edge direction is ignored. This effectively means one can trace a path from any paper to any other in the sample, suggesting that many papers share common "ancestors" in terms of cited works. We interpret this as indicating that the papers in our sample represent a single, well-connected scientific conversation, with little evidence of true silos.

```
is_dag = nx.is_directed_acyclic_graph(G_internal)
print(f"Is DAG: {is_dag}")
```

```
Is DAG: True
```

We also find, not surprisingly then, that the graph is generally directed and acyclic — a Directed Acyclic Graph (DAG) is one where the nodes have directed edges and there are no (or perhaps very few) loops (where, for instance, a paper cites itself or a child paper cites a parent paper which then cites the child paper in a loop).

```
if is_dag:
    longest = max(
        (
            nx.dag_longest_path_length(G_internal.subgraph(c))
            for c in nx.weakly_connected_components(G_internal)
        ),
        default=0
    )
    print(f"Longest citation chain: {longest} steps")
```

```
Longest citation chain: 21 steps
```

We also found that the longest lineage in the citation network involved 21 steps (edges). And next, we can look at some general metrics, like number of papers (to confirm that the whole sample was ingested in Python), number of unique DOIs, and so on:

```
num_internal_papers = df_indexed.index.nunique()
print(f"Papers in dataset (internal): {num_internal_papers}")
# Count total extracted citations across all papers
total_extracted_dois = df_indexed['Cited_DOIs'].apply(len).sum()
print(f"Total cited DOIs extracted: {total_extracted_dois}")
# Count unique cited DOIs
all_cited_dois = set(
    doi
    for sublist in df_indexed['Cited_DOIs']
    for doi in sublist
)
print(f"Unique cited DOIs: {len(all_cited_dois)}")
# Nodes that are both cited and in dataset
internal_dois = set(df_indexed.index)
shared_dois = internal_dois & all_cited_dois
external_only = all_cited_dois - internal_dois

print(f"Internal DOIs also cited: {len(shared_dois)}")
print(f"External-only DOIs: {len(external_only)}")
```

```
Papers in dataset (internal): 360
Total cited DOIs extracted: 20217
Unique cited DOIs: 11360
Internal DOIs also cited: 248
External-only DOIs: 11112
```

We then counted the number of edges (citation connections) and the number that appear to be forward in time (which in general should be impossible because it would indicate a paper citing a paper that was published after it, which could happen occasionally but should be very rare). A forward-in-time citation could occur if, for instance, a preprint was cited or an 'online first' article with a later publication date was cited by a paper.

```
# Build a lookup from DOI to publication year
doi_to_year = dict(zip(df_indexed.index, df_indexed['Publication_year']))

# Identify any edges that violate citation chronology
forward_edges = [
    (citing, cited)
    for citing, cited in G_internal.edges
    if doi_to_year.get(citing, float('inf')) <
       doi_to_year.get(cited, float('-inf'))
]

print(f"Total edges: {G_internal.number_of_edges()}")
print(f"Forward-in-time edges: {len(forward_edges)}")
```

```
Total edges: 1816
Forward-in-time edges: 1
```

In the next few cells, we built a plot intended to show the citation network along a time-axis.

```
# Build node positions with time on X and jittered Y
pos = {}
for node in G_internal.nodes:
    year = doi_to_year.get(node, None)
    if year is not None:
        x = year
        y = np.random.normal(0, 1)
        pos[node] = (x, y)
```

```
# Extract years from positions
years = [pos[n][0] for n in G_internal.nodes if n in pos]
min_year = int(min(years))
max_year = int(max(years))

# Generate reasonable tick range
tick_interval = 2 if max_year - min_year < 20 else 5
xticks = list(range(min_year, max_year + 1, tick_interval))
```

```
def hash_float(s, scale=1.0):
    return (int(hashlib.sha1(s.encode()).hexdigest(), 16) % 1000) / 1000 * scale

pos = {}
for node in G_internal.nodes:
    year = doi_to_year.get(node)
    if year is not None:
        x = year
        y = hash_float(node, scale=3.0)  # Stable but scattered
        pos[node] = (x, y)
```

### Citation Network Clustering

To see structure in the citation network more clearly, we then used the Leiden network clustering algorithm to automatically determine the number and composition of likely clusters. The Leiden algorithm is a standard community detection method that partitions a network into clusters (i.e., communities) by maximizing a function known as modularity. It improves upon earlier approaches (like the Louvain algorithm) by ensuring that clusters are well-connected internally and more stable, avoiding disconnected or badly connected groups.

For our purposes, the algorithm helps reveal coherent subgroups of papers that are more densely interlinked by citations. We can then assign colours to the nodes based on their cluster memberships and structure the network-over-time plot to clearly show the identified structures.

```
# Convert NetworkX graph to igraph
G_undirected = G_internal.to_undirected()
nx_nodes = list(G_undirected.nodes())
nx_edges = list(G_undirected.edges())

g = ig.Graph()
g.add_vertices(nx_nodes)
g.add_edges([(nx_nodes.index(u), nx_nodes.index(v)) for u, v in nx_edges])
```

```
# Run Leiden community detection
partition = leidenalg.find_partition(g, leidenalg.ModularityVertexPartition)

# Map communities back to original node labels
leiden_clusters = {
    nx_nodes[i]: cluster_id
    for i, cluster_id in enumerate(partition.membership)
}

# Number of communities
num_communities = len(set(leiden_clusters.values()))
print(f"Detected communities (Leiden): {num_communities}")
```

```
Detected communities (Leiden): 7
```

```
df_indexed['Leiden_cluster'] = df_indexed.index.map(leiden_clusters)
```

```
nodes = list(G_internal.nodes)
cluster_ids = df_indexed.loc[nodes, 'Leiden_cluster']
unique_clusters = sorted(cluster_ids.dropna().unique())
num_clusters = len(unique_clusters)
cluster_to_band = {cid: i for i, cid in enumerate(unique_clusters)}

cmap = get_cmap('tab10', num_clusters)
node_colors = [cmap(cluster_to_band.get(cluster_ids.loc[n], 0)) for n in nodes]

band_height = 1.0
jitter = 0.3
pos_clustered = {}
for n in nodes:
    x_orig = pos[n][0]
    y0 = cluster_to_band.get(cluster_ids.loc[n], 0) * band_height
    y = y0 + random.uniform(-jitter, jitter)
    pos_clustered[n] = (x_orig, y)

all_years = [int(pos[n][0]) for n in nodes]
min_year, max_year = min(all_years), max(all_years)
tick_interval = 5
x_ticks = list(range(min_year, max_year + 1, tick_interval))

plt.figure(figsize=(16, 6))
```

```
nx.draw_networkx_nodes(
    G_internal, pos_clustered,
    node_size=50, node_color=node_colors,
    alpha=0.9, hide_ticks=False
)
nx.draw_networkx_edges(
    G_internal, pos_clustered,
    edge_color='gray', alpha=0.1,
    arrows=False, hide_ticks=False
)

# Re-enable axis and set ticks/labels
ax = plt.gca()
ax.set_axis_on()
ax.set_xticks(x_ticks)
ax.set_xticklabels(x_ticks, rotation=45)
y_locs = [i * band_height for i in range(num_clusters)]
ax.set_yticks(y_locs)
ax.set_yticklabels([str(cid) for cid in unique_clusters])

plt.title("Citation Network (Leiden Clusters with Jitter & Proper Axis)")
plt.xlabel("Publication Year")
plt.ylabel("Leiden Cluster ID")
plt.xlim(min_year - 1, max_year + 1)
plt.tight_layout()

# Save the figure
plt.savefig("cite_net_time.svg", format='svg')
plt.savefig("cite_net_time.png", format='png', dpi=300)

plt.show()
```

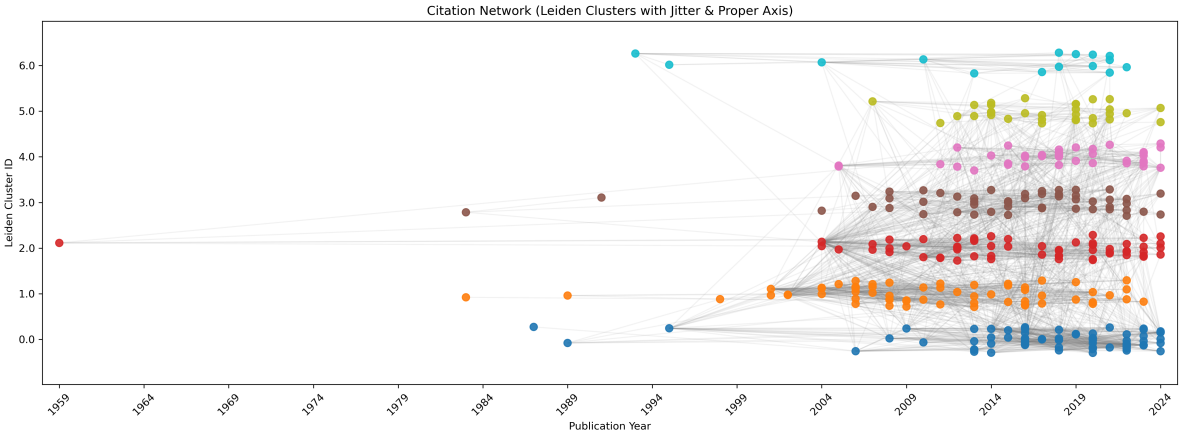



Figure 1: Citation Network (Leiden Clusters with Jitter & Proper Axis)

As Figure Figure 1 above shows, some of the identified clusters have significant time depth while others are more recent, and some involve a lot of papers while others are sparse. It is important to note that these clusters are not fully deterministic even though Leiden is highly stable. We re-ran the last few code cells a large number of times, and readers are welcome to repeat this process and see for themselves that the results are very stable.

### Explaining Clustering

The next section of this notebook includes analyses intended to better understand the identified structure (clusters) in the citation network. We focus on a selection of variables that might elucidate the academic conversation about megafauna extinctions and help to explain why clusters appear to exist in the network. We explore:

1. **Disciplinary Effects** (are clusters predicted by the disciplines associated with the relevant papers — e.g., papers in archaeology-focused journals are more likely to cite other archaeological papers, giving rise to clustering);
2. **Extinction Cause** (are clusters predicted by the main extinction causes promoted by the relevant papers);
3. **Coauthorship** (are clusters explained by co-authorship effects whereby authors cite close colleagues, making citation network clusters reflect authorship clusters)

First we plot the fraction of each citation cluster made up of papers tagged with a given discipline (Figure Figure 2 below based on the Web of Science categorization):

```
# Count raw number of papers by cluster and discipline
discipline_counts = (
    df_indexed
```

```
    .groupby(['Leiden_cluster', 'Categories'])
    .size()
    .unstack(fill_value=0)
)

# Normalize by total per cluster (row-wise)
discipline_proportions = discipline_counts.div(
    discipline_counts.sum(axis=1),
    axis=0
)

# Plot as stacked bar chart of proportions
ax = discipline_proportions.plot(
    kind='bar',
    stacked=True,
    figsize=(12, 6),
    colormap='tab20'
)

plt.title('Relative Disciplinary Composition of Leiden Clusters')
plt.xlabel('Leiden Cluster')
plt.ylabel('Proportion of Papers')
plt.xticks(rotation=0)
plt.legend(title='Discipline', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

# Save the figure
plt.savefig("leiden_disciplinary_composition.svg", format='svg')
plt.savefig("leiden_disciplinary_composition.png", format='png', dpi=300)

plt.show()
```

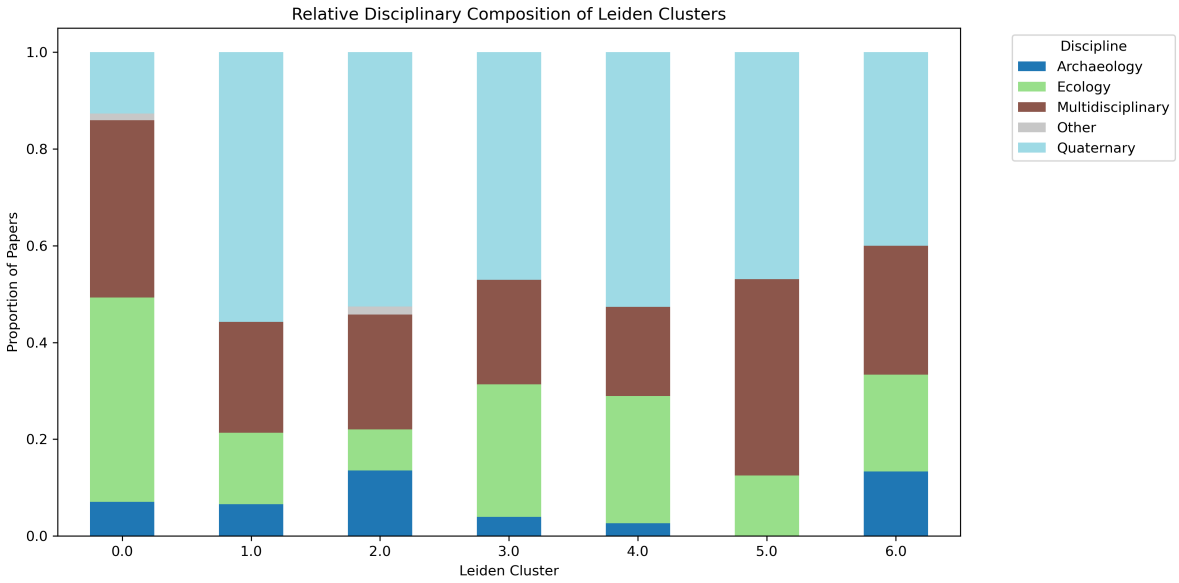



Figure 2: Disciplinary Composition of Clusters

Next, we produce a similar plot, but looking this time at the prorportion of papers in each cluster (Figure Figure 3 below) that promote a given extinction cause — determined by our review:

```
# Create a contingency table: Leiden cluster vs. extinction cause
extinction_counts = pd.crosstab(df_indexed['Leiden_cluster'],
                                df_indexed['Extinction Cause'])

# Normalize row-wise to get proportions
extinction_proportions = extinction_counts.div(extinction_counts.sum(axis=1),
                                               axis=0)

# Plot the stacked bar chart
ax = extinction_proportions.plot(
    kind='bar',
    stacked=True,
    figsize=(12, 6),
    colormap='tab20'
)

plt.title("Relative Extinction Argument Composition per Leiden Cluster")
plt.xlabel("Leiden Cluster")
plt.ylabel("Proportion of Papers")
```

```
plt.xticks(rotation=0)
plt.legend(title='Extinction Cause', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()

# Save the figure
plt.savefig("leiden_extinction_composition.svg", format='svg')
plt.savefig("leiden_extinction_composition.png", format='png', dpi=300)

plt.show()
```

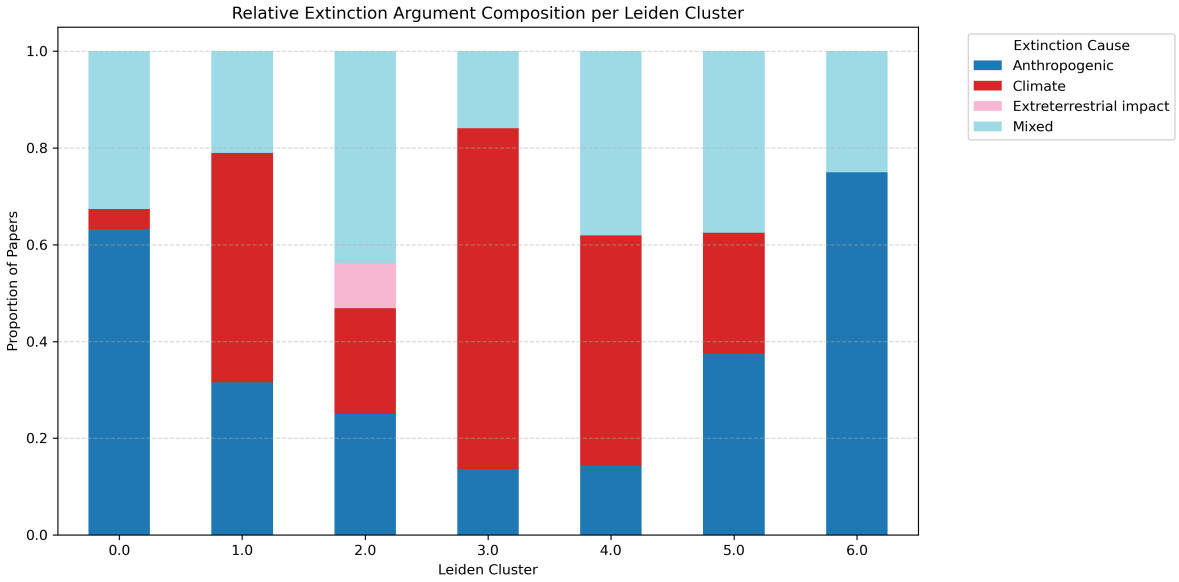



Figure 3: Extinction argumeent composition of clusters

### Co-Authorship

The last variable we considered, co-authorship networks, was the most complicated to develop. Over the next few cells of code, we extract the author names from the review sample bibliographic data and then construct a network where edges reflect two authors (the nodes) having been listed as co-authors on at least one paper.

```
# Create co-authorship graph
G_coauthor = nx.Graph()

for authors in df_indexed['Authors'].dropna():
```

```
    author_list = [a.strip() for a in authors.split(';') if a.strip()]
    for a1, a2 in combinations(author_list, 2):
        if G_coauthor.has_edge(a1, a2):
            G_coauthor[a1][a2]['weight'] += 1
        else:
            G_coauthor.add_edge(a1, a2, weight=1)
```

```
# Compute node degrees
degrees = dict(G_coauthor.degree())
max_degree = max(degrees.values())

# Node sizes scaled (optional: tweak the multiplier)
node_sizes = [100 + 900 * (degrees[n] / max_degree) for n in G_coauthor.nodes]

# Spring layout with edge weights as "gravity"
pos = nx.spring_layout(G_coauthor,
                       k=0.5,
                       iterations=100,
                       weight='weight',
                       seed=42)

# Plotting
plt.figure(figsize=(16, 12))
nx.draw_networkx_nodes(G_coauthor, pos, node_size=node_sizes, alpha=0.8)
nx.draw_networkx_edges(G_coauthor, pos, alpha=0.1)

plt.title(
    "Co-authorship Network "
    "(Node Size = Degree, Gravity = Co-authorship Frequency)"
)
plt.axis('off')
plt.tight_layout()

# Save the figure
plt.savefig("coauth_net.svg", format='svg')
plt.savefig("coauth_net.png", format='png', dpi=300)

plt.show()
```

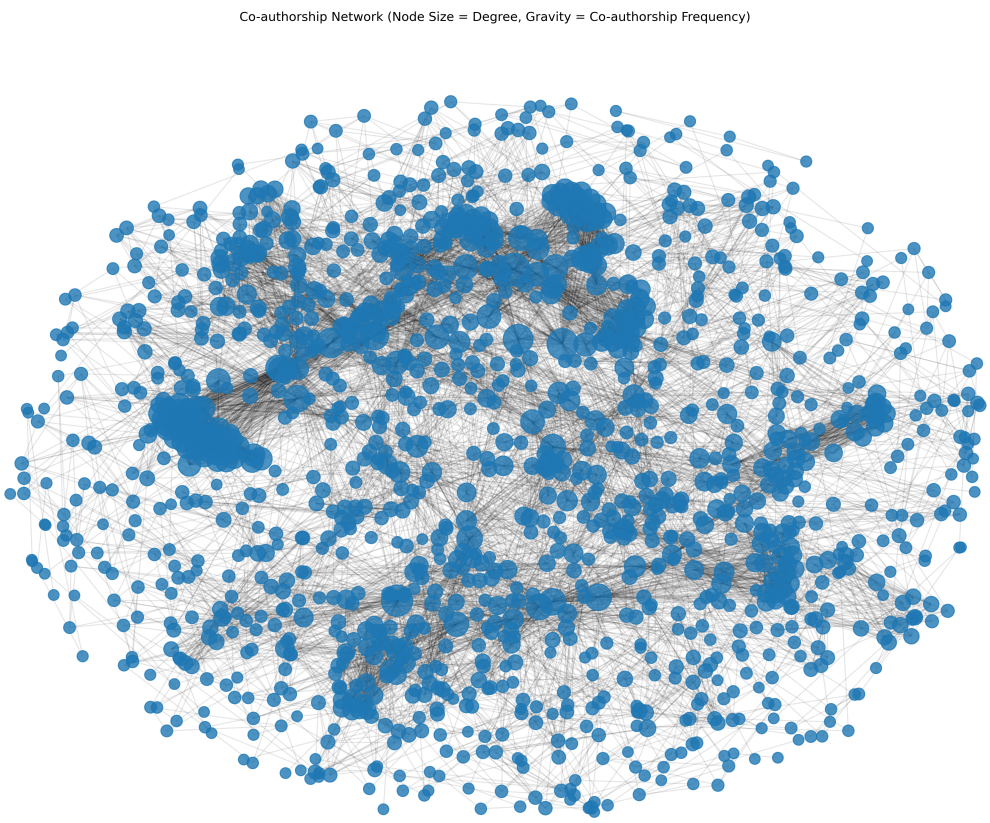



Figure 4: Coauthorship network

The network plot above (Figure @coauth-net) shows a large, complex structure typical of human social networks. It exhibits "small-world" dynamics, meaning it is well-connected overall, without isolated groups, while still containing distinct clusters of authors who form tight-knit communities. Importantly, a few authors play a dominant role in linking the network together, whereas the majority of authors have relatively few connections, sometimes only a single co-authorship link to the broader network.

Next, we used the Leiden algorithm again to detect clusters in the co-author network. The results indicate a large number of potential communities or clusters, many more than could meaningfully account for the 6 clusters we identified in the citation network. In part, this is simply because there are a large number of smaller potential communities. But, as is obvious from the plot of the co-authorship network, just a few communities are highly dominant. So, in order to extract usable insights and limit the number of potential explanatory variables (for explaining/predicting structure in the citation network), we extracted information about the proportion of edges accounted for by each co-authorship cluster. This allowed us to determine

whether a few clusters were hyper-dominant in the graph and, thus, we could justify limiting our further exploration to just those ones.

```
# Convert to igraph
G_ig = ig.Graph.TupleList(G_coauthor.edges(), directed=False)

# Run Leiden
partition = leidenalg.find_partition(G_ig, leidenalg.ModularityVertexPartition)

# Map author → cluster
author_clusters = {
    G_ig.vs[v]['name']: cid
    for cid, cluster in enumerate(partition)
    for v in cluster
}
```

```
def paper_author_clusters(authors_str):
    if pd.isna(authors_str): return []
    authors = [a.strip() for a in authors_str.split(';')]
    return list({author_clusters.get(a) for a in authors if a in author_clusters})

df_indexed['Author_clusters'] = df_indexed['Authors'].apply(paper_author_clusters)
```

We counted the number of co-authorship connections accounted for by each cluster and then ordered those from highest to lowest. From those sorted data, we calculated a cumulative count and proportion sequence, which can be read a bit like cumulative variance accounted for in a Principal Components context — i.e., if a few clusters are hyper-dominant, we expect a rapid increase in cumulative edge counts and the proportion of total edges, which is exactly what we found (see the dataframe printed below followed by a cumulative plot).

```
# For each edge, get the author cluster IDs
edge_cluster_pairs = []

for a1, a2 in G_coauthor.edges:
    c1 = author_clusters.get(a1)
    c2 = author_clusters.get(a2)
    if c1 is not None and c1 == c2:  # Only count within-cluster edges
        edge_cluster_pairs.append(c1)

# Count how many edges fall into each cluster
cluster_edge_counts = Counter(edge_cluster_pairs)
```

```
# Convert to dataframe and sort
import pandas as pd

edge_df = pd.DataFrame.from_dict(cluster_edge_counts,
                                 orient='index',
                                 columns=['Edge_Count'])
edge_df.index.name = 'Author_Cluster'
edge_df = edge_df.sort_values('Edge_Count', ascending=False)
edge_df['Cumulative'] = edge_df['Edge_Count'].cumsum()
edge_df['Proportion'] = edge_df['Edge_Count'] / edge_df['Edge_Count'].sum()
edge_df['Cumulative_Proportion'] = edge_df['Proportion'].cumsum()

edge_df.head(10)
```

| Author_Cluster | Edge_Count | Cumulative | Proportion | Cumulative_Proportion |
|---|---|---|---|---|
| 3 | 1595 | 1595 | 0.170606 | 0.170606 |
| 8 | 1247 | 2842 | 0.133383 | 0.303990 |
| 0 | 945 | 3787 | 0.101080 | 0.405070 |
| 1 | 883 | 4670 | 0.094449 | 0.499519 |
| 2 | 876 | 5546 | 0.093700 | 0.593219 |
| 4 | 601 | 6147 | 0.064285 | 0.657503 |
| 5 | 394 | 6541 | 0.042144 | 0.699647 |
| 6 | 354 | 6895 | 0.037865 | 0.737512 |
| 12 | 348 | 7243 | 0.037223 | 0.774735 |
| 7 | 330 | 7573 | 0.035298 | 0.810033 |

```
plt.figure(figsize=(10, 5))
plt.plot(range(1, len(edge_df)+1), edge_df['Cumulative_Proportion'], marker='o')
plt.xlabel("Author Clusters (sorted by size)")
plt.ylabel("Cumulative Proportion of Intra-cluster Co-authorship Edges")
plt.title("Dominance of Author Clusters in Co-authorship Network")
plt.grid(True)
plt.tight_layout()

# Save the figure
plt.savefig("coauth_cluster_dominance.svg", format='svg')
plt.savefig("coauth_cluster_dominance.png", format='png', dpi=300)

plt.show()
```

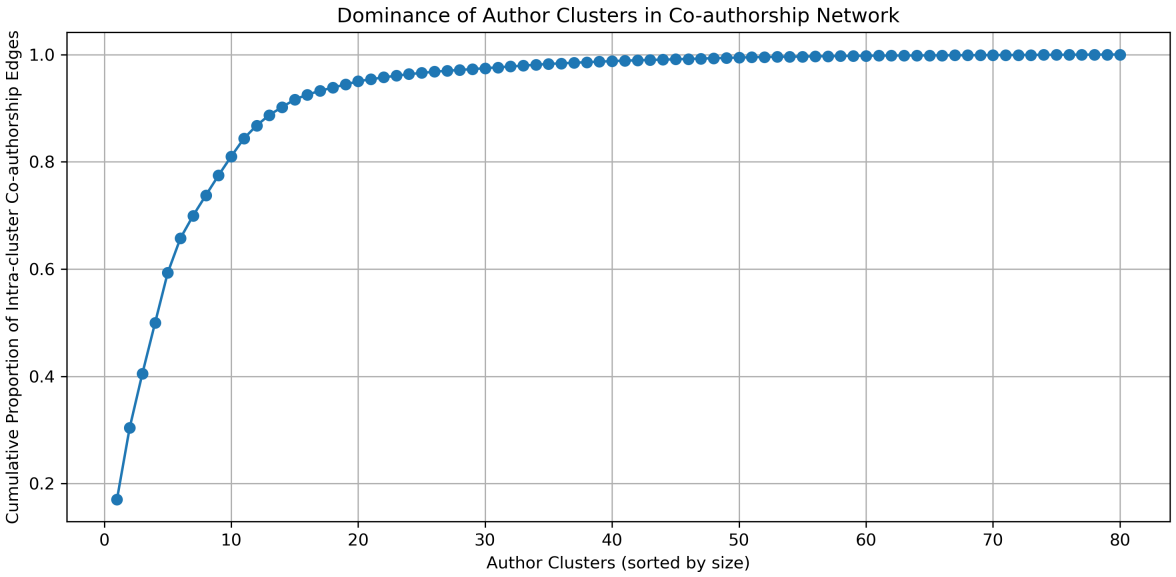



Figure 5: Dominance of clusters within co-authorship network

According to the plot above (Figure Figure 5), we can see that the top 10 clusters by edge count account for 80% of the total edges in the network. In contrast, the remaining 70 clusters identified by the Leiden algorithm account for only 20%.

Next, identify the co-author clusters that occur most frequently across all papers in the review sample. These are the co-authorship clusters most responsible for the literature (and ideas) in the review sample.

```
# Count all author cluster occurrences
all_clusters_flat = [
    c
    for row in df_indexed['Author_clusters']
    if isinstance(row, list)
    for c in row
]

cluster_counts = Counter(all_clusters_flat)

# Identify top 9 clusters by frequency
top_n = 9
top_clusters = {c for c, _ in cluster_counts.most_common(top_n)}

# reduce clusters by binning low-count clusters into 'other'
def reduce_clusters(cluster_list):
```

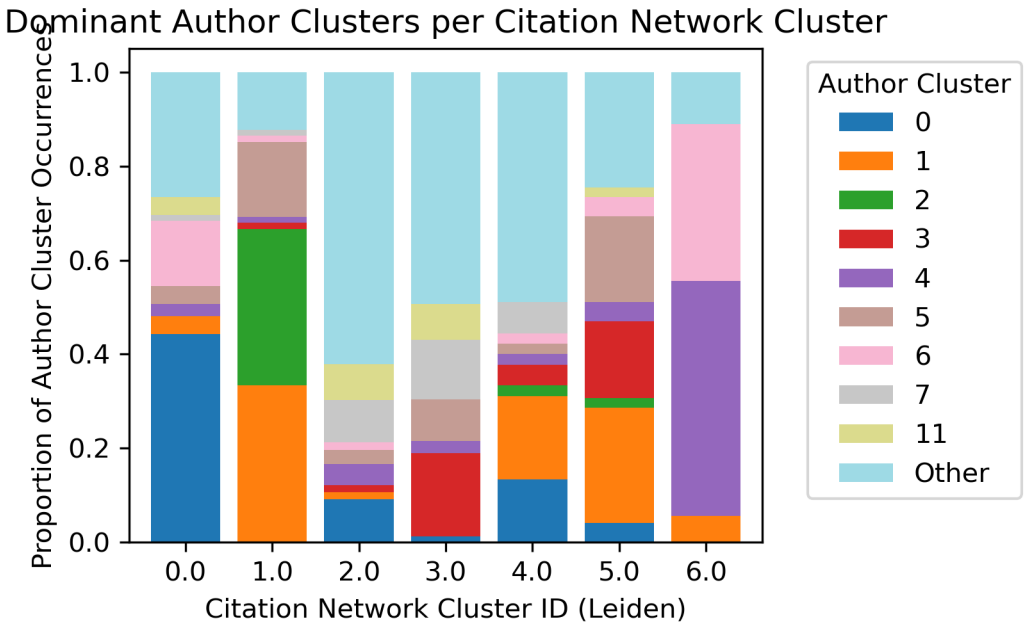
```
    if not isinstance(cluster_list, list) or len(cluster_list) == 0:
        return ["Other"]
    return [c if c in top_clusters else "Other" for c in cluster_list]

# Apply reduction
df_indexed['Reduced_Author_Cluster'] = (
    df_indexed['Author_clusters'].apply(reduce_clusters)
)
```

We then plotted the frequency distribution (as stacked bar charts) of co-authorship clusters per citation network cluster:

```
# Explode the lists into separate rows
df_exploded = df_indexed.explode('Reduced_Author_Cluster')

# Prepare count table
reduced_ct = pd.crosstab(df_exploded['Leiden_cluster'],
                         df_exploded['Reduced_Author_Cluster'])
reduced_ct = reduced_ct.div(reduced_ct.sum(axis=1),
                            axis=0)  # Normalize to proportions

# Plot
plt.figure(figsize=(12, 6))
reduced_ct.plot(kind='bar', stacked=True, colormap='tab20', width=0.8)

plt.title("Dominant Author Clusters per Citation Network Cluster")
plt.xlabel("Citation Network Cluster ID (Leiden)")
plt.ylabel("Proportion of Author Cluster Occurrences")
plt.xticks(rotation=0)
plt.legend(title="Author Cluster", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.savefig("coauth_cluster_dominance_per_cite_net_cluster.svg",
            format='svg')
plt.savefig("coauth_cluster_dominance_per_cite_net_cluster.png",
            format='png',
            dpi=300)

plt.show()
```

```
<Figure size 3600x1800 with 0 Axes>
```

(a) Dominance of clusters within citation clusters




(b)

Figure 6

The plot above (Figure Figure 6) shows that some of the top co-authorship clusters dominate several of the citation network clusters. This pattern suggests that much of the debate about megafauna extinction — and the corresponding citation clusters in the broader literature — may be shaped by a relatively small number of co-author communities. However, the "Other" category, which groups many smaller and less influential author clusters, still plays a major role, dominating three of the citation network clusters. Overall, these results indicate that while there are clear dominant groups who publish together and preferentially cite certain works, there also exists a large number of smaller, less prolific groups contributing to the literature.

### Formally Modelling Citation Cluster

In this section, we more formally model the relationships between the three variables — Discipline, Extinction Cause, and Co-Authorship — and citation network cluster membership. A Bayesian logit-softmax model allows us to quantify the contribution of each variable while explicitly incorporating uncertainty, providing a nuanced view of how these social and thematic

factors shape the literature. The multinomial logistic model estimates the impact each predictor has on the log-odds of a paper having membership in a given citation cluster, and the 'softmax' transformation is a link function that insures the probabilities are positive and sum to one. We included an intercept to improve identifiability and stability, which means that the coefficients in the model (these are essentially regression coefficients) are interpreted relative to a baseline cluster. The prior on is a Gaussian distribution with a standard deviation of 2, which is weakly informative and permits a wide range of effects per predictor.

To simplify the model and avoid excessive complexity, we used only the modal co-authorship cluster (i.e., the most frequent cluster) as a potential explanatory variable for each paper. This choice represents a trade-off: while it does not capture all possible combinations of author community affiliations, it enables a tractable analysis that directly tests whether belonging predominantly to a single co-authorship cluster increases the probability of assignment to a particular citation network cluster. In other words, whether strong community ties among authors systematically shape citation patterns. This choice is additionally justified by the visual dominance in the frequency plot above of single authorship clusters in each citation cluster — in most cases, one authorship cluster (or the 'other' group) dominates a given citation network cluster. Given our goal is not to predict which co-authorship clusters dominate a specific citation network cluster, we think this is a reasonable trade-off against model complexity.

```
def prime_cluster(cluster_list):
    if not isinstance(cluster_list, list) or len(cluster_list) == 0:
        return "Other"
    # Replace small clusters with "Other" first
    reduced = [c if c in top_clusters else "Other" for c in cluster_list]
    # Find the most common (modal) cluster
    modal_cluster = Counter(reduced).most_common(1)[0][0]
    return modal_cluster

df_indexed['Modal_Author_Cluster'] = (
    df_indexed['Author_clusters'].apply(prime_cluster)
)
```

```
# 1. Define predictor and target columns
predictor_cols = ['Categories', 'Extinction Cause', 'Modal_Author_Cluster']
target_col = 'Leiden_cluster'

# 2. Drop rows with missing values in predictors or target
df_clean = df_indexed.dropna(subset=predictor_cols + [target_col]).copy()

# Coerce all predictors to string to ensure consistent encoding
df_clean[predictor_cols] = df_clean[predictor_cols].astype(str)
```

```
# 3. Encode target variable (citation cluster)
y_encoder = LabelEncoder()
y = y_encoder.fit_transform(df_clean[target_col])  # shape (n_samples,)

# 4. One-hot encode predictor variables
X_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
X = X_encoder.fit_transform(df_clean[predictor_cols])

# 5. Get dimensions
n_classes = len(np.unique(y))
n_features = X.shape[1]
```

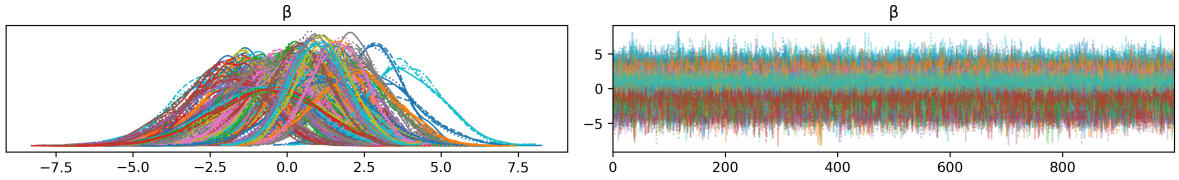
```
N, P = X.shape
K = n_classes

# augment X with ones for intercept
X_aug = np.column_stack([np.ones(N), X])  # shape (N, P+1)

with pm.Model() as model:
    X_shared = pm.Data("X_shared", X_aug)

    # We'll parameterize K-1 class coefficient vectors, and fix the last class to 0 (baseline
    #  _reduced shape: (P+1, K-1)
      = pm.Normal(" ", 0, 2.0, shape=(P+1, K-1))

    # add a zero column for the baseline class
    zero_col = pt.zeros((P+1, 1))
     _full = pt.concatenate([ , zero_col], axis=1)  # shape (P+1, K)

    logits = X_shared @  _full                          # (N, K)
    p = pm.math.softmax(logits, axis=1)                    # (N, K)

    y_obs = pm.Categorical("y_obs", p=p, observed=y)

    trace = pm.sample(1000, tune=1000, target_accept=0.9,
                      return_inferencedata=True, idata_kwargs={"log_likelihood": True},
                      progressbar=False)
    waic = pm.waic(trace)
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
```

```
NUTS: [ ]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took
/home/carleton@gea.mpg.de/miniconda3/envs/rapidgis/lib/python3.12/site-packages/arviz/stats/

For one or more samples the posterior variance of the log predictive densities exceeds 0.4.
See http://arxiv.org/abs/1507.04544 for details
```

Below, we produce a series of standard diagnostic plots for MCMC chains and posteriors.

```
az.plot_trace(trace, var_names=[" "])
plt.tight_layout()
plt.show()
```




```
az.summary(trace, var_names=[" "])[["r_hat","ess_bulk","ess_tail"]]
```

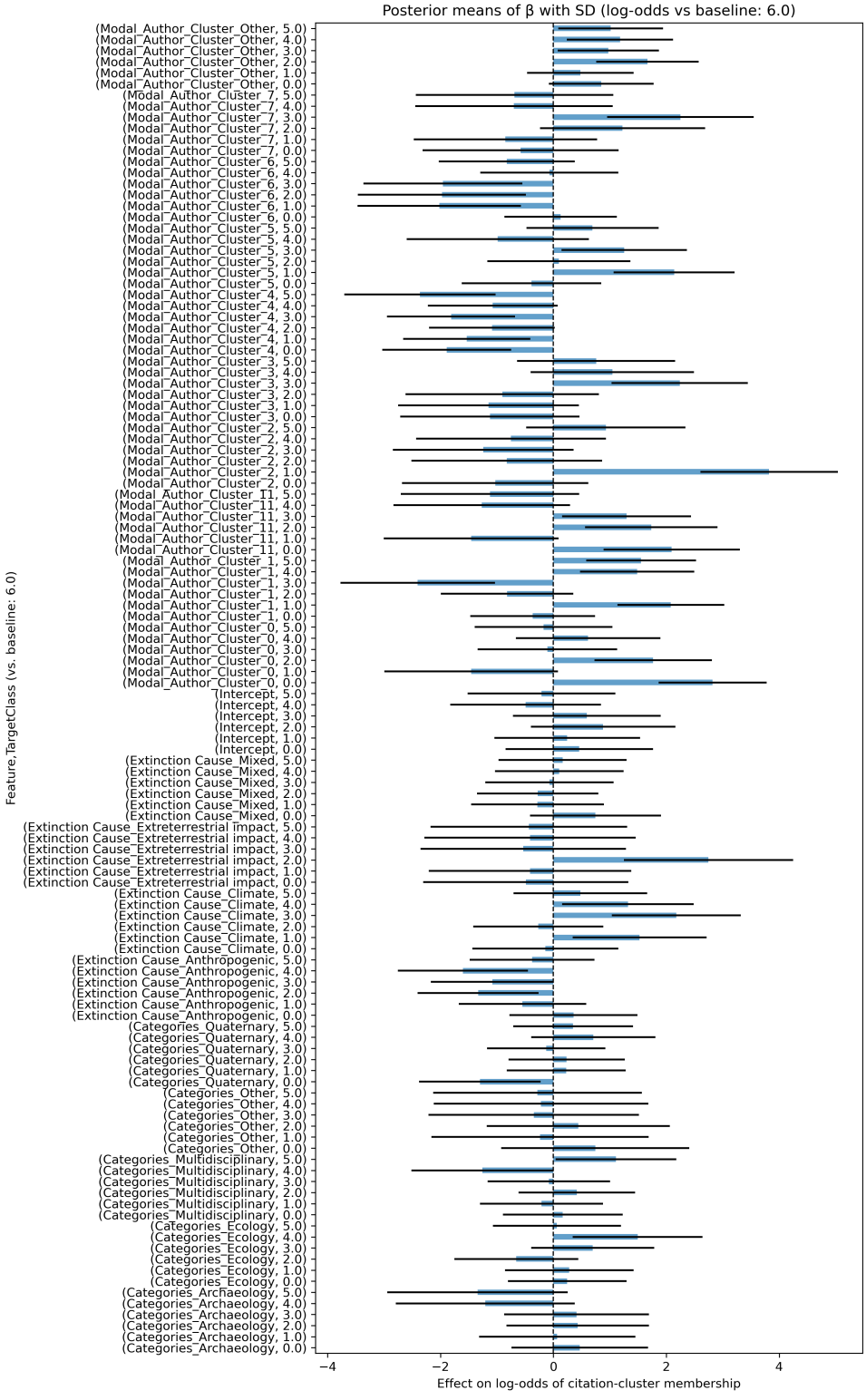
| | r_hat | ess_bulk | ess_tail |
|---|---|---|---|
| [0, 0] | 1.0 | 4633.0 | 3253.0 |
| [0, 1] | 1.0 | 3968.0 | 2882.0 |
| [0, 2] | 1.0 | 4785.0 | 3291.0 |
| [0, 3] | 1.0 | 4716.0 | 3149.0 |
| [0, 4] | 1.0 | 3924.0 | 2847.0 |
| ... | ... | ... | ... |
| [19, 1] | 1.0 | 4404.0 | 3576.0 |
| [19, 2] | 1.0 | 4603.0 | 3490.0 |
| [19, 3] | 1.0 | 4196.0 | 2884.0 |
| [19, 4] | 1.0 | 4346.0 | 2717.0 |
| [19, 5] | 1.0 | 4458.0 | 3104.0 |

The next two plots (Figures **?@fig-logodds-1** and Figure 7) is intended to show the modelling results in an interpretable way. The model aims to predict citation cluster membership, and one of the key parameters is the log-odds of cluster membership given a single predictor variable being true. The plot below represents these log-odds as a stacked set of box plots representing the posterior of the relevant coefficient. All log-odds are given as releative to an arbitrary baseline cluster in order to improve interpretability and stability (avoids label switching).

When the coefficient distribution overlaps zero, we would interpret that as little evidence for an effect of the relevant predictor variable on citation network cluster membership. If it's negative, then the corresponding predictor *reduces* the log-odds of membership in the relevant cluster. If it's positive, then that predictor variable appears (in our sample) to increase the odds of membership in the relevant citation cluster. So, effectively, any significant deviation from zero suggests that the predictor variable (e.g., Extinction Cause) has an effect on the log-odds of a given paper belonging to a given cluster.

```
# --- Names & dimensions ---
summary = az.summary(trace, var_names=[" "])

# baseline is the last class by construction
baseline_label = y_encoder.classes_[-1]

# feature names: add intercept to front
feature_names = X_encoder.get_feature_names_out(predictor_cols)
feature_names_aug = np.r_[["Intercept"], feature_names]  # length P+1

# target names: exclude baseline (last)
target_names_full = y_encoder.classes_
target_names_reduced = target_names_full[:-1]  # length K-1

# sanity check: lengths should match  shape
P1 = len(feature_names_aug)        # P+1
K1 = len(target_names_reduced)     # K-1
assert summary.shape[0] == P1 * K1, "Shape mismatch between  and names."

# Build MultiIndex in the order PyMC/ArviZ flattens  : (feature, class)
multi_index = pd.MultiIndex.from_product(
    [feature_names_aug, target_names_reduced],
    names=["Feature", "TargetClass (vs. baseline: %s)" % baseline_label]
)

summary.index = multi_index

# Optional: sort by feature then class
summary_sorted = summary.sort_index(level=0)

# --- Plot: posterior means with 1 SD bars (log-odds vs baseline) ---
fig, ax = plt.subplots(figsize=(10, 16))
summary_sorted["mean"].plot(
    kind="barh",
    xerr=summary_sorted["sd"],
```

```
    ax=ax,
    alpha=0.7
)
ax.axvline(0, color="k", linestyle="--", linewidth=1)
ax.set_title("Posterior means of   with SD (log-odds vs baseline: %s)" % baseline_label)
ax.set_xlabel("Effect on log-odds of citation-cluster membership")
plt.tight_layout()
plt.show()
```


Posterior means of β with SD (log-odds vs baseline: 6.0)
Feature,TargetClass (vs. baseline: 6.0)
(Modal_Author_Cluster_Other, 5.0)
(Modal_Author_Cluster_Other, 4.0)
(Modal_Author_Cluster_Other, 3.0)
(Modal_Author_Cluster_Other, 2.0)
(Modal_Author_Cluster_Other, 1.0)
(Modal_Author_Cluster_Other, 0.0)
(Modal_Author_Cluster_7, 5.0)
(Modal_Author_Cluster_7, 4.0)
(Modal_Author_Cluster_7, 3.0)
(Modal_Author_Cluster_7, 2.0)
(Modal_Author_Cluster_7, 1.0)
(Modal_Author_Cluster_7, 0.0)
(Modal_Author_Cluster_6, 5.0)
(Modal_Author_Cluster_6, 4.0)
(Modal_Author_Cluster_6, 3.0)
(Modal_Author_Cluster_6, 2.0)
(Modal_Author_Cluster_6, 1.0)
(Modal_Author_Cluster_6, 0.0)
(Modal_Author_Cluster_5, 5.0)
(Modal_Author_Cluster_5, 4.0)
(Modal_Author_Cluster_5, 3.0)
(Modal_Author_Cluster_5, 2.0)
(Modal_Author_Cluster_5, 1.0)
(Modal_Author_Cluster_5, 0.0)
(Modal_Author_Cluster_4, 5.0)
(Modal_Author_Cluster_4, 4.0)
(Modal_Author_Cluster_4, 3.0)
(Modal_Author_Cluster_4, 2.0)
(Modal_Author_Cluster_4, 1.0)
(Modal_Author_Cluster_4, 0.0)
(Modal_Author_Cluster_3, 5.0)
(Modal_Author_Cluster_3, 4.0)
(Modal_Author_Cluster_3, 3.0)
(Modal_Author_Cluster_3, 2.0)
(Modal_Author_Cluster_3, 1.0)
(Modal_Author_Cluster_3, 0.0)
(Modal_Author_Cluster_2, 5.0)
(Modal_Author_Cluster_2, 4.0)
(Modal_Author_Cluster_2, 3.0)
(Modal_Author_Cluster_2, 2.0)
(Modal_Author_Cluster_2, 1.0)
(Modal_Author_Cluster_2, 0.0)
(Modal_Author_Cluster_11, 5.0)
(Modal_Author_Cluster_11, 4.0)
(Modal_Author_Cluster_11, 3.0)
(Modal_Author_Cluster_11, 2.0)
(Modal_Author_Cluster_11, 1.0)
(Modal_Author_Cluster_11, 0.0)
(Modal_Author_Cluster_1, 5.0)
(Modal_Author_Cluster_1, 4.0)
(Modal_Author_Cluster_1, 3.0)
(Modal_Author_Cluster_1, 2.0)
(Modal_Author_Cluster_1, 1.0)
(Modal_Author_Cluster_1, 0.0)
(Modal_Author_Cluster_0, 5.0)
(Modal_Author_Cluster_0, 4.0)
(Modal_Author_Cluster_0, 3.0)
(Modal_Author_Cluster_0, 2.0)
(Modal_Author_Cluster_0, 1.0)
(Modal_Author_Cluster_0, 0.0)
(Intercept, 5.0)
(Intercept, 4.0)
(Intercept, 3.0)
(Intercept, 2.0)
(Intercept, 1.0)
(Intercept, 0.0)
(Extinction Cause_Mixed, 5.0)
(Extinction Cause_Mixed, 4.0)
(Extinction Cause_Mixed, 3.0)
(Extinction Cause_Mixed, 2.0)
(Extinction Cause_Mixed, 1.0)
(Extinction Cause_Mixed, 0.0)
(Extinction Cause_Extreterrestrial impact, 5.0)
(Extinction Cause_Extreterrestrial impact, 4.0)
(Extinction Cause_Extreterrestrial impact, 3.0)
(Extinction Cause_Extreterrestrial impact, 2.0)
(Extinction Cause_Extreterrestrial impact, 1.0)
(Extinction Cause_Extreterrestrial impact, 0.0)
(Extinction Cause_Climate, 5.0)
(Extinction Cause_Climate, 4.0)
(Extinction Cause_Climate, 3.0)
(Extinction Cause_Climate, 2.0)
(Extinction Cause_Climate, 1.0)
(Extinction Cause_Climate, 0.0)
(Extinction Cause_Anthropogenic, 5.0)
(Extinction Cause_Anthropogenic, 4.0)
(Extinction Cause_Anthropogenic, 3.0)
(Extinction Cause_Anthropogenic, 2.0)
(Extinction Cause_Anthropogenic, 1.0)
(Extinction Cause_Anthropogenic, 0.0)
(Categories_Quaternary, 5.0)
(Categories_Quaternary, 4.0)
(Categories_Quaternary, 3.0)
(Categories_Quaternary, 2.0)
(Categories_Quaternary, 1.0)
(Categories_Quaternary, 0.0)
(Categories_Other, 5.0)
(Categories_Other, 4.0)
(Categories_Other, 3.0)
(Categories_Other, 2.0)
(Categories_Other, 1.0)
(Categories_Other, 0.0)
(Categories_Multidisciplinary, 5.0)
(Categories_Multidisciplinary, 4.0)
(Categories_Multidisciplinary, 3.0)
(Categories_Multidisciplinary, 2.0)
(Categories_Multidisciplinary, 1.0)
(Categories_Multidisciplinary, 0.0)
(Categories_Ecology, 5.0)
(Categories_Ecology, 4.0)
(Categories_Ecology, 3.0)
(Categories_Ecology, 2.0)
(Categories_Ecology, 1.0)
(Categories_Ecology, 0.0)
(Categories_Archaeology, 5.0)
(Categories_Archaeology, 4.0)
(Categories_Archaeology, 3.0)
(Categories_Archaeology, 2.0)
(Categories_Archaeology, 1.0)
(Categories_Archaeology, 0.0)
−4
−2
0
2
4
Effect on log-odds of citation-cluster membership

```
# For each citation cluster
top_cited = {}

for cluster_id in sorted(df_indexed['Leiden_cluster'].unique()):
    top_papers = (
        df_indexed[df_indexed['Leiden_cluster'] == cluster_id]
        .sort_values('Times_cited_WoS', ascending=False)
        .head(5)[['Author', 'Publication_year', 'Article_Title']]
    )
    top_cited[cluster_id] = top_papers
```

We then collated the results by citation network cluster in an attempt to clarify the relationship. We also added the top cited papers in each cluster as text to the right of each panel to provide some context.

```
# 1) Extract summary and parse indices from ArviZ names like ' [fi, cj]'
summary_df = az.summary(trace, var_names=[" "], round_to=2).copy()

summary_df["feature_index"] = [
    int(s.split(",")[0].split("[")[1]) for s in summary_df.index
]
summary_df["class_index"] = [
    int(s.split(",")[1].split("]")[0]) for s in summary_df.index
]

# 2) Build augmented feature names (Intercept + one-hot names)
oh_names = X_encoder.get_feature_names_out(predictor_cols)  # length P
feature_names_aug = np.r_[["Intercept"], oh_names]          # length P+1

# Map feature indices -> names
summary_df["feature_name"] = summary_df["feature_index"].map(lambda i: feature_names_aug[i])

# 3) Feature groups (prefix-based); adjust to your actual prefixes
def group_of(name: str) -> str:
    if name == "Intercept":
        return "Intercept"
    if name.startswith("Modal_Author_Cluster") or name.startswith("Reduced_Author_Cluster"):
        return "Author Cluster"
    if name.startswith("Categories"):
        return "Discipline"
    if name.startswith("Extinction Cause"):
        return "Extinction Cause"
```

```
    return "Other"

summary_df["feature_group"] = summary_df["feature_name"].map(group_of)

# 4) Order features by groups: Intercept first, then Author, Discipline, Extinction
group_order_key = {
    "Intercept": 0, "Author Cluster": 1, "Discipline": 2, "Extinction Cause": 3, "Other": 4
}
ordered_features = (
    pd.Series(feature_names_aug)
      .sort_values(key=lambda x: x.map(lambda n: group_order_key.get(group_of(n), 9)))
      .tolist()
)
summary_df["feature_name"] = pd.Categorical(summary_df["feature_name"],
                                            categories=ordered_features, ordered=True)
summary_df = summary_df.sort_values(["class_index", "feature_name"])

# 5) Boundaries between feature groups for horizontal separators
group_boundaries = []
current_group = None
for i, fname in enumerate(ordered_features):
    g = group_of(fname)
    if g != current_group:
        group_boundaries.append(i - 0.5)
        current_group = g
group_boundaries = group_boundaries[1:]  # skip first boundary

# 6) Plot: one panel per non-baseline class (K-1)
class_labels_full = y_encoder.classes_
baseline_label = class_labels_full[-1]
class_labels_reduced = class_labels_full[:-1]  # indices 0..K-2

n_classes = summary_df["class_index"].nunique()
fig, axes = plt.subplots(n_classes, 1, figsize=(12, 2.8 * n_classes), sharex=True)

# handle case n_classes==1
if n_classes == 1:
    axes = [axes]

for c in range(n_classes):
    ax = axes[c]
    class_df = summary_df[summary_df["class_index"] == c].copy()
```

```
    class_df = class_df.sort_values("feature_name")

    ax.errorbar(
        x=class_df["mean"],
        y=class_df["feature_name"],
        xerr=class_df["sd"],
        fmt='o',
        ecolor='gray',
        elinewidth=1,
        capsize=3
    )
    ax.axvline(0, linestyle="--", color="black", linewidth=1)
    ax.set_title(f"Citation Cluster: {class_labels_reduced[c]} (vs baseline: {baseline_label]

    # dashed lines between feature groups
    for b in group_boundaries:
        ax.axhline(b, color="gray", linestyle="--", linewidth=0.5, alpha=0.5)

    # Optional: annotate top-cited per class label or index
    if 'top_cited' in globals() and (c in top_cited or class_labels_reduced[c] in top_cited)
        df_annot = top_cited.get(c, top_cited.get(class_labels_reduced[c], None))
        if df_annot is not None and len(df_annot):
            lines = [
                f"{str(row.get('Author','')).split(';')[0].strip()} "
                f"({int(row['Publication_year']) if pd.notna(row.get('Publication_year')) els
                f"{str(row.get('Article_Title',''))[:40]}..."
                for _, row in df_annot.iterrows()
            ]
            top_text = "\n".join(lines)
            ax.text(1.02, 0.5, top_text, transform=ax.transAxes, fontsize=8,
                    va='center', ha='left', family='monospace')

axes[-1].set_xlabel("Effect on log-odds of citation cluster (vs baseline)")
plt.tight_layout()
plt.subplots_adjust(right=0.8)

plt.savefig("citation_cluster_factors.svg", format='svg')
plt.savefig("citation_cluster_factors.png", format='png', dpi=300)
plt.show()
```

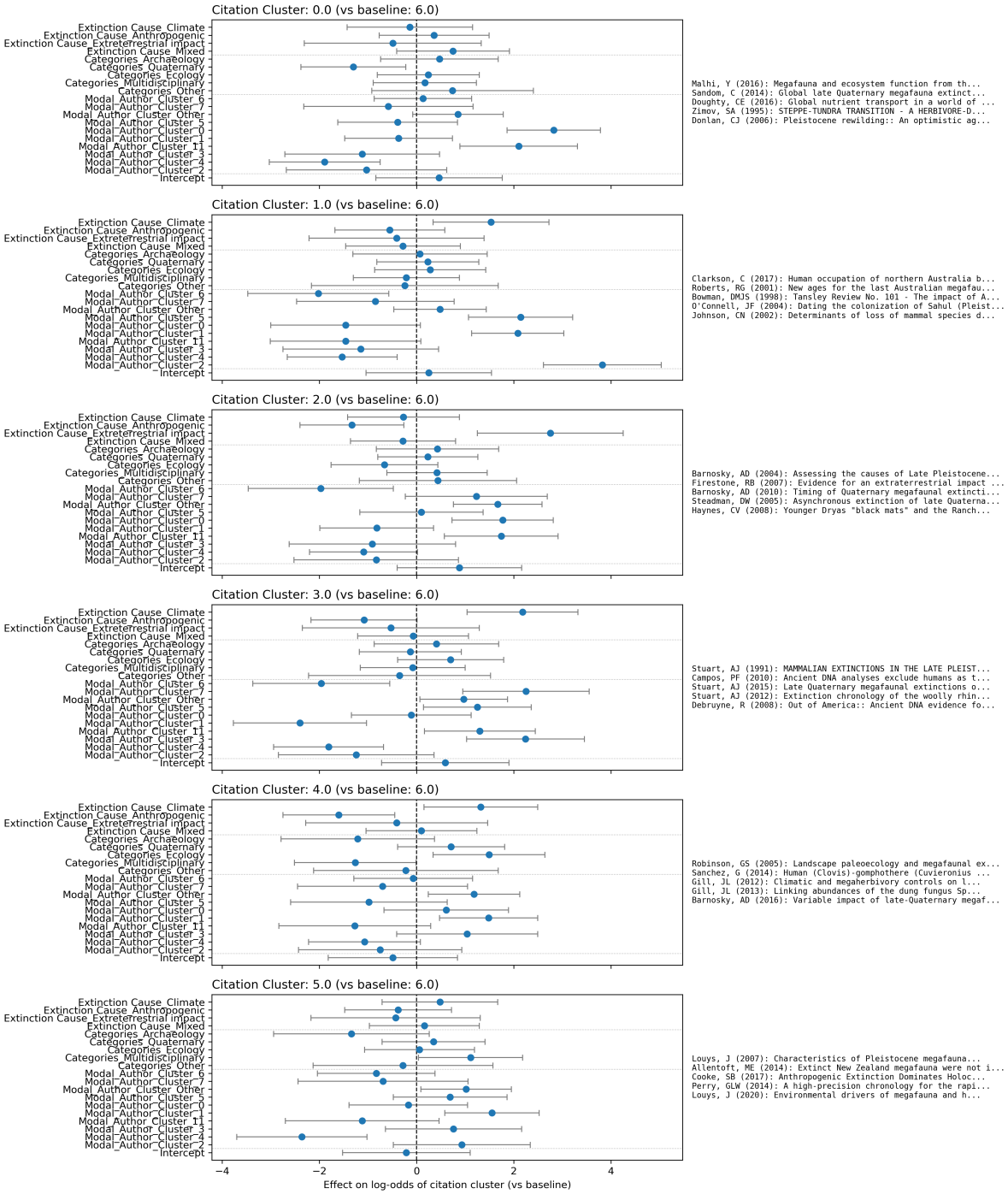



Figure 7: Posterior densities of predictor variables sorted by citation cluster

### Keyword Analysis

In this section, we investigate change over time in keyword frequency in the review sample. We use stream plotting (stacked area charts) showing the relative frequencies of different keywords changing over time (see Figure Figure 8). To focus on meaningful topical signals, we excluded generic or universal (in our original search query) terms (e.g., megafauna, extinction, quaternary) using a filtered root-word approach.

```
import re
# Define exclude list from search terms
exclude_roots = [
    "megafauna",
    "extinct",
    "quaternary",
    "pleistocene",
    "holocene",
    "record",
    "history",
    "mammal",
    "site"
]

def parse_keywords(keyword_str, exclude_roots=[]):
    if pd.isna(keyword_str):
        return []
    # Split and clean
    keywords = [kw.strip().lower() for kw in keyword_str.split(';') if kw.strip()]
    # Filter: exclude any keyword containing any of the root terms
    filtered_keywords = []
    for kw in keywords:
        if not any(re.search(rf"\b{root}", kw) for root in exclude_roots):
            filtered_keywords.append(kw)
    return filtered_keywords

# Parse and clean keywords
df_indexed['Parsed_Keywords'] = df_indexed['Keyword'].apply(
    lambda x: parse_keywords(x, exclude_roots=exclude_roots)
)

# Explode so each row is (DOI, year, keyword)
df_keywords = df_indexed[
    ['Publication_year', 'Parsed_Keywords']
].explode('Parsed_Keywords')
```

```
# Drop any rows missing year or keyword
df_keywords = df_keywords.dropna(subset=['Publication_year', 'Parsed_Keywords'])

# Count overall keyword frequencies
keyword_counts = Counter(df_keywords['Parsed_Keywords'])

# Pick top N keywords (after exclusion)
top_n = 10
top_keywords = {kw for kw, _ in keyword_counts.most_common(top_n)}

# Reduce keywords: keep top, else "Other"
def reduce_keyword(kw):
    return kw if kw in top_keywords else "Other"

df_keywords['Reduced_Keyword'] = df_keywords['Parsed_Keywords'].apply(reduce_keyword)
```

```
# Count number of papers per year per keyword
keyword_year_counts = (
    df_keywords
    .groupby(['Publication_year', 'Reduced_Keyword'])
    .size()
    .unstack(fill_value=0)
    .sort_index()
)

# Normalize to proportions if you prefer (optional)
keyword_year_props = keyword_year_counts.div(keyword_year_counts.sum(axis=1),
                                             axis=0)
```

```
# Choose colors: color-blind friendly palette
color_palette = plt.get_cmap('tab10').colors

# Define hatch patterns to cycle through
hatch_patterns = ['/', '\\', '|', '-', '+', 'x', 'o', 'O', '.', '*']

# Drop "Other" column if present
plot_df = keyword_year_counts.drop(columns="Other", errors='ignore')

# Sort columns by total
total_counts = plot_df.sum().sort_values(ascending=False)
plot_df = plot_df[total_counts.index]
```

```
# Normalize to proportions
plot_df_prop = plot_df.div(plot_df.sum(axis=1), axis=0)

# Smooth
plot_df_prop_smooth = plot_df_prop.rolling(window=2, min_periods=1).mean()

# Start figure
fig, ax = plt.subplots(figsize=(14, 7))

# Stack each area manually with hatch
bottom = None

for i, col in enumerate(plot_df_prop_smooth.columns):
    data = plot_df_prop_smooth[col]
    color = color_palette[i % len(color_palette)]
    hatch = hatch_patterns[i % len(hatch_patterns)]
    ax.fill_between(data.index,
                    bottom if bottom is not None else 0,
                    (bottom + data) if bottom is not None else data,
                    facecolor=color,
                    edgecolor='black',
                    hatch=hatch,
                    linewidth=0.5,
                    alpha=0.8,
                    label=col)
    bottom = data if bottom is None else bottom + data

# Title and labels
ax.set_title("Relative Frequency of Top 10 Keywords Over Time")
ax.set_xlabel("Publication Year")
ax.set_ylabel("Proportion of Papers")
ax.set_ylim(0, 1)

# Legend
ax.legend(title="Keyword", bbox_to_anchor=(1.05, 1), loc='upper left')

# Layout
plt.tight_layout(rect=[0, 0, 0.85, 1])

plt.savefig("keyword_trends_patterns.svg", format='svg')
plt.savefig("keyword_trends_patterns.png", format='png', dpi=300)
```

```
plt.show()
```

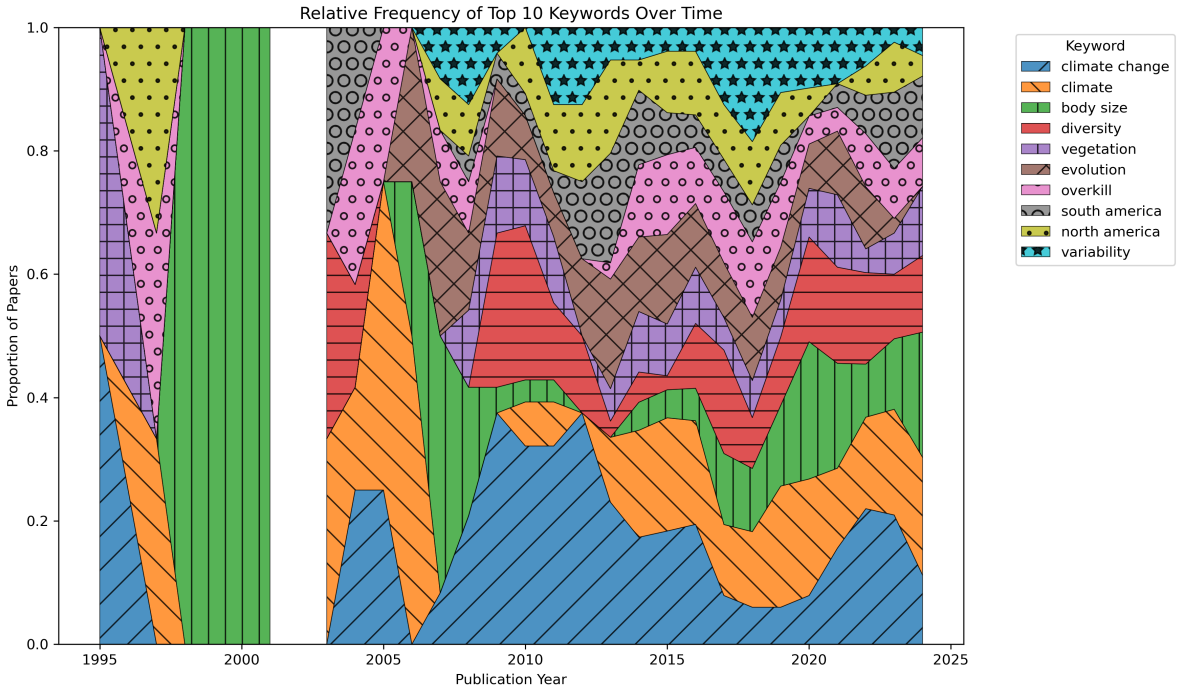



Figure 8: Change over time in top keywords

The gap above was caused by the absence of papers in 2001 and 2002 with keywords in the global top 10 (referring to the whole literature sample covering 1995-2024). This can be seen if we look at a table of records for the gap years and count the number of keywords for each paper that also occur in the global top ten (and that aren't excluded for other reasons):

```
# Define years of interest
years_focus = range(2000, 2006)

# Filter for those years
df_focus = df_indexed[df_indexed['Publication_year'].isin(years_focus)].copy()

# Helper: count raw keywords before exclusion
def count_raw_keywords(kw_str):
    if pd.isna(kw_str):
        return 0
    return len([k.strip() for k in kw_str.split(';') if k.strip()])
```

```
# Helper: count non-excluded parsed keywords
def count_non_excluded(parsed_list):
    if isinstance(parsed_list, list):
        return len(parsed_list)
    return 0

# Helper: count keywords in top N list
def count_top_n(parsed_list):
    if isinstance(parsed_list, list):
        return sum(1 for kw in parsed_list if kw in top_keywords)
    return 0

# Compute counts
df_focus['Total_Keyword_Count'] = df_focus['Keyword'].apply(count_raw_keywords)
df_focus['Non_Excluded_Count'] = (
    df_focus['Parsed_Keywords'].apply(count_non_excluded)
)
df_focus['TopN_Count'] = df_focus['Parsed_Keywords'].apply(count_top_n)

# Select and sort
df_table = df_focus[['Publication_year',
                     'Total_Keyword_Count',
                     'Non_Excluded_Count',
                     'TopN_Count']].sort_values(by='Publication_year')

# Make a copy
df_table_short = df_table.copy()

# Rename columns
df_table_short.columns = ['Year', 'Total_KW', 'NonExcl_KW', 'TopN_KW']

# Display
df_table_short
```

| DOI | Year | Total_KW | NonExcl_KW | TopN_KW |
|---|---|---|---|---|
| 10.1017/s0003598x00089195 | 2001 | 3 | 1 | 0 |
| 10.1126/science.1060264 | 2001 | 6 | 6 | 0 |
| 10.1098/rspb.2002.2130 | 2002 | 1 | 0 | 0 |
| 10.1073/pnas.232126899 | 2002 | 9 | 9 | 0 |
| 10.1046/j.1365-294x.2003.01701.x | 2003 | 10 | 9 | 1 |

| DOI | Year | Total_KW | NonExcl_KW | TopN_KW |
|---|---|---|---|---|
| 10.1016/s0895-9811(02)00145-1 | 2003 | 10 | 6 | 1 |
| 10.1016/s1040-6182(02)00201-x | 2003 | 5 | 4 | 1 |
| 10.1016/j.jhevol.2004.05.005 | 2004 | 10 | 7 | 0 |
| 10.1016/j.quascirev.2004.03.011 | 2004 | 10 | 7 | 0 |
| 10.1126/science.1101476 | 2004 | 10 | 6 | 2 |
| 10.1016/j.earscirev.2004.04.002 | 2004 | 10 | 8 | 0 |
| 10.1111/j.1442-9993.2004.01389.x | 2004 | 2 | 0 | 0 |
| 10.1130/g19957.1 | 2004 | 8 | 6 | 0 |
| 10.1016/j.jas.2003.11.005 | 2004 | 10 | 8 | 0 |
| 10.1080/03115510408619286 | 2004 | 10 | 8 | 0 |
| 10.1016/s1342-937x(05)70796-6 | 2004 | 0 | 0 | 0 |
| 10.1890/03-4064 | 2005 | 10 | 10 | 1 |
| 10.1016/j.quascirev.2004.08.019 | 2005 | 4 | 3 | 0 |
| 10.1073/pnas.0502777102 | 2005 | 10 | 8 | 0 |
| 10.1073/pnas.0408975102 | 2005 | 4 | 3 | 0 |

Counting keywords to determine if the cause of the plotting gap

To better capture the variation in the trend over time in prominent key words, we can instead examine the top 10 keywords in 5-year bins and simple list these top keywords per binned period in alphabetical order for easy visual parsing (see Figure Figure 9):

```
top_n = 10

# function for binning into 5-year bins
def assign_five_year_bin(year):
    if pd.isna(year):
        return None
    return int(year // 5 * 5)

df_indexed['Parsed_Keywords'] = df_indexed['Keyword'].apply(
    lambda x: parse_keywords(x, exclude_roots=exclude_roots)
)

df_keywords = df_indexed[
    ['Publication_year', 'Parsed_Keywords']
].explode('Parsed_Keywords')

df_keywords = df_keywords.dropna(
    subset=['Publication_year', 'Parsed_Keywords']
```

```
)

df_keywords['Five_Year_Bin'] = df_keywords['Publication_year'].apply(
    assign_five_year_bin
)

top_keywords_per_bin = {}

for bin_year in sorted(df_keywords['Five_Year_Bin'].dropna().unique()):
    bin_df = df_keywords[df_keywords['Five_Year_Bin'] == bin_year]
    bin_counts = Counter(bin_df['Parsed_Keywords'])
    top_keywords = [kw for kw, _ in bin_counts.most_common(top_n)]
    top_keywords_per_bin[bin_year] = sorted(top_keywords)

# plotting as a series of 'cards'
bin_years_sorted = list(top_keywords_per_bin.keys())
n_bins = len(bin_years_sorted)
top_row_bins = bin_years_sorted[:4]
bottom_row_bins = bin_years_sorted[4:]

fig, axes = plt.subplots(
    2,
    max(len(top_row_bins), len(bottom_row_bins)),
    figsize=(
        3 * max(len(top_row_bins), len(bottom_row_bins)),
        10
    ),
    constrained_layout=True
)

# Flatten axes array for easy indexing
axes_flat = axes.flatten()

# Card colorspatches
card_colors = ['#f8f9fa', '#e9ecef']

# Plot cards
for i, bin_year in enumerate(bin_years_sorted):
    ax = axes_flat[i]
    ax.axis("off")
    rect = patches.FancyBboxPatch((0, 0), 1, 1,
                                  boxstyle="round,pad=0.02",
```

```
                                    transform=ax.transAxes,
                                    facecolor=card_colors[i % len(card_colors)],
                                    edgecolor='gray',
                                    linewidth=1.0)
    ax.add_patch(rect)

    # Title
    ax.text(0.5, 0.92, f"{int(bin_year)}s", fontsize=12, fontweight='bold',
            ha="center", va="top", transform=ax.transAxes)

    # Keywords
    keywords = top_keywords_per_bin[bin_year]
    for j, kw in enumerate(keywords):
        y_pos = 0.85 - j * 0.075
        ax.text(0.5, y_pos, kw,
                fontsize=10, ha="center", va="top",
                rotation=30, rotation_mode='anchor',
                transform=ax.transAxes)

# Hide any unused subplots
for k in range(len(bin_years_sorted), len(axes_flat)):
    axes_flat[k].axis("off")

fig.suptitle("Top 10 Keywords in 5-year Bins", fontsize=16, y=1.03)

plt.savefig("keyword_trends_binned.svg", format='svg')
plt.savefig("keyword_trends_binned.png", format='png', dpi=300)

plt.show()
```


Figure 9: Top ten keywords per 5-year bin