**PROMPT**

I am an academic geographer embarking on a new project called GLOS: Geographic Lens on Stories. The goal is to develop computational tools for comparative studies of folkloric text, focusing on folktales and creation myths to start. I do have considerable experience in older NLP methods (topic modeling, vector similarity measures, etc.), but limited experience with new machine learning and LLM methodologies.

In early explorations I have digitized the canonical folklore indexes - Thompson's motif index and Uther's tale types index, logged them in a postgres databases, generated embeddings for each, and built a simple tool to take any entered text generated an embedding for it, and return alternatively the 10 nearest neighbor motifs or 10 nearest neighbor tale types. I have begun recording geographic metadata for each but this is exceptionally difficult because locational information may be in the form of countries, historical regions, tribes, or languages; footprints for countries and modern provinces are straigthforward to come by but the rest are not.

The results of the simple nearest neighbor tool are interesting, but not really satisfying. The goal is to identify semantic/conceptual similarities apart from term co-occurence and efficacy is modest so far.

As a separate higher level experiment I have digitized a sampler of 9 creation myths from "Primal Myths" by B. Sproul, and have used embedding to cluster the 60-70 segments from the myths and asked an LLM model to summarize the concatenated segments in natural language. Again, the results are modest. The challenge is abstracting concepts from segments pulled from myths. This script also generates similarity measures for the 9 myths and visualizations of them; those are promising but somewhat 'black box-ish' as they derive from distributions of segment embeddings.

I would like you to consider all of this and propose some changes to these methods, or entirely new ones that help me create formal measures of concept content - perhaps leveraging the motif and tale-type indexes mentioned earlier.

The overarching goal is to tie these comparisons to geographic locales, exposing similarities and differences in folkloric stories across cultures.

**RESPONSE**

Thank you for sharing your fascinating GLOS project. The intersection of geography, folklore, and computational methods is a rich area for exploration. I can see you've already built a solid foundation with your embedding approaches and database work. Let me propose some methodological improvements and new directions that might help you achieve your goals.

***Enhancing Your Current Approaches***

**1. Hierarchical Embedding Models for Folklore**

Your current embedding approach might be treating all text at the same level of granularity. Consider implementing a hierarchical embedding structure:

* Motif-level embeddings: Fine-tuned specifically on folkloric motifs
* Tale-type embeddings: Capturing broader narrative structures
* Theme embeddings: Higher-level conceptual patterns across stories

This hierarchical approach could help you distinguish between surface-level textual similarities and deeper conceptual connections. For example, two creation myths might use different imagery (flood vs. cosmic egg) but express the same fundamental concept (order emerging from chaos).

**2. Contrastive Learning for Folkloric Elements**

Your current nearest-neighbor approach might benefit from contrastive learning techniques:

# Conceptual implementation of contrastive learning for folklore

def contrastive\_loss(anchor, positive, negative, margin=1.0):

"""

Anchor: embedding of a folklore motif

Positive: embedding of semantically similar motif (perhaps from different culture)

Negative: embedding of unrelated motif

"""

d\_pos = torch.sum((anchor - positive) \*\* 2)

d\_neg = torch.sum((anchor - negative) \*\* 2)

return torch.clamp(d\_pos - d\_neg + margin, min=0)

This would explicitly teach your model to recognize *conceptual similarities across different cultural expressions* while distinguishing them from unrelated concepts.

**New Methodological Directions**

**1. Knowledge Graph Approach for Folkloric Elements**

Instead of relying solely on embeddings, consider building a knowledge graph that explicitly represents relationships between motifs, tale types, and geographic locations:

# Pseudocode for knowledge graph construction

G = nx.DiGraph()

# Add nodes for different entity types

for motif in thompson\_motifs:

G.add\_node(motif.id, type='motif', description=motif.description)

for tale\_type in uther\_types:

G.add\_node(tale\_type.id, type='tale\_type', description=tale\_type.description)

for location in geographic\_entities:

G.add\_node(location.id, type='location',

name=location.name,

coordinates=location.coordinates,

region\_type=location.type) # country, historical region, etc.

# Add edges representing relationships

for motif\_occurrence in motif\_occurrences:

G.add\_edge(motif\_occurrence.motif\_id, motif\_occurrence.location\_id,

weight=motif\_occurrence.frequency)

This approach would allow you to:

* Explicitly model relationships between folklore elements
* Perform graph-based analyses (centrality, community detection)
* Integrate geographic information directly into your analytical framework

**2. Semantic Frame Analysis for Folkloric Concepts**

Inspired by FrameNet in linguistics, you could develop a folkloric frame system that categorizes narrative elements by their conceptual roles rather than surface text:

# Example frames for creation myths

creation\_frames = {

'primordial\_substance': {

'roles': ['substance', 'state', 'properties'],

'examples': ['water', 'void', 'egg']

},

'creator\_entity': {

'roles': ['entity', 'attributes', 'motivation'],

'examples': ['deity', 'animal', 'first being']

},

'creation\_process': {

'roles': ['action', 'duration', 'stages'],

'examples': ['separation', 'emergence', 'transformation']

}

}

You could train a classifier to identify these frames in your texts, allowing for cross-cultural comparison at a more abstract level.

Does this entail enumerating the examples ahead of time, or can those be elicited by an LLM? e.g. ['separation', 'emergence', 'transformation'] might not include all categorical possibilities. For creation myths, with only 124 to hand, what constitutes a training set?

**3. Leveraging LLMs for Structured Extraction**

Instead of using LLMs just for summarization, you could prompt them to extract structured information from folkloric texts:

def extract\_folkloric\_structure(text):

prompt = f"""

Analyze the following folkloric text and extract its structural elements according to these categories:

1. Agents: Who are the primary actors in this narrative?

2. Actions: What key actions or transformations occur?

3. Objects: What significant objects or elements appear?

4. Spatial Setting: What geographic or cosmic spaces are depicted?

5. Temporal Setting: What time frame or sequence is established?

6. Causality: What causes and effects are described?

Text: {text}

Response format: JSON with the categories as keys and extracted elements as values.

"""

response = llm\_client.generate(prompt=prompt)

return json.loads(response)

This approach would give you structured data that's more amenable to comparative analysis than raw embeddings.

I don’t yet have a corpus of folktales. Named agents do not replicate across tales or myths – would have to be agent/type I think (deity, animal, natural force (flood as agent?) etc.)

**4. Geographic Representation Enhancement**

For your geographic metadata challenge, consider:

* Hierarchical Geographic Ontology: Build a hierarchical structure where historical regions, tribes, and languages can be linked to modern geographic entities:

geo\_hierarchy = {

"historical\_region": {

"Mesopotamia": {

"modern\_regions": ["Iraq", "Syria", "Kuwait"],

"approximate\_coordinates": [33.1, 44.6], # Centroid

"time\_period": "3100 BCE - 539 BCE"

}

},

"linguistic\_group": {

"Nahuatl": {

"modern\_regions": ["Mexico"],

"approximate\_coordinates": [19.4, -99.1], # Mexico City area

"language\_family": "Uto-Aztecan"

}

}

}

* Gazetteer Integration: Integrate historical gazetteers like Pleiades (for ancient places) or the World Historical Gazetteer:

# Query historical gazetteers for place names

def query\_historical\_gazetteer(place\_name):

# Example using World Historical Gazetteer API

response = requests.get(

f"https://whgazetteer.org/api/place",

params={"q": place\_name}

)

return response.json()

**Integration and Visualization**

To tie these approaches together, consider:

* Spatial-Conceptual Heatmaps: Generate heatmaps showing the distribution of specific motifs or conceptual frames across geographic regions.
* Narrative Flow Maps: Visualize how narrative elements move across geographic space, showing cultural transmission patterns. that data is not available
* Interactive Exploration Interface: Build a tool allowing scholars to explore the relationships between geographic regions and folklore elements dynamically.

**Implementation Strategy**

Given your background, I would suggest an incremental approach:

* Start by enhancing your embedding model with domain-specific training on folkloric texts.
* Implement the knowledge graph approach for a subset of your data to evaluate its effectiveness.
* Experiment with the LLM-based structured extraction on a small corpus of well-documented myths.
* Develop a prototype visualization that integrates geographic information with your analysis results.

Would you like me to elaborate on any of these approaches or perhaps suggest a specific implementation plan for one of them? I'm particularly interested in how you're currently generating embeddings and what challenges you've encountered with the geographic metadata.

**PROMPT**

I attached the script used to generate embeddings for Thompson's motifs, which I have stored in a postgres table. Other tables hold metadata I can use for filtering steps in apps. The motif types include 'references' e.g. "Bleeding wood as sign of Doomsday. Referenced in Jewish religious texts" that ostensibly provide useful context.

Re: geographic challenges stem from unmappability of historical regions, tribes, and languages - given that bounderies/spatial footprints of these are difficult or impossible to determine; extimations will generate "maps that lie" in the Mark Monmonier phrasing.

I have developed a working table with e.g. motif\_id and its 'ref\_term', trying to geolocate these (languages, religions, tribes) is next to impossible, so some kind of novel normalization technique is necessary to perform any kind of spatial analysis. Not hopeful.

examples of 'ref\_term':

Greek-Roman, Guaray', Gypsy, Halmahera, Hanover, Hebrew, Herero, Hindu

**RESPONSE**

I appreciate you sharing your embedding script and more details about the geographic challenges you're facing with the GLOS project. Let me provide some thoughts on potential improvements to your methods and suggestions for addressing the geographic normalization issues.

**Current Embedding Approach Analysis**

Looking at your script, I notice you're using OpenAI's text-embedding-3-small model to generate embeddings for Thompson's motifs. This is a good starting point as it provides semantic understanding beyond keyword matching. However, there are some potential enhancements we could consider:

**1. Domain-Specific Fine-Tuning**

The general-purpose embeddings may not fully capture the nuanced connections in folkloric texts. Consider fine-tuning your embedding model on a corpus of folkloric texts to better capture domain-specific relationships:

def create\_contrastive\_training\_pairs(motifs):

"""

Create training pairs for contrastive learning:

- Similar motifs from different cultures as positive pairs

isn’t this already encoded in the TMI, by means of its hierarchy?

- Unrelated motifs as negative pairs

"""

pairs = []

for motif in motifs:

# Find similar motifs based on expert annotations or thematic categorization

similar\_motifs = find\_similar\_motifs(motif, similarity\_threshold=0.7)

# Create positive pairs (anchor, positive)

for similar in similar\_motifs:

if similar['culture'] != motif['culture']: # Cross-cultural similarity

pairs.append((motif['text'], similar['text'], 1)) # 1 indicates positive pair

# Create negative pairs (anchor, negative)

dissimilar\_motifs = find\_dissimilar\_motifs(motif)

for dissimilar in dissimilar\_motifs:

pairs.append((motif['text'], dissimilar['text'], 0)) # 0 indicates negative pair

return pairs

This approach would help your embeddings differentiate between surface-level textual similarity and deeper conceptual connections, which seems to be one of your challenges.

How does this fit in or impact the architecture of the current tool (get embedding for given text; finds nearest neighbors in database of all embeddings – a simple vector distance?)

**2. Contextual Embeddings with Metadata**

Your script currently embeds only the text content. Consider incorporating metadata such as cultural context or categorical information into the embedding process:

async def process\_batch\_with\_context(batch):

"""Process a batch of motifs with their contextual information."""

motif\_ids, texts, contexts = zip(\*batch)

# Create enhanced prompts that include contextual information

enhanced\_texts = [f"Context: {context}\nMotif: {text}"

for text, context in zip(texts, contexts)]

embeddings = await get\_embeddings(enhanced\_texts)

return list(zip(motif\_ids, texts, contexts, embeddings))

This approach can help provide more nuanced embeddings that account for cultural context, which seems to be important for your comparative analysis.

Elaborate upon context beyond the ref\_term given

**Addressing Geographic Challenges**

The examples of reference terms you shared (Greek-Roman, Guaray', Gypsy, etc.) highlight the challenge of mapping folkloric elements to geographic locations. Here are some approaches that might help:

**1. Multi-Level Geographic Representation**

Instead of trying to map everything to precise boundaries, consider a multi-level approach that explicitly acknowledges uncertainty:

def create\_geographic\_representation(ref\_term):

"""

Create a multi-level geographic representation for a reference term.

"""

geo\_info = {

'original\_term': ref\_term,

'confidence': 'low', # Default confidence level

'representations': []

}

# Check for exact matches in our gazetteer

exact\_match = lookup\_exact\_match(ref\_term)

if exact\_match:

geo\_info['representations'].append({

'type': 'exact',

'geometry': exact\_match['geometry'],

'time\_period': exact\_match['time\_period'],

'confidence': 'high'

})

geo\_info['confidence'] = 'high'

return geo\_info

# Check for linguistic/cultural group mapping

cultural\_match = lookup\_cultural\_group(ref\_term)

if cultural\_match:

geo\_info['representations'].append({

'type': 'cultural\_region',

'geometry': cultural\_match['approximate\_region'],

'time\_period': cultural\_match['time\_period'],

'confidence': 'medium',

'notes': 'Approximate cultural region based on historical records'

})

geo\_info['confidence'] = 'medium'

# For completely unmappable terms, provide contextual information

if not geo\_info['representations']:

contextual\_info = get\_contextual\_information(ref\_term)

if contextual\_info:

geo\_info['representations'].append({

'type': 'contextual',

'related\_regions': contextual\_info['related\_regions'],

'description': contextual\_info['description'],

'confidence': 'low',

'notes': 'Cannot be mapped directly; providing related information'

})

return geo\_info

This approach acknowledges the inherent uncertainty in mapping historical cultural entities while still providing useful geographic context.

This suggests side-by-side locational figures: a geographic map, a language matrix, a religion matrix. Any given tale or text would “light up” areas of one , some, or all of these. It does not enable any kind of spatial query however – nothing can I believe.

**2. Cultural Distance Matrix**

Instead of trying to place everything on a continuous geographic space, consider creating a "cultural distance matrix" that represents proximity based on both geographic and cultural factors:

def build\_cultural\_distance\_matrix(ref\_terms):

"""

Build a distance matrix between cultural references that accounts for

geographic proximity, linguistic similarity, and historical interaction.

"""

n = len(ref\_terms)

distance\_matrix = np.zeros((n, n))

for i in range(n):

for j in range(i+1, n):

# Get various distance metrics

geographic\_dist = calculate\_geographic\_distance(ref\_terms[i], ref\_terms[j])

linguistic\_dist = calculate\_linguistic\_distance(ref\_terms[i], ref\_terms[j])

historical\_dist = calculate\_historical\_interaction(ref\_terms[i], ref\_terms[j])

# Combine into a single distance metric (with weights)

combined\_dist = (0.4 \* geographic\_dist +

0.3 \* linguistic\_dist +

0.3 \* historical\_dist)

distance\_matrix[i, j] = combined\_dist

distance\_matrix[j, i] = combined\_dist # Matrix is symmetric

return distance\_matrix, ref\_terms

This matrix could then be used for clustering or visualization, representing cultural proximity without requiring precise geographic boundaries.

**3. Contextual Geographic Embedding**

Create embeddings that represent the geographic context of a reference term based on historical and cultural information, even when precise boundaries don't exist:

def create\_geo\_context\_embedding(ref\_term):

"""

Create an embedding that represents the geographic context of a reference term.

"""

# Construct a rich textual description of the geographic context

context\_description = f"The term '{ref\_term}' refers to "

# Add known information

term\_info = get\_term\_info(ref\_term)

if term\_info.get('type') == 'linguistic\_group':

context\_description += f"a linguistic group that historically occupied "

context\_description += f"regions in what is now approximately {', '.join(term\_info.get('modern\_regions', []))}. "

elif term\_info.get('type') == 'cultural\_group':

context\_description += f"a cultural group associated with "

context\_description += f"{term\_info.get('description', '')}. "

# Add relevant geographic relationships

related\_groups = get\_related\_groups(ref\_term)

if related\_groups:

context\_description += f"This group had cultural interactions with {', '.join(related\_groups)}. "

# Create an embedding of this rich geographic context

embedding = get\_embedding(context\_description)

return {

'ref\_term': ref\_term,

'context\_description': context\_description,

'geo\_context\_embedding': embedding

}

These geographic context embeddings could then be used to find similarities between reference terms based on their cultural and historical geographic relationships, rather than precise coordinates.

**Novel Approach for Folkloric-Geographic Analysis**

Let me propose a comprehensive approach that integrates both the folkloric content analysis and the geographic challenges:

**1. Cultural Knowledge Graph**

Instead of trying to map everything to a traditional geographic coordinate system, build a knowledge graph that represents relationships between cultural entities, motifs, and approximate regions:

def build\_cultural\_knowledge\_graph():

"""

Build a knowledge graph of cultural entities, motifs, and their relationships.

"""

G = nx.Graph()

# Add nodes for motifs

for motif in get\_all\_motifs():

G.add\_node(motif['id'],

type='motif',

text=motif['text'],

embedding=motif['embedding'])

# Add nodes for cultural references

for ref\_term in get\_all\_ref\_terms():

G.add\_node(ref\_term,

type='cultural\_reference',

geo\_context=create\_geo\_context\_embedding(ref\_term))

# Add edges between motifs and their cultural references

for motif\_id, ref\_term in get\_motif\_references():

G.add\_edge(motif\_id, ref\_term, type='referenced\_in')

# Add edges between culturally related references

for ref1, ref2, similarity in get\_cultural\_similarities():

if similarity > 0.5: # Only add edges above a threshold

G.add\_edge(ref1, ref2,

type='cultural\_similarity',

weight=similarity)

return G

This graph could then be used for various analyses:

* Finding motifs that appear in culturally similar contexts
* Identifying cultural "clusters" that share similar folkloric elements
* Tracing the potential diffusion of motifs across cultural boundaries

**2. Structured Motif Analysis with LLMs**

Use LLMs to extract structured information from motifs, allowing for more nuanced comparisons:

def extract\_motif\_structure(motif\_text):

"""

Use an LLM to extract structured information from a motif.

"""

prompt = f"""

Analyze the following folkloric motif and extract structured information:

Motif: {motif\_text}

Please extract the following elements:

1. Actors (e.g., gods, animals, humans)

2. Actions (e.g., creation, transformation, punishment)

3. Objects (e.g., natural elements, magical items)

4. Concepts (e.g., death, rebirth, order/chaos)

5. Setting (temporal and spatial context)

Return as JSON with these categories as keys.

"""

response = llm\_client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are an expert folklorist who analyzes motifs and extracts structured information."},

{"role": "user", "content": prompt}

]

)

structured\_info = json.loads(response.choices[0].message.content)

return structured\_info

This structured information could be stored alongside your embeddings, allowing for more specific comparative analyses.

**3. Geographic Analysis Through Cultural Diffusion**

Instead of trying to place everything on a traditional map, consider representing geographic relationships through cultural diffusion patterns:

def analyze\_cultural\_diffusion(knowledge\_graph, motif\_id):

"""

Analyze the potential diffusion pattern of a motif through cultural connections.

"""

# Get all cultural references for this motif

cultural\_refs = [node for node in nx.neighbors(knowledge\_graph, motif\_id)

if knowledge\_graph.nodes[node]['type'] == 'cultural\_reference']

# Create a subgraph of just the cultural references and their connections

cultural\_subgraph = knowledge\_graph.subgraph(cultural\_refs)

# Find communities in this subgraph

communities = nx.community.greedy\_modularity\_communities(cultural\_subgraph)

# For each community, analyze its geographic context

diffusion\_patterns = []

for i, community in enumerate(communities):

community\_refs = list(community)

# Get geographic context embeddings for this community

geo\_embeddings = [knowledge\_graph.nodes[ref]['geo\_context']['geo\_context\_embedding']

for ref in community\_refs]

# Analyze the geographic characteristics of this community

diffusion\_patterns.append({

'community\_id': i,

'cultural\_references': community\_refs,

'geographic\_description': summarize\_geographic\_context(community\_refs),

'cohesion': calculate\_community\_cohesion(community, cultural\_subgraph)

})

return diffusion\_patterns

This approach acknowledges the impossibility of precise geographic mapping while still providing insights into cultural relationships and potential diffusion patterns.

**Implementation Strategy**

Given the complexity of these approaches, I recommend an incremental implementation strategy:

* Start with enhancing your embeddings:
  + Experiment with contextual embeddings that incorporate cultural information
  + Explore contrastive learning to better differentiate concept-level similarities
* Build the cultural knowledge graph:
  + Begin with a simplified version focusing on motifs and their reference terms
  + Gradually incorporate more complex relationships as your understanding develops
* Develop the geographic context representations:
  + Create rich textual descriptions of geographic contexts for reference terms
  + Generate embeddings for these descriptions to enable similarity comparisons
* Implement structured motif analysis:
  + Use LLMs to extract structured information from a sample of motifs
  + Compare the results to your embedding-based approach to evaluate effectiveness

Would this approach help address some of the challenges you're facing with both conceptual similarity detection and geographic normalization? I'd be happy to elaborate on any specific aspect that you find most promising or relevant to your immediate research needs.

**PROMPT**

please clarify about cultural context: is that determined solely by the ref\_term, e.g. Hebrew? If so, why would I develop a similarity measure, when all motifs identified as 'Hebrew' are tagged as such? Or would I be asking an LLM model for text explaining the geographic context, e.g given the term 'Hebrew' or "Gypsy" and an embedding for that - then using two embedding values in comparisons?

You've raised an excellent question about the cultural context approach. Let me clarify this concept more thoroughly, as it's a nuanced part of the proposed methodology.

**Understanding Cultural Context in Folkloric Analysis**

The cultural context isn't determined solely by the reference term (like "Hebrew" or "Gypsy"). You're right that if we merely grouped motifs by their explicit tags, we wouldn't gain any new insights beyond what's already in your database. The approach I'm suggesting is more sophisticated and involves several layers of information and analysis.

*What Cultural Context Really Means*

When I talk about cultural context, I'm referring to a rich representation that includes:

* Historical placement: When and where a cultural group existed, including migrations and changes over time
* Cultural relationships: How this group interacted with other groups (trade, conflict, cultural exchange)
* Linguistic relationships: Related languages or dialects that might indicate shared narrative traditions
* Environmental contexts: Geographic features, climate, and resources that might influence folklore themes
* Religious/belief systems: Underlying worldviews that shape narrative elements

*How to Generate and Use This Cultural Context*

Let me walk through a concrete example to illustrate the process:

Step 1: Enriching Reference Terms

For a term like "Hebrew," we would create a rich context beyond just the label:

def enrich\_reference\_term(ref\_term):

"""Create an enriched representation of a cultural reference term."""

# Start with basic information from your database

basic\_info = get\_basic\_term\_info(ref\_term) # Your existing metadata

# Expand with scholarly knowledge (could be from an LLM or knowledge base)

scholarly\_context = get\_scholarly\_context(ref\_term)

enriched\_context = {

"term": ref\_term,

"type": basic\_info.get("type", "unknown"), # linguistic, ethnic, geographic, etc.

"temporal\_range": scholarly\_context.get("temporal\_range", "unknown"),

"geographic\_associations": [

{

"region": "Eastern Mediterranean",

"modern\_countries": ["Israel", "Palestinian Territories", "parts of Jordan"],

"confidence": "high"

},

{

"region": "Diaspora communities",

"modern\_countries": ["Spain", "Eastern Europe", "North Africa"],

"period": "70 CE - present",

"confidence": "medium"

}

],

"related\_cultures": [

{"culture": "Canaanite", "relationship": "geographic proximity", "influence\_direction": "bidirectional"},

{"culture": "Egyptian", "relationship": "historical interaction", "influence\_direction": "bidirectional"},

{"culture": "Babylonian", "relationship": "historical interaction", "influence\_direction": "received"}

],

"linguistic\_family": "Semitic",

"related\_languages": ["Aramaic", "Arabic", "Phoenician"],

"belief\_system": "Judaism",

"narrative\_traditions": ["Torah", "Talmudic literature", "Midrash"]

}

return enriched\_context

Step 2: Creating Multi-Modal Representations

This enriched context would be represented in multiple ways:

* Textual description: A narrative paragraph describing the cultural context, which could be embedded:

def generate\_cultural\_context\_description(enriched\_context):

"""Generate a textual description of cultural context for embedding."""

description = f"The term '{enriched\_context['term']}' refers to a {enriched\_context['type']} "

description += f"associated primarily with the {', '.join(enriched\_context['geographic\_associations'][0]['modern\_countries'])} region. "

description += f"This culture existed from {enriched\_context['temporal\_range']} and "

description += f"had significant interactions with {', '.join([r['culture'] for r in enriched\_context['related\_cultures']])}. "

description += f"Their folklore emerges from a {enriched\_context['belief\_system']} worldview "

description += f"and is preserved in traditions like {', '.join(enriched\_context['narrative\_traditions'])}."

return description

* Structured features: A vector of features representing different aspects of the cultural context:

def extract\_cultural\_feature\_vector(enriched\_context):

"""Extract a numerical feature vector from enriched cultural context."""

# Example features (in practice, you would have many more nuanced features)

geographic\_region\_encoding = encode\_region(enriched\_context['geographic\_associations'][0]['region'])

temporal\_encoding = encode\_time\_period(enriched\_context['temporal\_range'])

linguistic\_encoding = encode\_language\_family(enriched\_context['linguistic\_family'])

belief\_system\_encoding = encode\_belief\_system(enriched\_context['belief\_system'])

# Combine into a feature vector

feature\_vector = np.concatenate([

geographic\_region\_encoding,

temporal\_encoding,

linguistic\_encoding,

belief\_system\_encoding

])

return feature\_vector

* Embedded representation: An embedding of the textual description:

def get\_cultural\_context\_embedding(enriched\_context):

"""Get an embedding for the cultural context."""

description = generate\_cultural\_context\_description(enriched\_context)

embedding = get\_embedding(description)

return embedding

**Step 3: Using Cultural Context in Motif Analysis**

Now, when analyzing motifs, you would use this enriched cultural context in several ways:

* Enhanced motif representations:

def create\_context\_aware\_motif\_representation(motif, reference\_terms):

"""Create a motif representation that incorporates cultural context."""

# Get the basic motif embedding

motif\_embedding = motif['embedding']

# Get enriched contexts for all reference terms associated with this motif

contexts = [enrich\_reference\_term(term) for term in reference\_terms]

context\_embeddings = [get\_cultural\_context\_embedding(ctx) for ctx in contexts]

# Create a combined representation

# (You could use various approaches: concatenation, weighted combination, etc.)

combined\_representation = {

"motif\_id": motif['id'],

"text\_embedding": motif\_embedding,

"cultural\_contexts": contexts,

"context\_embeddings": context\_embeddings,

"combined\_embedding": combine\_embeddings(motif\_embedding, context\_embeddings)

}

return combined\_representation

* Cultural similarity analysis:

def calculate\_cultural\_similarity(context1, context2):

"""Calculate similarity between cultural contexts."""

# Multiple similarity measures

geographic\_sim = geographic\_similarity(

context1['geographic\_associations'],

context2['geographic\_associations']

)

temporal\_sim = temporal\_similarity(

context1['temporal\_range'],

context2['temporal\_range']

)

linguistic\_sim = linguistic\_similarity(

context1['linguistic\_family'],

context1['related\_languages'],

context2['linguistic\_family'],

context2['related\_languages']

)

belief\_sim = belief\_system\_similarity(

context1['belief\_system'],

context2['belief\_system']

)

# Weighted combination

overall\_similarity = (

0.3 \* geographic\_sim +

0.2 \* temporal\_sim +

0.3 \* linguistic\_sim +

0.2 \* belief\_sim

)

return overall\_similarity

*The Value This Adds to Your Analysis*

This approach offers several advantages over simply grouping motifs by their tagged culture:

* Discovery of unexpected connections: You might find that motifs from cultures tagged as "Hebrew" and "Babylonian" share more similarities than you'd expect based on their cultural proximity during certain historical periods.
* Nuanced cultural mapping: Rather than treating "Hebrew" as a monolithic category, you'd recognize that Hebrew folklore from different periods might have different characteristics and influences.
* Geographic analysis without precise boundaries: You could identify patterns of motif similarity that correlate with geographic proximity, linguistic relationships, or historical interactions without requiring precise geographic boundaries.
* Temporal dynamics: You could analyze how motifs evolve and spread over time by incorporating the temporal dimension of cultural contexts.

**Practical Implementation**

In practical terms, you would:

* Start with reference terms from your database ("Hebrew", "Gypsy", etc.)
* Enrich these terms with additional context (historical, geographical, linguistic)
* Generate embeddings for these enriched contexts
* Use these context embeddings alongside your motif embeddings in your analysis

The similarity measures would then operate on this richer representation of cultural contexts, not just on the simple reference term labels.

*Example of Insights This Could Yield*

* With this approach, you might discover patterns like:
* Certain creation myth motifs cluster together among cultures with animistic belief systems, regardless of their geographic distance
* Flood myths show greater similarity among cultures in river delta regions across different continents than among culturally related groups in different environmental contexts
* Trickster motifs exhibit different characteristics in nomadic versus settled cultures, regardless of their linguistic relationships

Does this clarification help you understand how the cultural context approach differs from simply grouping by reference terms? And do you see how it could add value to your analysis beyond what you're currently doing with direct embedding comparisons?

**PROMPT**

with respect to fine tuning, please outline steps I would take - building a training set of similar and dissimilar motifs, but then training a model? what model, e.g. using what services and resources from anthropic and/or hugging face?

**RESPONSE**

Fine-Tuning Embeddings for Folkloric Analysis

Let me outline a comprehensive approach to fine-tuning embeddings for your folkloric analysis. This process involves several stages, from creating a specialized training dataset to implementing and evaluating the fine-tuned model.

**1. Building a Training Dataset**

The first step is creating a high-quality training dataset that captures the nuances of folkloric similarities and differences.

Data Collection

Start by assembling pairs of motifs that are either conceptually similar or dissimilar:

def build\_training\_dataset():

# Connect to your database

conn = psycopg2.connect(\*\*DB\_PARAMS)

cursor = conn.cursor()

# Fetch all motifs

cursor.execute("SELECT motif\_id, motif\_text\_with\_references FROM folklore.embed\_text\_motifs")

motifs = cursor.fetchall()

training\_pairs = []

# For each motif, create positive and negative pairs

for i, (motif\_id, motif\_text) in enumerate(motifs):

# 1. Find positive pairs (similar motifs)

# Option A: Use expert annotations if available

cursor.execute("""

SELECT similar\_motif\_id FROM folklore.motif\_similarities

WHERE motif\_id = %s AND similarity\_score > 0.8

""", (motif\_id,))

similar\_motifs = cursor.fetchall()

# Option B: If no expert annotations, use thematic categorization

if not similar\_motifs:

cursor.execute("""

SELECT m2.motif\_id, m2.motif\_text\_with\_references

FROM folklore.embed\_text\_motifs m1

JOIN folklore.embed\_text\_motifs m2 ON

m1.motif\_category = m2.motif\_category AND

m1.motif\_id != m2.motif\_id

WHERE m1.motif\_id = %s

LIMIT 5

""", (motif\_id,))

similar\_motifs = cursor.fetchall()

# Add positive pairs to training data

for similar\_id in similar\_motifs:

cursor.execute("SELECT motif\_text\_with\_references FROM folklore.embed\_text\_motifs WHERE motif\_id = %s", (similar\_id[0],))

similar\_text = cursor.fetchone()[0]

training\_pairs.append({

"anchor": motif\_text,

"compare": similar\_text,

"label": 1 # Similar

})

# 2. Find negative pairs (dissimilar motifs)

# Sample motifs from different categories

cursor.execute("""

SELECT m2.motif\_id, m2.motif\_text\_with\_references

JOIN folklore.embed\_text\_motifs m2 ON

FROM folklore.embed\_text\_motifs m1

m1.motif\_category != m2.motif\_category AND

m1.motif\_id != m2.motif\_id

WHERE m1.motif\_id = %s

ORDER BY RANDOM()

LIMIT 5

""", (motif\_id,))

dissimilar\_motifs = cursor.fetchall()

# Add negative pairs to training data

for \_, dissimilar\_text in dissimilar\_motifs:

training\_pairs.append({

"anchor": motif\_text,

"compare": dissimilar\_text,

"label": 0 # Dissimilar

})

# Log progress

if i % 100 == 0:

print(f"Processed {i} motifs")

cursor.close()

conn.close()

# Save to file

with open("folklore\_training\_pairs.json", "w") as f:

json.dump(training\_pairs, f)

return training\_pairs

**Data Enrichment and Augmentation**

To improve training quality, enrich your dataset with additional contextual information:

def enrich\_training\_data(training\_pairs):

enriched\_pairs = []

for pair in training\_pairs:

# Extract categories, cultural contexts, and other metadata

anchor\_metadata = extract\_metadata(pair["anchor"])

compare\_metadata = extract\_metadata(pair["compare"])

# Create enriched training examples

enriched\_pairs.append({

"anchor\_text": pair["anchor"],

"anchor\_context": f"Category: {anchor\_metadata['category']}, Culture: {anchor\_metadata['culture']}",

"compare\_text": pair["compare"],

"compare\_context": f"Category: {compare\_metadata['category']}, Culture: {compare\_metadata['culture']}",

"similarity\_score": 1.0 if pair["label"] == 1 else 0.0,

"explanation": generate\_similarity\_explanation(pair["anchor"], pair["compare"], pair["label"])

})

return enriched\_pairs

**2. Fine-Tuning Approach Options**

You have several options for fine-tuning embeddings, depending on your resources and technical requirements.

**Option 1: Fine-Tune Using OpenAI's API**

OpenAI allows fine-tuning of their embedding models for specific domains:

import openai

import pandas as pd

def prepare\_openai\_fine\_tuning\_data(enriched\_pairs):

# Format data for OpenAI fine-tuning

fine\_tuning\_data = []

for pair in enriched\_pairs:

fine\_tuning\_data.append({

"input": pair["anchor\_text"],

"comparison": pair["compare\_text"],

"score": pair["similarity\_score"]

})

# Convert to JSONL format

df = pd.DataFrame(fine\_tuning\_data)

df.to\_json("openai\_fine\_tuning\_data.jsonl", orient="records", lines=True)

return "openai\_fine\_tuning\_data.jsonl"

def fine\_tune\_openai\_embedding\_model(file\_path):

# Upload file to OpenAI

upload\_response = openai.File.create(

file=open(file\_path, "rb"),

purpose="fine-tune"

)

file\_id = upload\_response.id

# Create fine-tuning job

fine\_tuning\_response = openai.FineTuningJob.create(

training\_file=file\_id,

model="text-embedding-3-small", # Base model

suffix="folklore-analysis" # Creates "text-embedding-3-small:folklore-analysis"

)

return fine\_tuning\_response.id

Note: This is a simplified example. OpenAI's actual fine-tuning API may have different parameters and requirements, so refer to their documentation for the most up-to-date information.

**Option 2: Fine-Tune Using Sentence-Transformers (Hugging Face)**

For more control over the process, you can fine-tune a Sentence-Transformers model from Hugging Face:

from sentence\_transformers import SentenceTransformer, InputExample, losses

from torch.utils.data import DataLoader

import torch

def fine\_tune\_sentence\_transformer(enriched\_pairs):

# Load base model

model = SentenceTransformer('all-mpnet-base-v2') # Good starting point

# Prepare training examples

train\_examples = []

for pair in enriched\_pairs:

train\_examples.append(InputExample(

texts=[pair["anchor\_text"], pair["compare\_text"]],

label=pair["similarity\_score"]

))

# Create data loader

train\_dataloader = DataLoader(train\_examples, shuffle=True, batch\_size=16)

# Use cosine similarity loss for training

train\_loss = losses.CosineSimilarityLoss(model)

# Train the model

model.fit(

train\_objectives=[(train\_dataloader, train\_loss)],

epochs=10,

warmup\_steps=100,

output\_path="folklore-embeddings-model"

)

return model

**Option 3: Develop a Custom Embedding Model with Anthropic**

While Anthropic doesn't currently offer a public fine-tuning API specifically for embeddings, you could use their Claude API to create a domain-specific mapping layer on top of general embeddings:

from openai import OpenAI

import anthropic

import numpy as np

def create\_anthropic\_embedding\_enhancer(enriched\_pairs):

# Initialize clients

openai\_client = OpenAI()

anthropic\_client = anthropic.Anthropic()

# Function to generate enhanced embeddings using Claude

def enhance\_embedding(text, base\_embedding):

# Use Claude to analyze folkloric content

prompt = f"""

Analyze this folkloric motif text and identify its key conceptual elements:

{text}

Extract the following aspects:

1. Primary actors/agents

2. Actions or events

3. Symbolic elements

4. Cultural significance

5. Universal themes

Format your response as a structured JSON with these categories.

"""

response = anthropic\_client.messages.create(

model="claude-3-opus-20240229",

max\_tokens=1000,

messages=[

{"role": "user", "content": prompt}

]

)

# Extract structured analysis

analysis = json.loads(response.content[0].text)

# Convert analysis to a textual representation

enhanced\_text = f"""

Actors: {', '.join(analysis['Primary actors/agents'])}

Actions: {', '.join(analysis['Actions or events'])}

Symbols: {', '.join(analysis['Symbolic elements'])}

Significance: {analysis['Cultural significance']}

Themes: {', '.join(analysis['Universal themes'])}

"""

# Get embedding of enhanced text

enhanced\_embedding\_response = openai\_client.embeddings.create(

input=enhanced\_text,

model="text-embedding-3-small"

)

enhanced\_embedding = enhanced\_embedding\_response.data[0].embedding

# Combine original and enhanced embeddings (with weighting)

combined\_embedding = np.concatenate([

np.array(base\_embedding) \* 0.7,

np.array(enhanced\_embedding) \* 0.3

])

# Normalize

combined\_embedding = combined\_embedding / np.linalg.norm(combined\_embedding)

return combined\_embedding.tolist()

# Build a mapping function

return enhance\_embedding

This approach uses Claude's understanding of folklore to create a more conceptually focused representation that can be combined with standard embeddings.

**3. Evaluation and Implementation**

Once you've fine-tuned your embeddings, it's crucial to evaluate their performance and implement them in your system.

*Evaluation*

def evaluate\_embeddings(model\_type, test\_pairs):

"""Evaluate embedding model performance on folklore-specific pairs."""

correct\_predictions = 0

total\_predictions = len(test\_pairs)

for pair in test\_pairs:

# Get embeddings

if model\_type == "openai":

anchor\_embedding = get\_openai\_embedding(pair["anchor\_text"], "your-fine-tuned-model")

compare\_embedding = get\_openai\_embedding(pair["compare\_text"], "your-fine-tuned-model")

elif model\_type == "sentence-transformer":

model = SentenceTransformer("folklore-embeddings-model")

anchor\_embedding = model.encode(pair["anchor\_text"])

compare\_embedding = model.encode(pair["compare\_text"])

elif model\_type == "anthropic-enhanced":

enhancer = create\_anthropic\_embedding\_enhancer([])

base\_anchor = get\_openai\_embedding(pair["anchor\_text"], "text-embedding-3-small")

base\_compare = get\_openai\_embedding(pair["compare\_text"], "text-embedding-3-small")

anchor\_embedding = enhancer(pair["anchor\_text"], base\_anchor)

compare\_embedding = enhancer(pair["compare\_text"], base\_compare)

# Calculate similarity

similarity = cosine\_similarity(anchor\_embedding, compare\_embedding)

# Determine prediction (thresholding)

predicted\_similar = similarity > 0.75

actual\_similar = pair["similarity\_score"] > 0.5

if predicted\_similar == actual\_similar:

correct\_predictions += 1

accuracy = correct\_predictions / total\_predictions

print(f"Accuracy for {model\_type}: {accuracy:.4f}")

return accuracy

**Implementation in Your Existing System**

Once you've selected the best approach, update your embedding generation script:

# Modified version of your existing script to use fine-tuned embeddings

import os

import asyncio

import psycopg2

from psycopg2.extras import execute\_values

from dotenv import load\_dotenv

from openai import AsyncOpenAI

import time

# Load environment variables

load\_dotenv()

# Set up AsyncOpenAI client

client = AsyncOpenAI(api\_key=os.getenv('OPENAI\_API\_KEY'))

# Database connection parameters

DB\_PARAMS = {

"host": "localhost",

"database": "staging",

"user": "postgres",

"port": 5435

}

# Use your fine-tuned model

openai\_model = 'text-embedding-3-small:folklore-analysis' # Your fine-tuned model

BATCH\_SIZE = 100 # Adjust based on your API rate limits and performance

async def get\_embeddings(texts):

try:

response = await client.embeddings.create(input=texts, model=openai\_model)

return [embedding.embedding for embedding in response.data]

except Exception as e:

print(f"Error in get\_embeddings: {e}")

return []

async def process\_batch(batch):

motif\_ids, texts = zip(\*batch)

embeddings = await get\_embeddings(texts)

return list(zip(motif\_ids, texts, embeddings))

async def process\_embeddings(batch\_size=BATCH\_SIZE, limit=None):

conn = psycopg2.connect(\*\*DB\_PARAMS)

cursor = conn.cursor()

try:

# Fetch unprocessed rows

query = """

SELECT motif\_id, motif\_text\_with\_references

FROM folklore.embed\_text\_motifs

WHERE motif\_id NOT IN (SELECT motif\_id FROM folklore.motif\_embeddings\_fine\_tuned)

"""

if limit:

query += f" LIMIT {limit}"

cursor.execute(query)

rows = cursor.fetchall()

print(f"Total rows to process: {len(rows)}")

for i in range(0, len(rows), batch\_size):

batch = rows[i:i + batch\_size]

results = await process\_batch(batch)

# Insert results into the database

insert\_query = """

INSERT INTO folklore.motif\_embeddings\_fine\_tuned (motif\_id, motif\_text, embedding)

VALUES %s

"""

execute\_values(cursor, insert\_query, results)

conn.commit()

print(f"Processed batch {i // batch\_size + 1}, total rows: {i + len(batch)}")

print("Processing complete.")

except Exception as e:

print(f"An error occurred: {e}")

finally:

cursor.close()

conn.close()

**4. Computational Resources and Costs**

Let's discuss the resources needed for each approach:

*OpenAI Fine-Tuning*

Cost: The fine-tuning process typically costs in the range of $0.03-$0.10 per training record, depending on record length. For a dataset of 5,000 pairs, this could range from $150-$500.

Time: Fine-tuning can take 1-4 hours depending on dataset size.

Usage costs: After fine-tuning, embedding generation costs are similar to the base model, around $0.0001 per 1K tokens.

*Sentence-Transformers (Hugging Face)*

Computing resources: Requires a GPU for efficient training. An NVIDIA T4 or V100 would be sufficient.

Cloud costs: A suitable GPU instance on AWS or Google Cloud would cost $0.50-$2.00 per hour. Training might take 2-8 hours depending on dataset size.

Deployment: Once trained, the model can be deployed on a CPU machine for inference, costing $0.05-$0.10 per hour for a moderate load.

Anthropic-Enhanced Approach

API costs: Each analysis with Claude would cost approximately $0.01-$0.03 per motif, depending on length. You could reduce costs by processing motifs in batch and caching results.

Computational resources: Minimal local computation needed; most processing happens via the API.

Total cost: For analyzing 10,000 motifs, expect costs of $100-$300.

5. Practical Recommendations

Based on your project's needs, here are my recommendations:

For Immediate Results with Modest Budget

Start with the Anthropic-Enhanced Approach:

* Use Claude to perform conceptual analysis of your motifs
* Combine these analyses with your existing embeddings

This provides a conceptual layer without the need for extensive training

**Implementation steps:**

# Process each motif with Claude

def process\_motifs\_with\_claude():

conn = psycopg2.connect(\*\*DB\_PARAMS)

cursor = conn.cursor()

cursor.execute("SELECT motif\_id, motif\_text\_with\_references FROM folklore.embed\_text\_motifs")

motifs = cursor.fetchall()

client = anthropic.Anthropic()

for motif\_id, motif\_text in motifs:

# Check if already processed

cursor.execute("SELECT 1 FROM folklore.motif\_conceptual\_analysis WHERE motif\_id = %s", (motif\_id,))

if cursor.fetchone():

continue

# Get Claude analysis

prompt = f"""

Analyze this folkloric motif and identify its key conceptual elements:

{motif\_text}

Extract and return ONLY a JSON object with these categories:

1. actors: [list of primary characters or entities]

2. actions: [list of key actions or events]

3. symbols: [list of symbolic elements]

4. themes: [list of universal themes]

5. cultural\_context: brief description of cultural context

"""

try:

response = client.messages.create(

model="claude-3-sonnet-20240229", # More affordable than Opus

max\_tokens=1000,

messages=[

{"role": "user", "content": prompt}

]

)

analysis = json.loads(response.content[0].text)

# Store in database

cursor.execute("""

INSERT INTO folklore.motif\_conceptual\_analysis

(motif\_id, actors, actions, symbols, themes, cultural\_context)

VALUES (%s, %s, %s, %s, %s, %s)

""", (

motif\_id,

json.dumps(analysis.get('actors', [])),

json.dumps(analysis.get('actions', [])),

json.dumps(analysis.get('symbols', [])),

json.dumps(analysis.get('themes', [])),

analysis.get('cultural\_context', '')

))

conn.commit()

except Exception as e:

print(f"Error processing motif {motif\_id}: {e}")

# Pause to respect API rate limits

time.sleep(0.5)

**For Long-Term Investment with Larger Budget**

Proceed with OpenAI Fine-Tuning:

* Create a high-quality training dataset using expert annotations
* Fine-tune the embedding model specifically for folklore analysis

This provides the best performance but requires more upfront investment

Implementation workflow:

* Start with 1,000-2,000 carefully curated training pairs
* Perform initial fine-tuning and evaluation
* Expand the training set based on performance analysis
* Fine-tune the final model with the complete dataset

**Conclusion**

Fine-tuning embeddings for folkloric analysis offers significant benefits for your GLOS project. By creating representations that better capture the conceptual relationships in folkloric texts, you'll be able to identify meaningful patterns across cultural boundaries.

The Anthropic-enhanced approach provides a good balance of cost and effectiveness for initial exploration. As your project matures, investing in a fully fine-tuned model through OpenAI or Sentence-Transformers would provide the most robust solution for long-term research.