Domain Specific AI

SD 18

Jacob Duba, Conor O'Shea, Carter Cutsforth, Diego Perez, and Keenan Jacobs

Improving Natural Language Code Search

Domain Specific AI **Problem**: Improve performance on niche topics

Retrieval Augmented Generation for grounding

Improve for High Performance Computing

Improve for any proprietary codebase

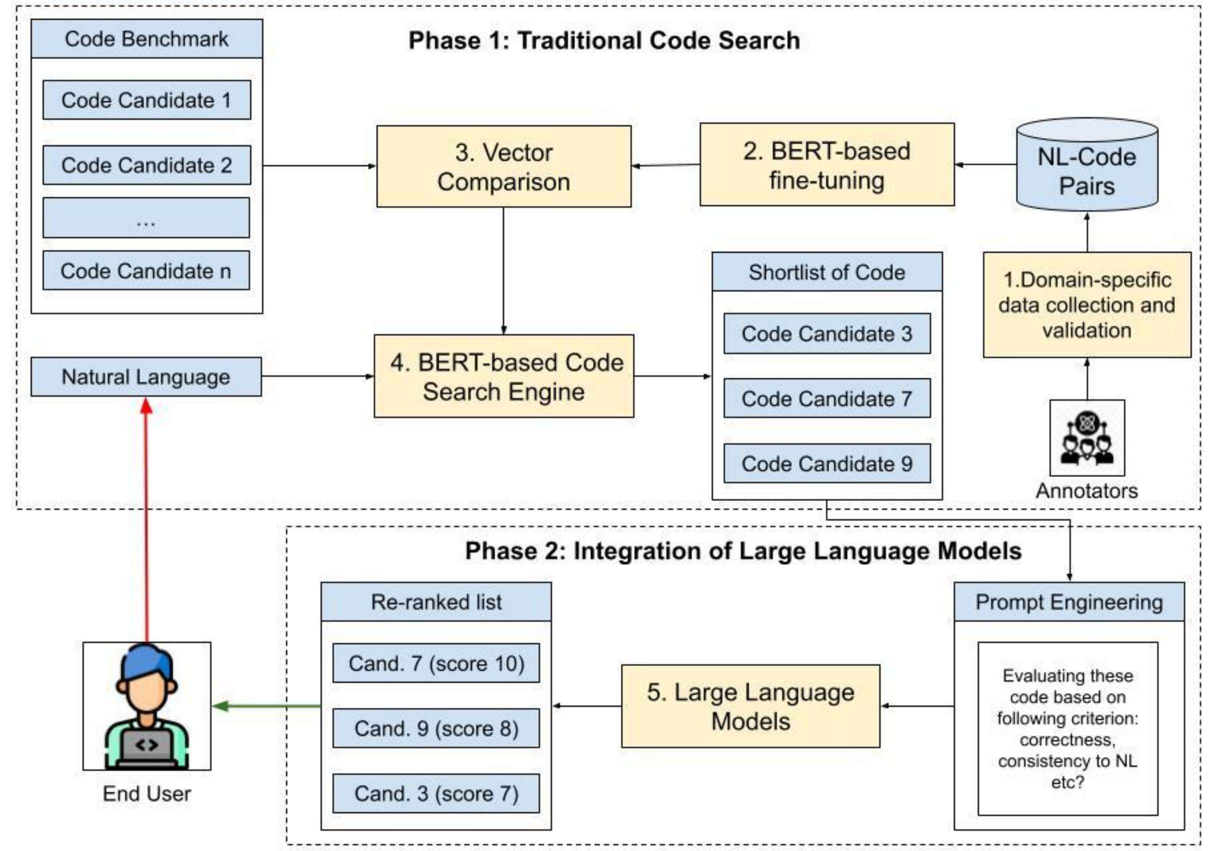
Improving Natural Language Code Search

Keyword search: solved

Natural Language (NL) search: unsolved

Dr. Phan's research: improve NL code search by having LM rerank top 10 vector searches

Proof of Concept



Proof of Concept

Challenge: starting the project

- How to break big problem into small problems
- Distributing small problems

Proof of Concept

Solution:

- Jacob - Scaffolded project
- Carter - Loading dataset
- Diego - Process and Store
- Conor - Read and Search
- Keenan - Rerank top 10 with LM

Proof of Concept

Achieving Deliverables

Challenge: Project couldn’t communicate research

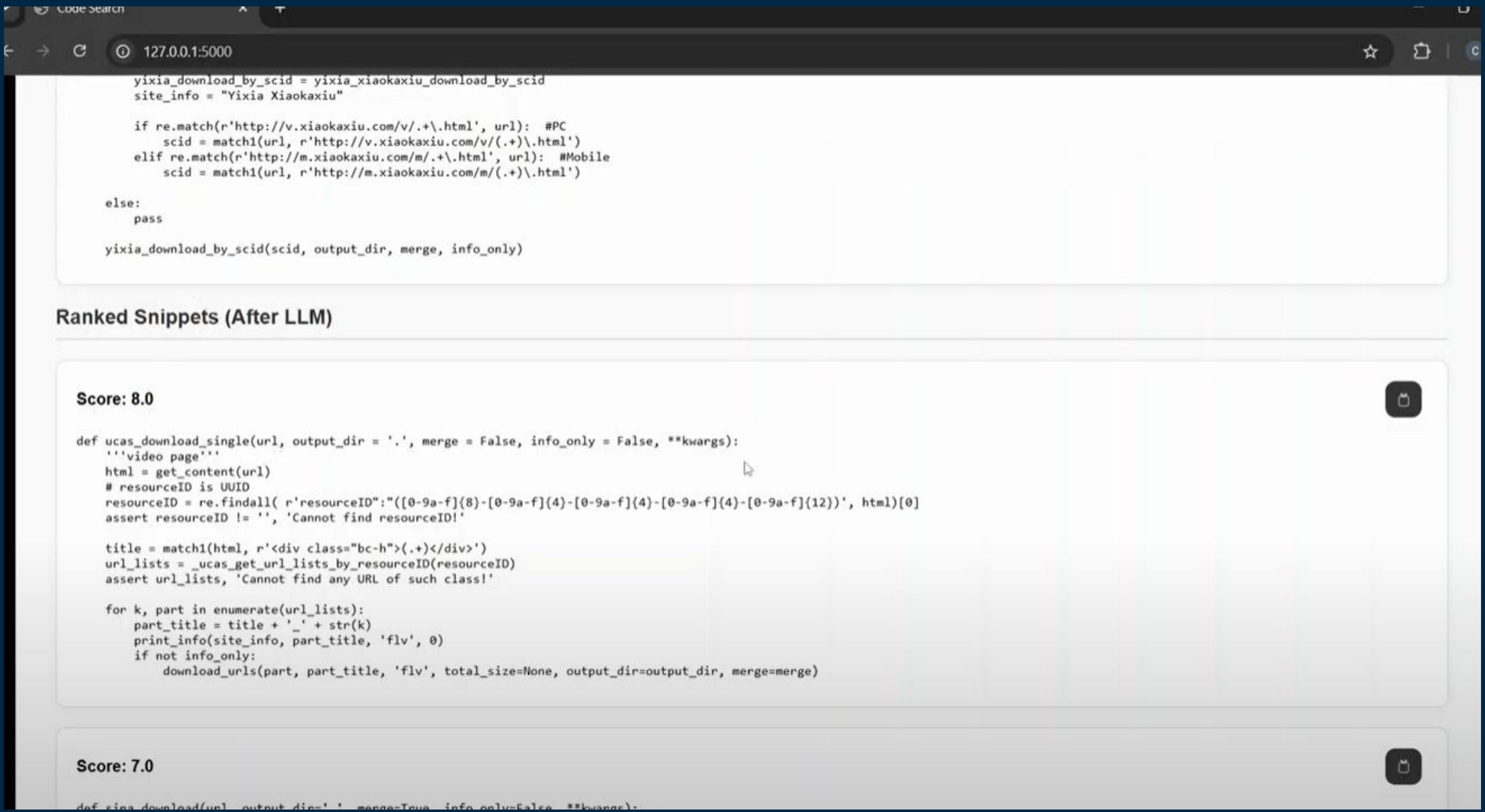
- Demo was unusable
  - Too slow to handle 50+ embeddings
  - No UI
- No benchmarks

Achieving Deliverables

Solution:

- Diego - Store in SQLite, process on GPU
- Keenan - Develop interface
- Carter - Set up infrastructure for LM
- Conor - Write/run benchmarks

Achieving Deliverables




```
        yixia_download_by_scid = yixia_xiaokaxiu_download_by_scid
        site_info = "Yixia Xiaokaxiu"

        if re.match(r'http://v.xiaokaxiu.com/v/.+\.html', url):  #PC
            scid = match1(url, r'http://v.xiaokaxiu.com/v/(.+)\.html')
        elif re.match(r'http://m.xiaokaxiu.com/m/.+\.html', url):  #Mobile
            scid = match1(url, r'http://m.xiaokaxiu.com/m/(.+)\.html')

    else:
        pass

    yixia_download_by_scid(scid, output_dir, merge, info_only)
```

Ranked Snippets (After LLM)

Score: 8.0

```
def ucas_download_single(url, output_dir = '.', merge = False, info_only = False, **kwargs):
    '''video page'''
    html = get_content(url)
    # resourceID is UUID
    resourceID = re.findall( r'resourceID":"([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})', html)[0]
    assert resourceID != '', 'Cannot find resourceID!'

    title = match1(html, r'<div class="bc-h">(.+)</div>')
    url_lists = _ucas_get_url_lists_by_resourceID(resourceID)
    assert url_lists, 'Cannot find any URL of such class!'

    for k, part in enumerate(url_lists):
        part_title = title + '_' + str(k)
        print_info(site_info, part_title, 'flv', 0)
        if not info_only:
            download_urls(part, part_title, 'flv', total_size=None, output_dir=output_dir, merge=merge)
```

Score: 7.0

CodeSearchNet Connection

Design Challenge: Loading Datasets

- Needed a dataset to search using natural language
- Project requires code and comments

CodeSearchNet Connection –Minor

Design Solution: Hugging Face CodeSearchNet

- 2 million datasets
- Match Project needs

Hugging Face

LLM Integration

Design Challenge: Expensive Language Models

- LLM Calls are Expensive
- Network Drops
- Privacy

LLM Integration -Major

Design Solution : Locally Hosted Model

- Speeds up query times
- Project fully hosted locally
- Practically free

LLM Integration

Ollama

- Open Source
- Easy to integrate
- Multi-model testing possible

Challenges: model limitations, compute intensive

LLM Integration

Major Accomplishment : Ollama querying

- Determining Model to use
- Reworking of existing model search
- 33% faster query times

Tools Used:
Python, Ollama, OpenRouter API

Benchmarking

Design Challenge: How to judge LLM quality?

- Accuracy of reranking top 10
- Dr. Phan's research

Implemented with: Python, UniXcoder, Ollama

Benchmarking

Solution: Benchmarking

- Adjust existing data format to handle LLM
- Take UniXcoder top 10, judge LLM’s reranking
- Need Ollama for API calls

Major accomplishment

Benchmarking

Model	Clone Detection				Code Search		
	POJ-104	BigCloneBench			CosQA	AdvTest	CSN
	MAP@R	Recall	Precision	F1-score	MRR		
RoBERTa	76.67	**95.1**	87.8	91.3	60.3	18.3	61.7
CodeBERT	82.67	94.7	93.4	94.1	65.7	27.2	69.3
GraphCodeBERT	85.16	94.8	95.2	95.0	68.4	35.2	71.3
SynCoBERT	88.24	-	-	-	-	38.3	74.0
PLBART	86.27	94.8	92.5	93.6	65.0	34.7	68.5
CodeT5-base	88.65	94.8	94.7	95.0	67.8	39.3	71.5
UniXcoder	**90.52**	92.9	**97.6**	**95.2**	**70.1**	**41.3**	**74.4**
-w/o contras	87.83	94.9	94.9	94.9	69.2	40.8	73.6
-w/o cross-gen	90.51	94.8	95.6	95.2	69.4	40.1	74.0
-w/o comment	87.05	93.6	96.2	94.9	67.9	40.7	72.6
-w/o AST	88.74	92.9	97.2	95.0	68.7	40.3	74.2
-using BFS	89.44	93.4	96.7	95.0	69.3	40.1	74.1
-using DFS	89.74	94.7	94.6	94.7	69.0	40.2	74.2

Benchmarking

UniXcoder results

Code Search	
CosQA	AdvTest
MRR	
70.1	**41.3**

Our results

CosQA

```
***** Eval results *****
  eval_mrr = 0.331
***** Eval results *****
  eval_mrr = 0.392
***** Eval results *****
  eval_mrr = 0.423
***** Eval results *****
  eval_mrr = 0.444
```

Average: 0.395

39.5 < 70.1

AdvTest

```
***** Eval results *****
  eval_mrr = 0.414
***** Eval results *****
  eval_mrr = 0.432
***** Eval results *****
  eval_mrr = 0.487
```

Average: 0.444

44.4 > 41.3

Vector Search

Design Challenge: How to find user queries?

- App needs to take user input for search
- Find most similar code segments

Implemented with: Python, Numpy, UniXcoder

Vector Search

Solution: Vector similarity search

- Embed code segments
- Dot product of vector comparison
- Ask LLM to rerank top 10

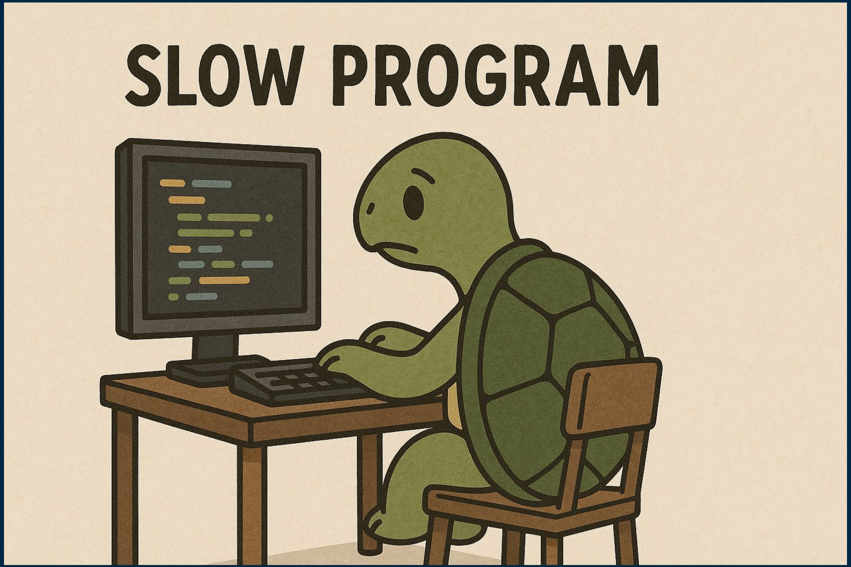
Major accomplishment

Performance Issues

- Performing vector search one-by-one
  - Vector comparison is an expensive computation
- JSON file operations are expensive

How bad can it be?

- Data.json could reach **248MB** and take upwards of 90 minutes*



Optimization Solutions

- SQLite is more efficient than JSON files.
- PyTorch provides easy use of hardware devices through their **DeviceModel**

How to solve the JSON issue?

- Querying vs reading
- Storing binaries vs writing strings to a file

vs.

PyTorch

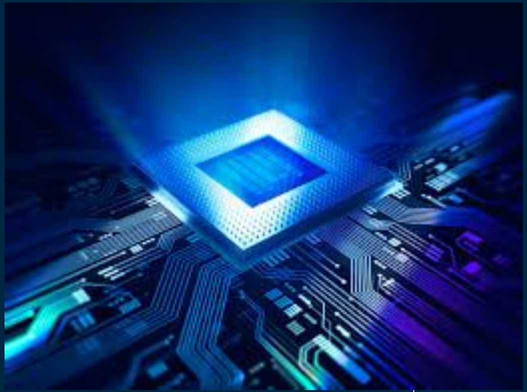
- Leading machine learning library
- Easy identification of available resources
  - Contains the **device** model

GPUs are awesome

- Able to process multiple, multi-dimensional embeddings concurrently
  - Faster tokenization
  - Easy parallel processing through batching
  - Bigger batches fed to the model (CUDA specifically)
- Parallel SQLite queries
  - Speeds up data loading
  - Speeds up data creation

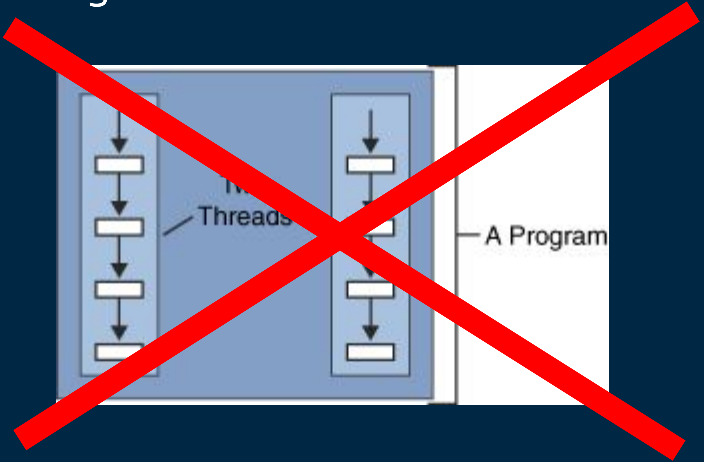
Not everyone has GPUs

- Program must be backwards compatible with a CPU-based solution



SQLite Issues

- Does not persist along threads



Give each thread their own connection

Results

- Data creation time experienced a **72%** improvement
- Data file (*embeddings.db)* is **80%** smaller than original data.json
- Searching embeddings.db can be up to **100x** faster*

LLM Integration

```
# Scoring prompt for LLM
SCORING_PROMPT_TEMPLATE = """
You are an AI evaluating code snippets.

- Your task: Score this snippet's relevance to the query.
- **Return ONLY a number between 0-10**.
- **DO NOT** explain your reasoning.
- **DO NOT** include any text, words, punctuation, or comments.
- Example response: `8`

### Query:
{query}

### Code Snippet:
{snippet}

### Response Format:
- Output **only** a single integer between `0` and `10`.
- No additional text.
- If unsure, provide your **best numerical estimate**.
"""
```

Design Challenges - LLM

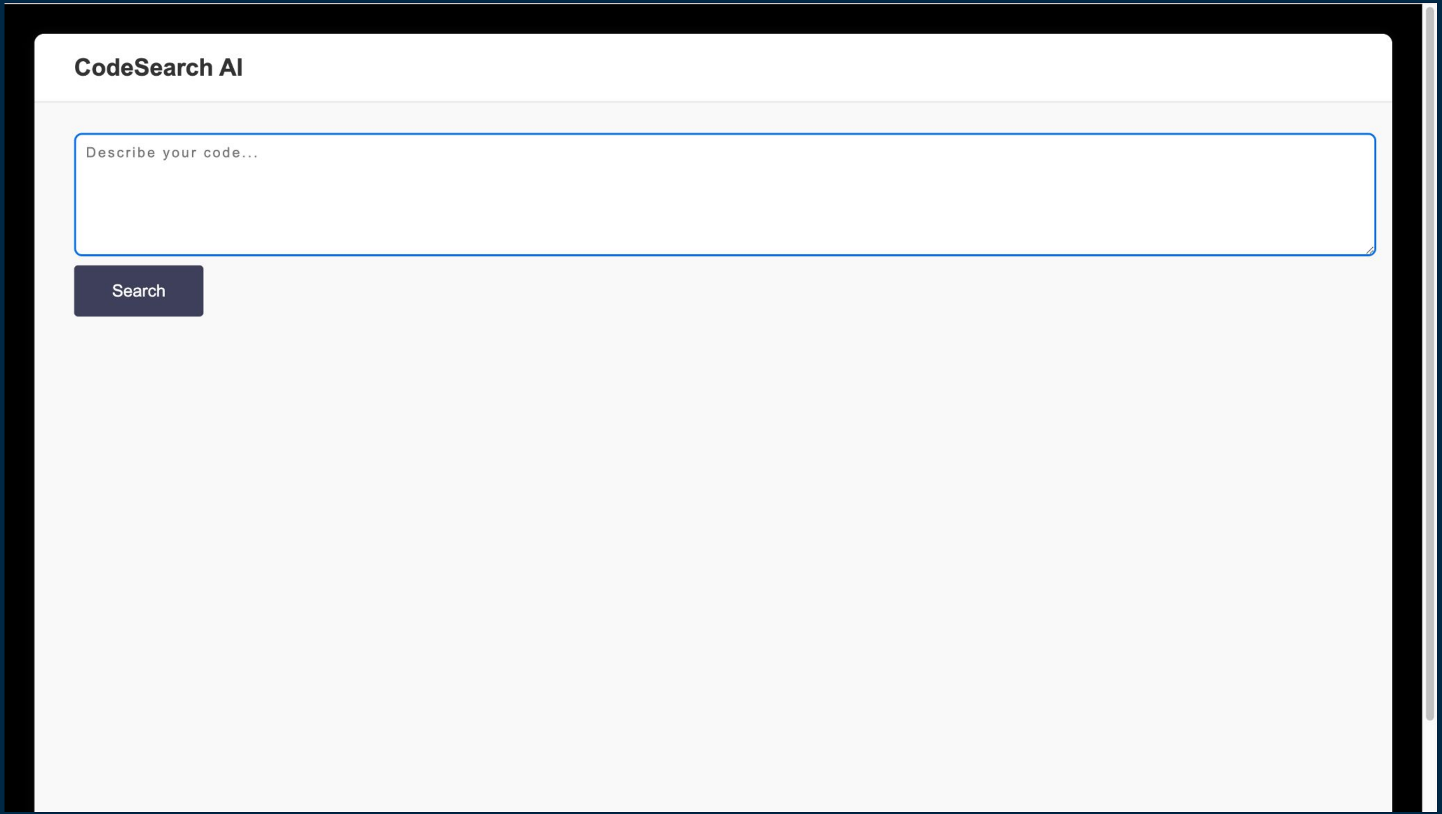
deepseek

- Tied scores → hurt MRR
- Ignored prompt format
- Frequent API timeouts
- Fixed with retries + smaller batches
- Prompt tweaks improved stability
- Core issue: model limitations, not prompt quality

Major Accomplishments - LLM

- Built a full end-to-end pipeline from natural language input to ranked code results
- Modular design allows easy updates and debugging
- Integrated LLM re-ranking with clean, consistent prompt format
- System works reliably with both small and large code inputs

Frontend Design



Design Challenges - Frontend

- LLM handled smoothly
- Scores unclear to users
- Prompt formatting preserved
- Simple and clean design

Major Accomplishments – Frontend

- Clean, responsive layout with lightweight CSS
- Developer-focused design with minimal distractions
- No JS framework — kept it simple and fast
- Added visual polish: loading animation & copy buttons

Team Member Contributions

Keenan Jacobs	Frontend, LLM integration, Flask app, prompt design, reports
Diego Perez	Embedding pipeline, SQLite storage, Hardware Acceleration/Parallelizing
Conor O'Shea	Vector search, benchmarking, embedding accuracy
Carter Cutsforth	Dataset initialization, LLM test integration, Ollama integration
Jacob Duba	Project management, System Architecture, Pairing

Demo

Q&A

Thank You