Session 03 — Qualitative Coding with LLMs

Session goal

By the end of this session, you should be able to:

- apply **embeddings** to qualitative coding workflows,
- filter transcript chunks by **relevance** to a research question,
- classify chunks using a **pre-defined codebook** (theme list),
- understand when to use **LLM-based coding** vs. embeddings,
- explore **inductive patterns** through clustering.

> This session applies the foundational concepts from Session 02 to real qualitative research workflows.
>
> **Important:** This is a teaching workflow. In real projects, you should agree on analytic choices with your PI / research team.

Key concepts

Qualitative coding with embeddings

Traditional qualitative coding is time-intensive:

- Read through transcripts
- Identify relevant passages
- Apply codes from a codebook
- Track inter-rater reliability

With embeddings, you can:

- Automatically identify relevant passages
- Classify text by semantic similarity to theme definitions
- Scale to larger datasets
- Maintain consistency and reproducibility

Key insight: Embeddings don't replace human judgment—they augment and scale it.

The qualitative coding workflow

```
Interview transcript
↓
Parse speakers and their statements
↓
Group responses by moderator questions
↓
Generate embeddings for each chunk (question + responses)
```

```
↓
[Optional] Filter by relevance to research question
↓
Compare chunks to theme definitions
↓
Assign codes based on similarity scores
↓
Review and validate results
```

Relevance filtering

Problem: Interview transcripts cover many topics. You may only care about specific research questions.

Solution: Embed your research question and compare it to each chunk. Keep only chunks above a similarity threshold.

Example:

```
Research question: "How do participants seek help when facing challenges?"
↓
Embed question → [0.123, -0.456, 0.789, ...]
↓
Compare to all chunks
↓
Keep chunks with similarity > 0.4
```

Benefits:

- Reduces noise in subsequent analysis
- Focuses coding effort on relevant content
- Makes theme classification more accurate

Theme classification

Goal: Assign codes from a predefined codebook to relevant chunks.

How it works:

1. Define themes as text descriptions (your codebook)
2. Generate embeddings for each theme definition
3. For each chunk, compute similarity to all themes
4. Assign based on your classification rule

Classification strategies:

Strategy	When to use	How it works
Single-label (argmax)	Each chunk has one primary theme	Assign the theme with highest similarity
Multi-label (threshold)	Chunks can have multiple themes	Assign all themes above a threshold (e.g., 0.5)
Continuous scores	Themes are dimensions, not categories	Use similarity scores as covariates in analysis
Exploratory clustering	No predefined themes	Group similar chunks and review clusters

---

When to use LLM coding vs. embeddings

Approach	Best for	Pros	Cons
Embeddings	Scale, consistency, predefined codes	Fast, reproducible, cheap	May miss nuance
LLM coding	Nuanced judgment, binary decisions	Flexible, can explain reasoning	Slower, more expensive
Hybrid	Complex coding with scale needs	Combines strengths	Requires more setup

Rule of thumb:

- Use embeddings for **classification** (which theme?)
- Use LLMs for **judgment** (does this mention X? what's the tone?)

Guided activities

Preparation

Make sure you have:

- create environment (`just venv`)
- `.env` with `OPENAI_API_KEY`
- the repo open in VS Code
- completed Session 02 (embeddings basics)
- activate environment: `.venv/Scripts/activate.ps1`

Activity A — Create embeddings for transcript

Optional first step: If working with transcripts in languages other than English and you want to translate them, run:

```
python examples/01_translate_transcript.py
```

> **Note:** You can also work directly in the original language (e.g., code in Spanish without translating). Embeddings work in multiple languages.

Then create embeddings:

```
python examples/02_create_embeddings.py
```

What this script does:

1. Loads a sample transcript (Spanish focus group interview)
2. **Parses speakers**: Identifies each speaker (MODERADOR, FACILITADOR 1, etc.) and what they said
3. **Groups by moderator questions**: Creates chunks where each chunk contains:
   - The moderator's question
   - All participant responses to that question
4. Generates embeddings for each chunk
5. Saves results to `outputs/01_chunks_with_embeddings.csv`

Chunking strategy for focus groups:

Instead of splitting by paragraphs or fixed character counts, this script uses a **context-aware chunking** approach:

- Each moderator question starts a new chunk
- All participant responses following that question are grouped together
- This preserves the question-response context for better semantic analysis

Example chunk:

```
MODERADOR: What challenges did you face implementing the program?

FACILITADOR 1: The main challenge was coordinating schedules with families...
FACILITADOR 2: I found it difficult to adapt materials for different age
groups...
FACILITADOR 3: Time constraints were our biggest issue...
```

What to observe:

- How many chunks were created? (One per moderator question)
- What does the embedding column look like?
- How large is each embedding vector? (1536 dimensions for text-embedding-3-small)

What's happening:

- The script parses the transcript structure (speaker labels with colons)

- Groups participant responses under each moderator question
- Each combined chunk becomes a point in semantic space
- Similar question-response patterns will have nearby vectors
- This is a one-time process—embeddings are stored for reuse

Key insight: Context-aware chunking (by moderator questions) is more appropriate for focus group data than arbitrary paragraph splits. This preserves the conversational structure and improves downstream coding accuracy.

Activity B — Relevance filtering (question-focused approach)
Run:

```
python examples/03_relevance_filtering.py
```

What this script does:

1. Loads the chunks with embeddings from Activity A
2. Embeds a research question: *"What helped facilitators integrate Bloom with Love into existing family services?"*
3. Computes relevance score (similarity) for each chunk
4. Shows top relevant chunks and filters by threshold

When to use this approach:

Use relevance filtering when you want to **answer a specific research question**. This narrows your dataset to only chunks semantically related to your question, filtering out irrelevant content.

What to observe:

- Which chunks score highest for relevance?
- Are there irrelevant chunks being filtered out?
- What happens if you change the threshold?

What's happening:

```
Research question embedding: [0.12, -0.45, ...]
↓
Chunk 1 similarity: 0.65 ✅ Relevant
Chunk 2 similarity: 0.23 ❌ Filtered out
Chunk 3 similarity: 0.71 ✅ Relevant
...
```

Key insight: This is semantic search applied to qualitative coding—you're finding meaning matches, not keyword matches. Use this when you have a focused research question and want to filter your data before deeper analysis.

Try this: Modify the research question and see how the relevant chunks change.

Activity C — Theme classification with embeddings (deductive coding)
Run:

```
python examples/04_theme_classification_embeddings.py
```

What this script does:

1. Loads **all chunks** from Activity A (not filtered by research question)
2. Loads a predefined theme list (`data/themes/help_themes.json`) — your codebook
3. Embeds each theme definition
4. For each chunk, computes similarity to all themes
5. Assigns the best-matching theme (argmax strategy)
6. Saves results to CSV and generates an interactive HTML report
7. Shows examples of coded chunks per theme in console

Outputs generated:

- **CSV file**: `outputs/03_theme_classification.csv` — Full data with all similarity scores and theme assignments
- **HTML report**: `outputs/03_theme_classification_report.html` — Interactive visualization:
  - Summary cards showing chunk count and average score per theme
  - Expandable/collapsible sections for each theme
  - Top 10 examples per theme with similarity scores
  - Easy navigation and visual layout
- **Console output**: Enhanced statistics and top 3 examples per theme

To view the HTML report: Open `outputs/03_theme_classification_report.html` in your web browser (double-click the file or right-click → Open with → Browser)

When to use this approach:

Use theme classification when you want **traditional deductive coding**: you have a predefined codebook (theme dictionary) and want to classify all your data according to those codes. This is similar to manual coding with a predefined set of codes, but automated using semantic similarity.

Key difference from Activity B:

- **Activity B (relevance filtering):** Filters data by ONE research question → narrows dataset
- **Activity C (theme classification):** Applies MULTIPLE codes to all data → full codebook classification

What to observe:

- Do the assigned themes make sense?
- Are there themes that overlap too much?
- How confident are the assignments (similarity scores)?
- Which themes have the most/fewest chunks?
- Are the top-scoring examples truly representative of each theme?

What's happening:

```
Themes:
  - "Seeking help from family and friends"
  - "Professional help (doctors, counselors)"
  - "Community resources and organizations"

Chunk: "I talked to my sister and she gave me advice..."
↓
Similarity to Theme 1: 0.78 ← Assigned ✅
Similarity to Theme 2: 0.34
Similarity to Theme 3: 0.29
```

Key insight: Theme definitions matter! More specific, distinct definitions lead to better classification.

Discussion questions:

- Which chunks were hard to classify?
- Should we use multi-label classification instead?
- Are there missing themes in our codebook?

Activity D — Extract themes with LLM (inductive coding, optional)
Run:

```
python examples/05_extract_themes_llm.py
```

What this script does:

1. Loads the full transcript (translated English if available, or Spanish original)
2. Sends the entire transcript to the LLM without a specific research question
3. Asks the LLM to identify 8-15 recurring themes across all discussions
4. For each theme, generates:
   - Clear, concise theme name
   - Detailed definition (1-2 sentences)
   - Key examples or quotes that illustrate the theme
5. Saves results to `outputs/04_extracted_themes.txt`

When to use this approach:

Use LLM theme extraction when you **don't have a predefined codebook** and want to perform **inductive coding**. This is exploratory analysis where themes emerge from the data rather than being defined in advance.

Key difference from Activity C:

- **Activity C (deductive):** You define themes first → classify chunks by those themes

- **Activity D (inductive):** LLM reads all data → generates/discovers themes from patterns

What to observe:

- How do LLM-generated themes compare to your predefined ones (from Activity C)?
- Are the themes specific enough or too broad?
- Do the themes capture the breadth of discussion in the transcript?
- Which approach would you prefer for your research: predefined or LLM-extracted themes?

What's happening:

- The LLM reads through the entire transcript and identifies patterns
- It generates theme names and definitions based on what participants discussed
- This can serve as a starting point for codebook development
- You would typically review, refine, and validate these themes with your research team

Use case: When you don't have a predefined codebook and need exploratory theme identification.

Key insight: LLMs can help with inductive coding, but themes still need researcher validation.

Activity E — Non-verbal cue coding (optional)
Run:

```
python examples/06_nonverbal_coding_llm.py
```

What this script does:

1. Loads **all chunks** from Activity A (entire transcript)
2. For each chunk, asks the LLM to detect non-verbal cues (laughter, pauses, tone changes, etc.)
3. Extracts structured codes: presence of cues (YES/NO) and type of cue
4. Generates an interactive HTML report and CSV with results
5. Saves to `outputs/05_nonverbal_coding.csv` and `outputs/05_nonverbal_coding_report.html`

Outputs generated:

- **CSV file**: `outputs/05_nonverbal_coding.csv` — All chunks with non-verbal cue annotations
- **HTML report**: `outputs/05_nonverbal_coding_report.html` — Interactive visualization:
  - Summary statistics (total chunks, chunks with cues, percentage)
  - Expandable sections grouped by cue type (Laughter, Pauses, Confusion, etc.)
  - Full context for each chunk containing non-verbal signals
  - Easy navigation with visual layout
- **Console output**: Progress updates, summary statistics, and examples

To view the HTML report: Open `outputs/05_nonverbal_coding_report.html` in your web browser

What to observe:

- What kinds of non-verbal information does the LLM identify?
- How frequent are non-verbal cues in the transcript?
- Are certain types of cues more common in specific discussion topics?
- Could embeddings do this? Why or why not?
- How reliable are these judgments?

What's happening:

- The LLM is looking for indicators beyond semantic content
- It codes emotional tone, engagement, hesitation, laughter, etc.
- This complements thematic coding by capturing affective and interactional dimensions
- The script processes all chunks (may take several minutes depending on transcript size)

Use case: When you need to code for affect, engagement, communication style, or group dynamics that go beyond what participants explicitly said.

Key insight: LLMs can code for meta-communicative features that embeddings can't capture —but this requires clear instructions and validation. Non-verbal cues can reveal emotional responses, group dynamics, and engagement levels that inform interpretation of thematic content.

Activity F — Inductive clustering (optional)
Run:

```
python examples/07_inductive_clustering.py
```

What this script does:

1. Loads **all chunks** from Activity A (entire transcript with embeddings)
2. Runs K-Means clustering algorithm to group similar chunks (creates 8 clusters)
3. Generates t-SNE 2D visualization showing cluster relationships
4. Saves results to CSV and PNG
5. Shows example chunks from each cluster

Outputs generated:

- **CSV file**: `outputs/06_clusters.csv` — All chunks with assigned cluster labels
- **PNG visualization**: `outputs/06_clusters_tsne.png` — 2D scatter plot of clusters using t-SNE dimensionality reduction
- **Console output**: Summary statistics, cluster sizes, and representative examples from each cluster

When to use this approach:

Use clustering for **exploratory inductive analysis** when you don't have predefined themes and want to discover natural groupings in your data. The algorithm finds chunks that are semantically similar and groups them together.

What to observe:

- Do the clusters reveal meaningful patterns?
- Are there unexpected groupings that suggest themes you hadn't considered?
- How would you label these clusters as themes?
- Do cluster sizes make sense? (Some topics may naturally be discussed more)
- Looking at the t-SNE plot, are clusters well-separated or overlapping?

What's happening:

```
All chunk embeddings (1,536 dimensions) → K-Means (k=8 clusters)
↓
Cluster 0: [40 chunks] Pattern about [identify from examples]
Cluster 1: [35 chunks] Pattern about [identify from examples]
Cluster 2: [28 chunks] Pattern about [identify from examples]
...
↓
t-SNE reduces to 2D for visualization → PNG plot
```

How to interpret clusters:

1. Read the example chunks from each cluster
2. Identify common themes or topics across chunks in that cluster
3. Give the cluster a descriptive label
4. These labels become your emergent codebook
5. Validate with research team

Use case: Exploratory analysis when you don't have predefined themes. Clustering is especially useful for identifying unexpected patterns or when building a codebook from scratch.

Key insight: Clustering is hypothesis-generating, not hypothesis-testing. The algorithm finds mathematical patterns in semantic space—you provide the qualitative interpretation. Always review clusters with domain expertise to ensure they're meaningful, not just mathematically coherent.

Complete workflow example

Here's how you might combine these tools in a real project:

Scenario: Analyzing 8 focus groups about facilitators' experiences integrating a family support program into existing services.

Step 1 — Prepare data:

- Transcribe focus group recordings
- Ensure speaker labels are clear (MODERADOR, FACILITADOR 1, etc.)
- Translate if needed → `examples/01_translate_transcript.py` (optional if working in Spanish/original language)

Step 2 — Generate embeddings:

- Run `examples/02_create_embeddings.py` on all focus group transcripts
- Chunks are created by moderator questions (preserving question-response context)
- Store embeddings for reuse → `outputs/01_chunks_with_embeddings.csv`

Step 3A — Question-focused approach (if you have a specific research question):

- Define your research question (e.g., "What helped facilitators integrate Bloom with Love?")
- Run `examples/03_relevance_filtering.py`
- Keep only relevant chunks → `outputs/02_relevant_chunks.csv`
- Use these filtered chunks for targeted analysis

Step 3B — OR comprehensive coding approach (if coding all discussions):

- Skip relevance filtering
- Proceed directly to theme classification with all chunks

Step 4 — Classify themes (deductive approach):

- Have a predefined codebook? → `examples/04_theme_classification_embeddings.py`
- Review interactive HTML report (`outputs/03_theme_classification_report.html`)
- Analyze distribution of chunks across themes
- Identify representative examples for each theme

Step 5 — OR discover themes (inductive approach):

- No predefined codebook? → `examples/05_extract_themes_llm.py`
- LLM generates 8-15 themes from the data
- Review extracted themes → `outputs/04_extracted_themes.txt`
- Refine theme definitions with research team
- Create your codebook from these emergent themes
- Then proceed to theme classification (Step 4)

Step 6 — Code non-verbal dimensions:

- Run `examples/06_nonverbal_coding_llm.py`
- Identify laughter, pauses, group dynamics → `outputs/05_nonverbal_coding.csv`
- Review HTML report for patterns by cue type
- Analyze how non-verbal cues relate to discussion topics
- Use to inform interpretation of thematic findings

Step 7 — Validate and refine:

- Review coded chunks in HTML reports
- Check if theme definitions need adjustment
- Look for themes that overlap too much or are too broad
- Re-run classification with updated codebook if needed
- Compare deductive codes (Activity C) with inductive findings (Activity D & F)

Step 8 — Synthesize findings:

- Export coded data from CSVs

- Cross-tabulate themes with participant characteristics
- Identify patterns across focus groups
- Use representative quotes from HTML reports
- Integrate non-verbal codes to interpret engagement and affect

Key decision points:

- **Research question-focused vs. comprehensive?** → Determines if you use relevance filtering (Activity B)
- **Deductive vs. inductive?** → Determines if you start with codebook (Activity C) or generate themes (Activity D/F)
- **Scale vs. nuance?** → Embeddings for classification, LLMs for complex judgments
- **Focus group specific:** Non-verbal codes capture group dynamics that individual interviews miss

---

Mental model: Choosing your coding approach

```
Do you have a predefined codebook?
├─ YES → Use embedding classification (Activity C)
│   └─ Want to explore patterns? → Also run clustering (Activity F)
└─ NO → Extract themes with LLM (Activity D) OR cluster embeddings (Activity
F)
    └─ Then: Validate themes and proceed with classification

Need to code non-verbal cues or meta-features?
└─ Use LLM coding (Activity E)

Have a specific research question?
└─ YES → Filter by relevance first (Activity B), then code
```

Common issues and solutions

"All chunks get assigned to the same theme" → Theme definitions might overlap too much. Make them more distinct and specific.

"Relevance filtering removes too many chunks" → Lower your threshold or rephrase your research question to be broader.

"Clusters don't make sense" → Try different values of `k` (number of clusters). Clustering is exploratory—not all datasets have clear natural groupings. Review example chunks to identify patterns.

"Embedding classification misses subtle cases" → This is expected. Use hybrid approach: embeddings for scale, complement with manual review of edge cases.

"Costs are adding up with LLM coding" → Use embeddings for classification (cheap), LLMs only for non-verbal coding or theme extraction. Avoid LLM coding for every chunk.

"HTML reports aren't opening" → Make sure you're opening the .html files in a web browser (Chrome, Firefox, Edge), not in a text editor.

"Non-verbal coding takes too long" → This is expected—it processes every chunk with an LLM call. For large datasets, consider sampling or running overnight.

What you've learned

- ✅ Apply embeddings to qualitative coding workflows
- ✅ Filter transcript chunks by research question relevance
- ✅ Classify chunks using theme similarity (codebook-based, deductive)
- ✅ Extract themes inductively with LLMs
- ✅ Discover patterns through unsupervised clustering
- ✅ Code non-verbal and meta-features with LLM judgment
- ✅ Understand trade-offs: embeddings (scale) vs. LLMs (nuance)
- ✅ Generate interactive HTML reports for exploration and validation

---

Discussion prompts

Use these to reflect on your coding choices:

- Which chunks were filtered out as irrelevant? Are we okay with that?
- Do the top chunks per theme look right in the HTML reports?
- Are there themes that overlap too much or are too broad?
- How do deductive codes (Activity C) compare with inductive themes (Activity D & F)?
- Do the clusters reveal unexpected patterns we should investigate?
- How frequent are non-verbal cues, and what do they tell us about group dynamics?
- Would multi-label classification fit our research question better?
- How would you validate these codes with traditional qualitative methods?
- What would inter-rater reliability look like for embedding-based codes?

Key takeaways

1. **Embeddings scale qualitative coding**: Semantic similarity enables automated classification across large datasets
2. **Relevance filtering focuses analysis**: Don't code everything—code what matters to your research question
3. **Theme definitions matter**: Clear, distinct definitions improve classification accuracy
4. **Deductive vs. inductive**: Choose based on whether you have a predefined codebook or need exploratory analysis
5. **Embeddings vs. LLMs**: Use embeddings for classification (fast, cheap), LLMs for judgment and meta-features (slower, nuanced)
6. **Interactive reports enhance exploration**: HTML visualizations make it easier to validate and interpret results

7. **Validation is still essential**: Tools augment, not replace, researcher judgment and domain expertise

---

Next steps and extensions

If you want to go further:

- **Store embeddings efficiently**: Use a vector database (Pinecone, Weaviate) for large datasets
- **Track provenance**: Add chunk IDs and source metadata for traceability
- **Measure agreement**: Compare embedding codes to human codes (Cohen's kappa)
- **Multi-label classification**: Implement threshold-based multi-label assignment
- **Hierarchical themes**: Build theme taxonomies with parent-child relationships
- **Longitudinal analysis**: Compare themes across waves or time periods

Tools to explore:

- **NVivo**: Import embedding-based codes for further qualitative analysis
- **Atlas.ti**: Export coded chunks with similarity scores
- **Dedoose**: Integrate with mixed-methods analysis workflows

---

Resources

- OpenAI Embeddings Best Practices: https://platform.openai.com/docs/guides/embeddings/use-cases
- Qualitative Coding with AI: https://www.anthropic.com/research/qualitative-coding
- Vector Similarity for Researchers: https://www.pinecone.io/learn/semantic-search/
- K-Means Clustering Explained: https://scikit-learn.org/stable/modules/clustering.html#k-means

---

Congratulations! You've completed the hands-on training sessions. You now have the tools to apply LLMs and embeddings to real qualitative research workflows.