MedCodER: A Generative AI Assistant for Medical Coding

Overall Approach

MedCodER decomposes ICD coding into three interpretable stages—extraction, retrieval, and re-ranking—each leveraging the strengths of Generative AI and semantic search techniques.

1. Extraction

An LLM is prompted with chain-of-thought-style instructions to identify disease mentions, supporting evidence sentences (e.g., test results, treatment plans), and an initial set of potential ICD-10 codes directly from the medical record text. The model's outputs are then grounded back to the original record via fuzzy matching and BM25 to mitigate hallucinations ACL Anthology.

2. Retrieval

Extracted disease diagnoses are embedded and compared against a vector database built from textual descriptions of all valid ICD-10 codes (augmented with synonyms from the UMLS Metathesaurus). Semantic search retrieves the top-k most relevant candidate codes for each diagnosis, effectively narrowing the label space to a manageable set while preserving interpretability ACL Anthology.

3. Re-ranking

A second LLM—guided by the RankGPT framework—is used to re-rank the union of codes suggested in the extraction and retrieval steps. By considering only the extracted diagnoses and supporting evidence, this stage prioritizes codes based on contextually relevant information, yielding the final ranked list of ICD-10 predictions ACL Anthology.

Simplified 3-Step Overview (for our case)

Here's a concise, easy-to-grasp picture of how to combine generative AI with retrieval for ICD coding:

1. **Extraction**
   Use an LLM with "chain-of-thought" prompts to pull out disease mentions, supporting evidence sentences, and a first pass at possible ICD-10 codes from each document

Learn Prompting.

2. **Retrieval**
   Take those extracted disease terms and either (a) run a classic Okapi BM25 search over your ICD-10 code descriptions to narrow down candidates Wikipedia or (b) embed the terms and use FAISS for fast vector lookup against code definitions Engineering at Meta.

3. **Re-ranking**
   Feed the union of codes from steps 1 and 2 back into the LLM (via a ranking prompt) along with the evidence snippets, and ask it to sort and output the final, most contextually appropriate codes.

Core Components to Implement (for our case)

1. Extraction with an LLM

- **Chain-of-Thought Prompting**: Use structured prompts that ask the model to "think step by step" to identify clinical diagnoses, supporting evidence snippets, and an initial set of ICD-10 code proposals in JSON format ACL AnthologyarXiv.

- **Grounding via BM25**: Apply fuzzy string matching and an Okapi BM25 retriever (e.g., via the `rank-bm25` library) to align extracted evidence back to the original text, reducing hallucinations ACL AnthologyPyPI.

- **Implementation Notes**: Choose an LLM (GPT-4, Claude, or open models) and design prompts that output disease, evidence, and code fields; parse the JSON; then run fuzzy/BM25 matching (e.g., using `rank_bm25.BM25Okapi`) to verify and correct spans.

2. Retrieval Augmentation

- **UMLS-Enhanced Code Descriptions**: Compile textual definitions of all valid ICD-10 codes, enriched with synonyms from the UMLS Metathesaurus to handle varied medical terminology ACL AnthologyNational Library of Medicine.

- **Vector Indexing with FAISS**: Embed each code description using a sentence-embedding model (e.g., Sentence-Transformers or OpenAI embeddings) and index them with FAISS for fast approximate nearest-neighbor search GitHub.

- **Retrieval at Inference**: For each extracted diagnosis, compute its embedding and query the FAISS index to retrieve the top-k semantically closest ICD-10 codes by cosine

similarity Introduction | 🦜🔗 LangChain.

3. Code-to-Record Re-ranking

- **RankGPT-Style Prompting**: Combine the candidate codes from both extraction and retrieval steps and prompt an LLM to rank them, supplying only the extracted evidence and candidate descriptions to prioritize based on contextual relevance ACL AnthologyarXiv.

- **Prompt Design**: Structure the prompt to list "Supporting Evidence" and "Candidate Codes with Descriptions," then ask the model to output a final ranked list; parse the response to select the top codes.

Ways to obtain ICD-10 Codes

Here are several easy ways to obtain a full list of ICD-10 codes along with their human-readable descriptions:

1. Official WHO ICD-10 Codelists

The World Health Organization publishes downloadable CSVs that pair each code with its description.

- **Global Health Observatory (GHO) ICD10CAUSEGROUP Codelist**
    - Includes "CSV (text and codes)" that contains both the alphanumeric codes and their text labels.
    - Download link (choose "CSV (text and codes)"): WHO Apps
- **GHO ICD10CHAPTER Codelist**
    - Provides high-level chapter names (e.g. "A00–B99: Certain infectious and parasitic diseases").
    - Download link (choose "CSV (text and codes)"): WHO Apps

2. U.S. CDC ICD-10-CM Files

The Centers for Disease Control & Prevention maintain the official Clinical Modification (CM) versions of ICD-10. You can download full XML or PDF releases and extract code-to-description mappings yourself.

- **ICD-10-CM Files Page** (includes PDF/XML files dating back to 2019): CDC
- **CMS Valid ICD-10 Diagnosis List** (Excel):
    - Download the "Valid ICD-10 List (XLSX)" for the current year, which includes descriptions: Centers for Medicare & Medicaid Services

3. Python Packages with Built-In Code Descriptions

If you'd rather get descriptions programmatically, these libraries ship with up-to-date mappings:

- **`icd10-cm`** (PyPI)
    - Contains ICD-10-CM codes and their short descriptions.
    - Install via `pip install icd10-cm`: PyPI
- **`simple-icd-10`** (PyPI / GitHub)
    - Bundles the 2019 WHO ICD-10 CSV and lets you look up codes, ancestors/descendants, and descriptions in Python.
    - PyPI: PyPI
    - GitHub: GitHub
- **`python-icd10`** (GitLab)
    - Can automatically fetch and validate CDC's ICD-10 diagnosis database, providing code descriptions via a Python API.
    - GitLab: AI DevSecOps Platform

Key components for complete implementation

1. ICD-10 Code Resource

- **Download or load code–description pairs** from authoritative sources such as CMS's ICD-10-CM XLSX lists or a vetted GitHub CSV repository Centers for Medicare & Medicaid ServicesGitHub.

2. Data Preprocessing

- **Filter** by the set of filter-steps defined to select high-quality texts.
- **Clean and tokenize** the clinical notes (lowercasing, punctuation removal) to prepare inputs for the LLM and retrieval modules NVIDIA Developer.

3. LLM-Based Extraction Module

- **Chain-of-Thought Prompting**: Prompt your LLM (e.g., GPT-4) with "Let's think step by step…" to extract JSON-formatted diagnoses, evidence snippets, and an initial list of ICD-10 codes Learn Prompting.

4. Grounding Module (BM25)

- **Rank-BM25**: Use the `rank_bm25` library's Okapi BM25 implementation to fuzzy-match each extracted snippet back to the original text, anchoring the model outputs and reducing hallucinations PyPI Stack Overflow.

5. Retrieval Module (FAISS)

- **Embed** both extracted diagnosis terms and your ICD-10 code descriptions with a sentence embedding model.
- **Index and query** using FAISS for fast approximate nearest-neighbor search to get top-k semantically similar codes Pinecone GitHub.

6. Re-ranking Module (RankGPT)

- **Combine** the union of codes from extraction and retrieval.

- **Prompt** an LLM with a listwise “RankGPT” template to sort candidates by contextual relevance, outputting a final prioritized list DataCamp LlamaIndex.

7. UMLS Synonyms Expansion (Optional)

- **Leverage** the UMLS Metathesaurus API to augment code descriptions with synonyms and lexical variants, improving retrieval recall for diverse medical terminology National Library of Medicine.

8. Integration & Orchestration

- **Build** a pipeline that chains: Extraction → BM25 Grounding → FAISS Retrieval → RankGPT Re-ranking.
- **Implement** batching, logging, and error handling, ensuring each module passes along the `account` ID for alignment with `evaluation_df` NVIDIA Developer.

Step-by-Step Approach

1. Silver-Label Extraction (Offline)

- **Run an LLM** (e.g., GPT-4) with a simple “chain-of-thought” prompt on each cleaned document to extract:
    - **Disease mentions** (e.g., “acute renal failure”)
    - **Evidence sentences** (e.g., “Serum creatinine rose from 1.0 to 2.5 mg/dL”)
    - *(Optional)* Initial candidate ICD-10 codes for quality checking
- **Why:** Leveraging silver-standard annotations boosts downstream learning PMCCEUR-WS.

2. Input Augmentation

- **Construct each training example** by concatenating:

1. A **prefix**: `[EVIDENCE] Serum creatinine rose from 1.0 to 2.5 mg/dL. [MENTIONS] acute renal failure.`
2. The **cleaned full text** of the clinical note.

- **Why:** Prefixing evidence focuses the model on salient information while retaining full context arXiv.

3. Single-Pass Fine-Tuning

- **Fine-tune mamba** on these **augmented inputs** against your gold `Final_CM` labels.
- **Configuration:** Use your existing data splits (train/val/test), learning rates, and batch sizes.
- **Why:** The model learns to correlate both distilled evidence and broader context with ICD codes in **one** training run—no separate pretraining step needed ACL AnthologyACL Anthology.

4. Inference Pipeline

- For a new document:
    1. **Clean** the text.
    2. **Extract** silver evidence via the same LLM prompt.
    3. **Prefix** the evidence to the cleaned text.
    4. **Predict** ICD codes with your fine-tuned mamba.
- **Why:** Ensures consistency between training and inference, maximizing performance gains from the augmented inputs arXivarXiv.

Why This Is Efficient and Effective

- **Single Training Cycle:** Avoids the overhead of both silver-label pretraining and separate gold-fine-tuning steps, cutting training time in half compared to two-phase approaches arXiv.

- **Rich Context + Focused Cues:** The model benefits from LLM-curated highlights without losing broader context—addressing concerns about dropped information during cleaning Yu Meng.

- **Production-Ready:** You retain a **pure mamba** architecture in production; the paper's LLM is used only offline for data annotation CEUR-WS.

- **Proven Silver-Label Gains:** Similar hybrid methods (Clean-LaVe, distant supervision) show 3–8% performance lifts on IE tasks by integrating silver labels ACL AnthologyACL Anthology.