Adam Lucek Research Notes — mycoSwarm Reference

Source: YouTube @AdamLucek | lucek.ai | GitHub: ALucek
Compiled: 2026-02-13
Purpose: Reference for mycoSwarm development phases + InsiderLLM articles

---

Video 1: Improving RAG Retrieval by 60% with Fine-Tuned Embeddings

URL: https://www.youtube.com/watch?v= (23K views, ~11 months ago)
Core thesis: Generic embedding models underperform on domain-specific retrieval. Fine-tuning on your own knowledge base yields 50-60% improvement on retrieval metrics.

Key Concepts

The Problem:

- Standard embedding models (OpenAI, nomic, etc.) are generalized — they handle broad data well but struggle with domain-specific or niche content
- In RAG, retrieval accuracy matters MORE than LLM quality — the LLM only knows what it's given
- Most RAG pipelines never fine-tune their embeddings

The Method:

1. Start with existing chunked knowledge base (your ChromaDB collections)
2. Generate synthetic question-chunk pairs using an LLM (GPT-4o in his case, gemma3 for us)
3. For each chunk, generate a question that can ONLY be answered by that chunk
4. Fine-tune embedding model on these pairs using sentence-transformers
5. Loss function: Multiple Negatives Ranking Loss — each anchor-positive pair treats all other pairs as negatives

Data Structure:

- Positive pairs: (question, corresponding_chunk) — simplest format
- Triplets: (anchor_question, positive_chunk, hard_negative_chunk) — harder to create but more effective
- Hard negatives = similar but critically different/unhelpful documents

Important Distinction:

- This does NOT train the model to generalize to unseen documents
- It optimizes retrieval on an EXISTING knowledge base with UNSEEN queries

- Perfect for mycoSwarm: stable corpus (sessions + docs), variable queries

Matryoshka Representation Learning (MRL):

- Train model to encode important info in earlier dimensions of the vector
- Allows truncation: 768-dim → 256-dim or 128-dim with minimal accuracy loss
- Earlier dimensions = coarse/high-level info, later dimensions = fine details
- Use case: store in low-dim for fast search, re-rank with full-dim for accuracy
- Lightweight to add during training — just wraps the base loss function

Base Model Used: nomic modern-BERT-embed-base (768-dim output)

Evaluation Metrics:

- **NDCG@10** (primary): Normalized Discounted Cumulative Gain — captures both presence AND ranking of relevant docs
- **Accuracy@K**: At least one relevant doc in top K results
- **Precision@K**: Fraction of relevant docs among top K
- **Recall@K**: Fraction of ALL relevant docs found in top K
- **MRR**: Mean Reciprocal Rank — position of first relevant result
- **MAP**: Mean Average Precision — comprehensive ranking quality

Results:

- Training time: ~6 minutes on Colab GPU
- NDCG@10 improvement: ~50-60% across dimensions
- 64-dim fine-tuned ≈ 768-dim base model performance (huge efficiency win)

His code/model: Published on HuggingFace (AdamLucek profile)

mycoSwarm Application

Phase: Domain-Adapted Embeddings

1. **Collect training data:** Log queries + retrieved chunks from `search_all()`. Over time, build a dataset of what users actually ask vs what gets retrieved.
2. **Generate synthetic pairs:** Use gemma3 on 3090 to generate questions for each chunk in both ChromaDB collections (session_memory + mycoswarm_docs). No cloud needed.
3. **Fine-tune nomic-embed-text:** Using sentence-transformers + MRL wrapper. ~6 min training on 3090.
4. **Swap embedding model:** Point mycoSwarm config to fine-tuned model.

5. **MRL truncation for M710Qs:** Use 256-dim or 128-dim vectors on constrained nodes, full 768-dim on 3090 for re-ranking.
6. **Retrain periodically:** As knowledge base grows (new sessions, new docs), retrain embeddings. Quick 6-min job.

Potential gains:

- 50-60% better retrieval accuracy on existing knowledge base
- Faster search on M710Qs with truncated dimensions
- Two-tier search: fast low-dim shortlist → accurate high-dim re-rank

---

Video 2: From Retrieval to Navigation — The New RAG Paradigm

URL: https://www.youtube.com/watch?v= (4.2K views, 4 days ago)
Core thesis: RAG is shifting from a retrieval problem ("get right context into window") to a navigation problem ("let model navigate context itself").

Key Concepts

Two Context Delivery Paradigms:

	Traditional RAG (Retrieval)	File-System / RLM (Navigation)
Question	How do we get the right context in?	How do we let model find context itself?
Control	Developer controls context selection	Model controls context selection
Mechanism	Embedding + vector similarity search	Shell commands, file navigation, code execution
Speed	Fast, narrow	Slower, deeper
Scaling	Optimize a search engine	Optimize model reasoning
Best for	Chatbots, quick Q&A, well-defined domains	Complex analysis, large codebases, long-horizon tasks

Context Window as Public Good (Anthropic's framing):

- Everything must fit in the context window: instructions, history, retrieved context, system prompts
- Optimization problem: minimize tokens consumed, maximize relevance of what's included
- This is "context engineering"

Context Rot:

- Performance degrades around ~100K tokens even in models claiming 1M+ support
- Needle-in-a-haystack testing proves models can't reliably find info in long contexts
- **Distractors** (similar-but-wrong content) make it dramatically worse
- Lost-in-the-middle: models attend better to beginning and end of context, miss the middle

Why context rot happens (hypotheses):

- Training data lacks examples of effective long-context usage
- RoPE positional encodings have long-range decay biasing toward nearby tokens
- Softmax attention disproportionately allocates to early tokens
- Still an open research question

Positional Encoding Evolution:

- Absolute positional embeddings → hard context length limit (position table up to N)
- RoPE (Rotary Positional Embeddings) → relative positions via rotation, generalizes beyond training length
- RoPE: earlier positions = smaller rotations, later = larger. Relative offset dominates after rotation cancellation
- Enabled scaling from 2048 tokens → 1M+ tokens

Recursive Language Models (RLM):

- Paper by DSPy/ColBERT researchers
- Model gets a Python REPL environment with the prompt stored as a variable
- Can programmatically inspect, chunk, search, and process the content
- **Key feature:** `llm_query()` function spawns sub-agent in fresh context window
- Sub-agent returns result, captured as variable for further processing
- All orchestrated via generated code
- Demonstrated 10M+ token context handling
- Can be improved further with SFT and RL training on effective context navigation

RLM Example (Pride and Prejudice):

- 748K character book loaded as variable
- Model splits into 90K character chunks
- Spawns 9 sub-agent calls, one per chunk

- Verification sub-calls for aggregation and dedup
- Final answer assembled from sub-agent results
- 12 total LLM calls to process one query across the full book

The Tradeoff:

- RAG = fast, narrow, developer-controlled
- RLM/Navigation = slow, deep, model-controlled
- Not either/or — use both depending on the query type

mycoSwarm Application

What mycoSwarm already does right:

- Session-as-RAG with topic splitting = structured navigable context (not raw dump)
- [S]/[D] citation labels = model knows where context came from
- Context pollution fix = prevents distractor accumulation (exactly the problem Chroma documented)
- Multi-node routing = architectural foundation for sub-agent delegation

What mycoSwarm could add:

Phase: Adaptive Context Strategy

1. **"We discussed" detection:** When user references past context, switch from quick single-pass RAG to deeper multi-step navigation mode
2. **Selective depth:** Simple queries → fast RAG retrieval. Complex/multi-topic queries → iterative exploration with follow-up retrievals
3. **Fresh context sub-calls:** Route complex retrieval to separate gemma3 call that summarizes before injecting. Prevents distractor pollution in main context.
4. **Progressive disclosure for RAG:** Instead of dumping all 5 session + 5 doc chunks at once, let model request more context from specific sources. "I need more from [S2]" → load deeper.
5. **Navigation mode for large documents:** When user indexes a large document, don't just chunk-and-embed. Also create a navigable summary/TOC that the model can use to request specific sections.

RLM-style features for later phases:

- Orchestrator node can spawn sub-tasks to worker nodes in fresh contexts
- Worker returns summary/answer → orchestrator integrates without context pollution
- Code-as-orchestration: let model write search/filter logic for complex queries

---

Video 3: Agent Skills — Yet Another Tool Standard?

URL: https://www.youtube.com/watch?v=7fbY8k9Mz3M (5.7K views, 1 month ago)
Core thesis: Agent Skills package domain-specific workflows, scripts, and resources into discoverable formats for file-system-based agents.

Key Concepts

What Agent Skills Are:

- Portable units of functionality: instructions + scripts + resources + assets in one directory
- Discoverable by file-system-based agents (Claude Code, Cursor, Codex, etc.)
- Inject domain-specific or workflow-specific knowledge without re-prompting every time
- Natural progression: Function Calling → MCP → Agent Skills

The Stack (complementary, not competing):

1. **Function Calling:** Model generates structured tool call + arguments. Client executes.
2. **MCP (Model Context Protocol):** Standardizes tool execution and formatting. Server handles tools.
3. **Agent Skills:** Bundles everything for a specific workflow. Model navigates and loads as needed.

File-System-Based Agents:

- Operate in terminal/shell environments
- Given broad tools: bash commands, Python execution, file navigation (ls, cat, grep, cd, touch)
- File system = pseudo memory system (create file = write memory, read file = recall)
- agents.md = readme for the agent (repo-level context and preferences)
- Models already have strong code understanding from pre-training

Progressive Disclosure:

- Only load context to model as necessary
- Skill name + description always in context (~100 tokens)
- Full skill body loaded only when model decides to use it
- Complex workflows split into sub-files, loaded on demand
- Scripts handle complex operations — model just calls them

Skill Structure:

```
skill-name/
├── SKILL.md          # Required: front matter + body
├── scripts/          # Optional: executable scripts
├── references/       # Optional: sub-markdown files
└── assets/           # Optional: templates, data files
```

SKILL.md Format:

```
markdown

---
name: skill-name          # Must match directory name
description: Brief description of what skill does  # ~100 tokens max
license: MIT
metadata:
  author: "example.com"
  version: "1.0.0"
---

# Skill Title

## Overview
Concise description. Assume model is capable.

## When to Use
- Scenario 1
- Scenario 2

## Scripts
How to run each script with examples.

## Notes / Limitations
Important caveats.
```

Best Practices:

- Concise — don't over-explain. Model is capable.
- Frontload important info (models may only `head` the file)
- Gerund naming: "Processing PDFs" not "Documents"
- Third person: "Process Excel files" not "You can process..."
- Keep under 500 lines per markdown file
- Metadata under 100 tokens

- Consistent terminology throughout
- Provide examples and templates (few-shot)
- Split complex workflows into sub-files

His Example: Apple Notes Skill

- 4 scripts: create, delete, list, read notes
- Skill body: overview, when to use, HTML reference table, limitations
- Scripts use Apple's scripting language wrapped in bash-callable commands
- Published as Claude Code plugin

Notable Example: HuggingFace LLM Trainer Skill

- Allows Claude Code to validate, monitor, and train language models via HF ecosystem
- One of the most comprehensive skills he's seen

mycoSwarm Application

Phase: mycoSwarm as Skills Package

The Agent Skills format is directly relevant for packaging mycoSwarm capabilities:

1. **mycoSwarm Skills for Claude Code / Codex:**
   - Package mycoSwarm's RAG, session memory, and node management as skills
   - Allow coding agents to interact with mycoSwarm's knowledge base
   - Scripts: `search_docs.py`, `search_sessions.py`, `add_document.py`, `check_nodes.py`
2. **Internal Skill System for mycoSwarm:**
   - Define domain-specific skills that mycoSwarm's own agents can discover and load
   - Example: "beekeeping-knowledge" skill with specialized retrieval prompts
   - Example: "crypto-analysis" skill with Ghost-Trend indicator knowledge
   - Progressive disclosure: skill descriptions in base context, full instructions loaded on demand
3. **Skill-Based Workflow Packaging:**
   - Package complex mycoSwarm workflows as skills: "index-new-document", "search-and-summarize", "multi-node-query"
   - Share with other mycoSwarm users via the upcoming plugin/skill marketplace concept
4. **InsiderLLM as Skill Knowledge:**
   - Package article-writing workflows as skills for Claude Code

- Templates, style guides, SEO optimization steps all as discoverable skill resources

---

Combined Action Items for PLAN.md

Phase 23: Enhanced Retrieval (from Videos 1 + 2)

- [ ] **Synthetic training data generation:** Script to generate question-chunk pairs from ChromaDB collections using gemma3
- [ ] **Fine-tune nomic-embed-text:** sentence-transformers + MRL wrapper on 3090
- [ ] **MRL truncation support:** Allow configurable embedding dimensions per node (128 for M710Q, 768 for 3090)
- [ ] **Retrieval metrics dashboard:** Add NDCG@10, precision, recall tracking to dashboard
- [ ] **A/B comparison tool:** Compare base vs fine-tuned embedding retrieval on same queries

Phase 24: Adaptive Context Strategy (from Video 2)

- [ ] **"We discussed" detection:** Boost session_memory weighting when user references past conversations
- [ ] **Query complexity classifier:** Simple queries → single-pass RAG. Complex → multi-step navigation
- [ ] **Fresh context sub-calls:** Route complex retrieval to separate LLM call for summarization before injection
- [ ] **Progressive RAG disclosure:** Let model request deeper context from specific [S] or [D] sources
- [ ] **Context budget tracking:** Monitor token usage per turn, warn when approaching rot threshold (~100K)

Phase 25: Skills System (from Video 3)

- [ ] **Package mycoSwarm as Claude Code skill:** SKILL.md + scripts for external agent interaction
- [ ] **Internal skill discovery:** mycoSwarm agents discover and load domain-specific skills from a skills directory
- [ ] **Progressive skill loading:** Skill descriptions in base prompt, full body loaded on-demand
- [ ] **Skill authoring guide:** Documentation for users to create their own mycoSwarm skills

InsiderLLM Article Ideas (from all three videos)

1. **"Fine-Tune Your Embeddings: 60% Better RAG on a Budget GPU"** — Adam's method applied to local hardware
2. **"From Retrieval to Navigation: Why Your RAG Pipeline Is About to Change"** — the paradigm shift
3. **"Context Rot: Why 1M Token Windows Don't Mean What You Think"** — Chroma's findings + practical implications
4. **"Agent Skills: Package Your AI Workflows Like a Pro"** — practical guide for local AI users
5. **"Recursive Language Models: Teaching AI to Read Entire Books"** — RLM paper explained for practitioners

Key References

- Adam Lucek: lucek.ai | github.com/ALucek | huggingface.co/AdamLucek
- Philip Schmid blog: Fine-tuning embedding models for RAG (referenced in Video 1)
- Chroma technical report on context rot (referenced in Video 2)
- RLM paper: Recursive Language Models (DSPy/ColBERT researchers) (referenced in Video 2)
- Anthropic Agent Skills docs + skill authoring best practices (referenced in Video 3)
- HuggingFace hf-llm-trainer skill (referenced in Video 3)
- Sentence Transformers: sentence-transformers package (used in Video 1)
- Matryoshka Representation Learning paper (referenced in Video 1)