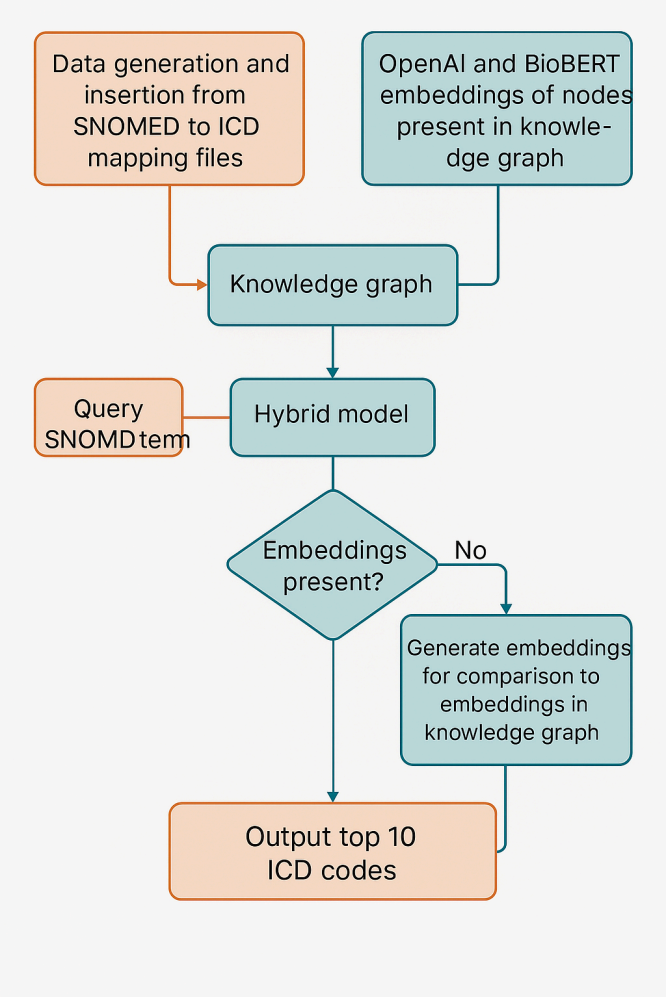
Knowledge Graph Query Engine for SNOMED to ICD Code Mapping

Abstract

This document describes the iterative development of a SNOMED-to-ICD code mapping system using embedding-based semantic search and hybrid reranking techniques. The goal is to reach 90% Top-10 accuracy. We evaluated three main approaches and achieved 86% using a hybrid reranker model based on BioBERT embeddings.

1. Overview

This query engine helps users retrieve the top 10 most relevant ICD codes from the knowledge graph by inputting SNOMED terms. The final aim is to obtain 90% Top-10 accuracy. Currently, the query engine is passing through LLM (azure open AI) to retrieve the top 10 ICD codes. We hope to replace the LLM engine with Knowledge graph. As Knowledge graph is better for structured tasks like ICD mapping because it provides precise, explainable, and rule-based results, while LLMs may hallucinate or miss domain-specific logic.



The above is an overview of the flow of the knowledge graph query engine.

2. Data Processing

Purpose: Three methods were tested for SNOMED-to-ICD mapping: entity type classification, paragraph enrichment, and direct mapping. Direct insertion of mappings without additional enrichment achieved the highest accuracy (82%) and was chosen for final use.

Method 1: GPT-based Entity Type Classification

I have tried categorising the Snomed descriptions into the 12 entity types available. The purpose of this is to do entity linking so that it can look out for keywords in the Snomed description in order to accurately output the relevant ICD codes.

However, it affects the embeddings as embedding a word by itself is not useful to output the ICD codes. There is also limited information for some entity types such as complication, trimester and other_info.

Entity Types:

1. Condition
2. Body part
3. Severity
4. Encounter type
5. Cause
6. Laterality
7. Person
8. Fetus
9. Procedure
10. Complication
11. Trimester
12. Other info

I have attempted to generate the entity types via both GPT 4o and 4o-mini. The results for 4o was better as it is able to identify and break down more specific words into each category (eg. unspecified). I decided to use the results from that for the following attempts.

The accuracy of using this method was not as satisfactory of up to 62% for unseen data even though all the seen data was able to be correctly predicted since it is already in the knowledge graph.

Method 2: Paragraph Enrichment Using GPT

Since embedding requires more chunks of text, I generated descriptions for each Snomed term to increase the paragraph of text available for embeddings. The use GPT 4o - mini to generate the paragraph of description using prompt as follows:

```
"Explain each of the following medical terms in a paragraph suitable for non-medical users. "
    "Do not repeat the term itself, and do not number or label the answers. Just return one paragraph per term, separated by line breaks:\n\n"
    F"{listed_terms}"
```

The following is an example of the output:

A	B	C	D	E	F
id	med_conce	d_preferre	0am_descri	d10am_cod	snomed_description
0	2.4E+08	Cholera du	Cholera du	A00.0	Cholera caused by a specific strain of bacteria is an infectious disease that leads to severe diarrhea and dehydration. This particular strain, known as the Classical biotype, is one of the older forms of the bacteria and can cause outbreaks, especially in areas with poor sanitation. The disease is primarily transmitted through contaminated water or food, and without prompt treatment, it can be life-threatening. Symptoms usually appear suddenly and can escalate quickly, necessitating immediate medical attention.

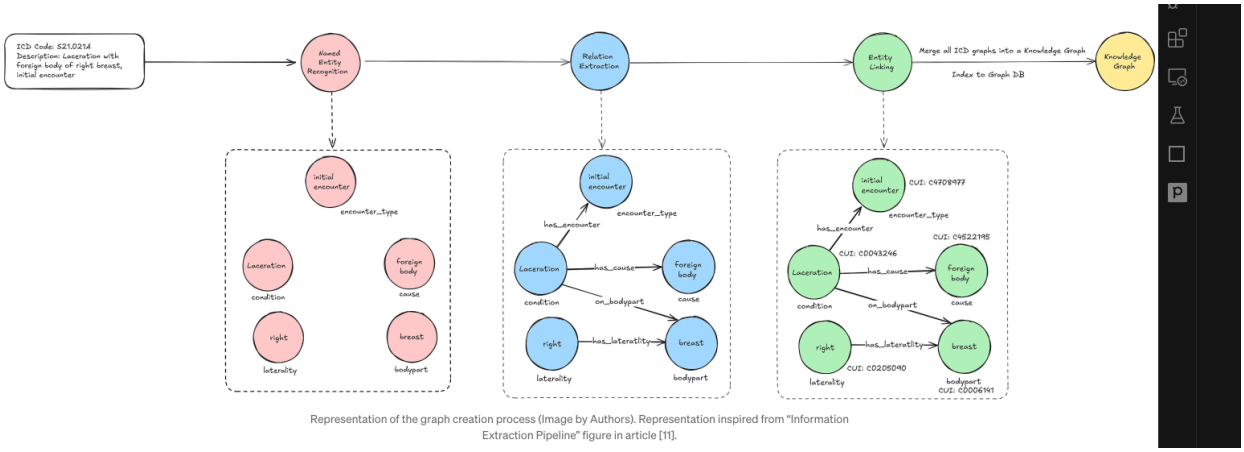
Although the accuracy has increased slightly, it reaches a maximum of 70% after all the following approaches are used. Hence, it did not help much in the retrieval process.

Method 3: Direct Mapping Insertion

Direct insertion of Snomed to ICD File mapping and ICD masterfile gave the highest accuracy when testing for unseen data of up to 82% accuracy as the base model. So I proceed on with this method instead for the insertion of data later on.

This means that no additional information or classification was added to it and embeddings are done on both Snomed and ICD terms and description later on.

Method 4: Categorisation of ICD Description



Representation of the graph creation process (Image by Authors). Representation inspired from "Information Extraction Pipeline" figure in article [11].

I have tried categorising the ICD descriptions into the 12 entity types available. The purpose of this is to do entity linking so that it can look out for keywords in the ICD description in order to accurately output the relevant ICD codes.

All the ICD codes in the master sheet have been categorised into the following:

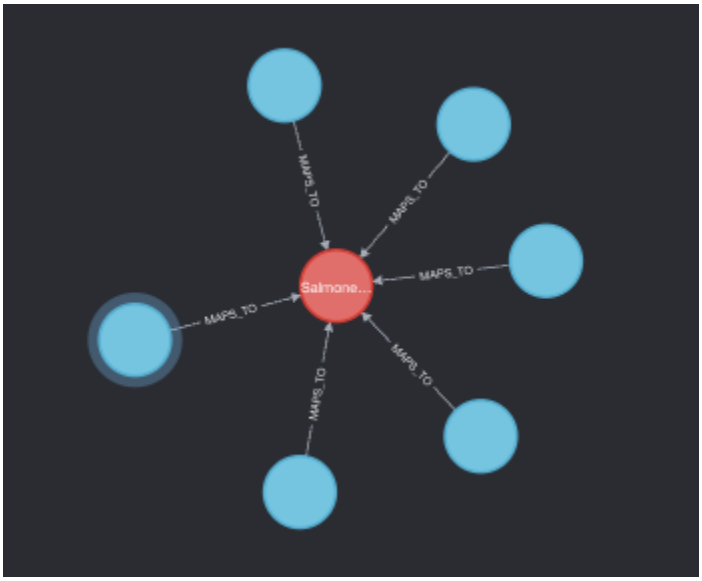
Entity Types:

1. Condition
2. Body part
3. Severity
4. Encounter type
5. Cause
6. Laterality
7. Person
8. Fetus
9. Procedure
10. Complication
11. Trimester
12. Other info

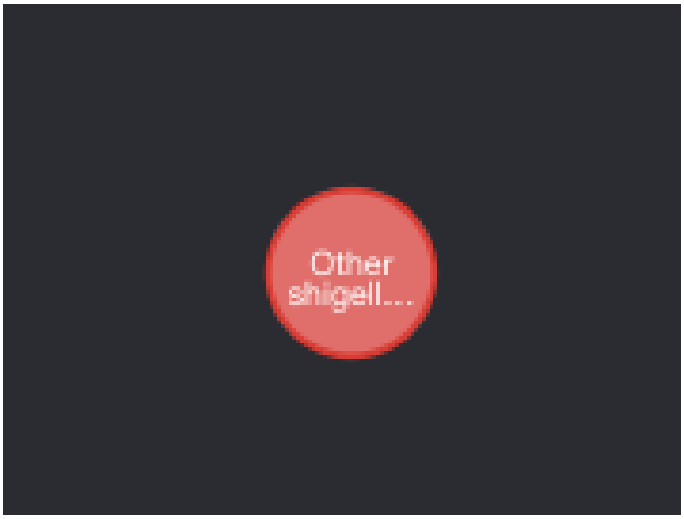
3. Insertion of Data

Purpose: The SNOMED-to-ICD mappings and ICD masterfile entries were inserted into Neo4j to form a knowledge graph. Linked SNOMED terms connect to ICD nodes via `MAPS_TO` relationships, while unlinked ICD codes remain as standalone nodes.

It involves the process of inserting data from Snomed to ICD code mapping files and ICD codes master files into the neo4j database to form the knowledge graph. It will form relations for each Snomed term to icd code or remain as an individual ICD node if it is not linked to any. The following illustrations show the visualisation of how the knowledge graph is like.



This is an example of an ICD code node (red) connected to relevant SNOMED terms (blue)



This is an example of how an ICD code is not connected to any Snomed term as it is extracted from the masterlist sheet

4. Embedding Models Evaluated

Purpose: Several embedding models were tested, including SapBERT, BioBERT, and OpenAI's ADA model. Due to compatibility issues, only BioBERT (768D) and OpenAI (1536D)

embeddings were used, and both were stored on SNOMED and ICD nodes in Neo4j for similarity-based retrieval.

I have tried several embedding models as listed below:

1. Sapbert: Cambridgeltl/SapBERT-from-PubMedBERT-fulltext - incompatible version
2. Embedding in Neo4j (gds.knn.stream) - low accuracy
3. BioBert/ ClinicalBert: emilyalsentzer/Bio_ClinicalBERT
4. OpenAI embedding: test-embedding-ada-002

Due to compatibility version issues with neo4j database, we have scaled to using only biobert embedding (768 dimension) and OpenAI embedding (1536 dimension).

Biobert embedding model: emilyalsentzer/Bio_ClinicalBERT
OpenAI embedding model: test-embedding-ada-002

Both Snomed and ICD code nodes have these 2 embeddings available as shown below.

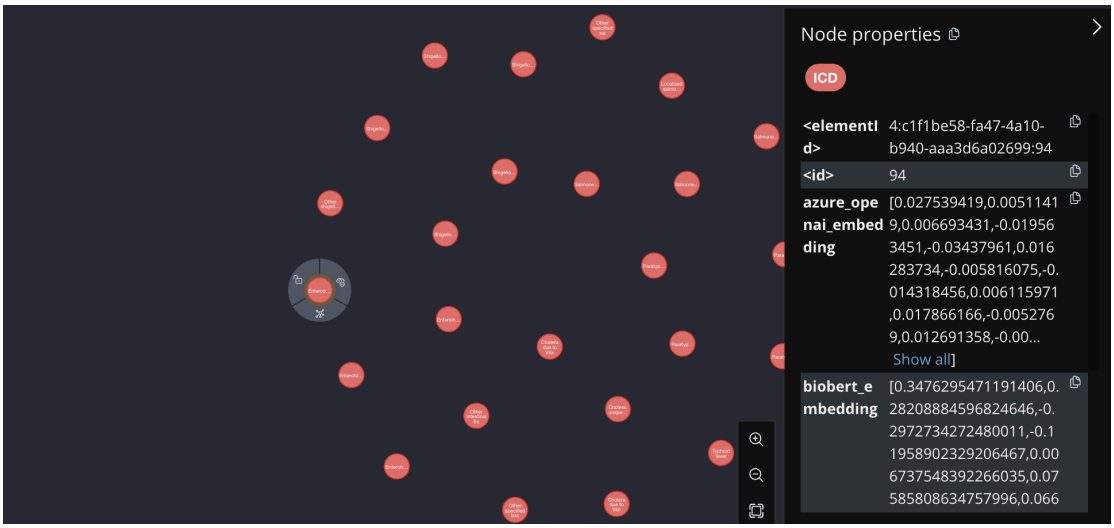


Illustration for ICD codes

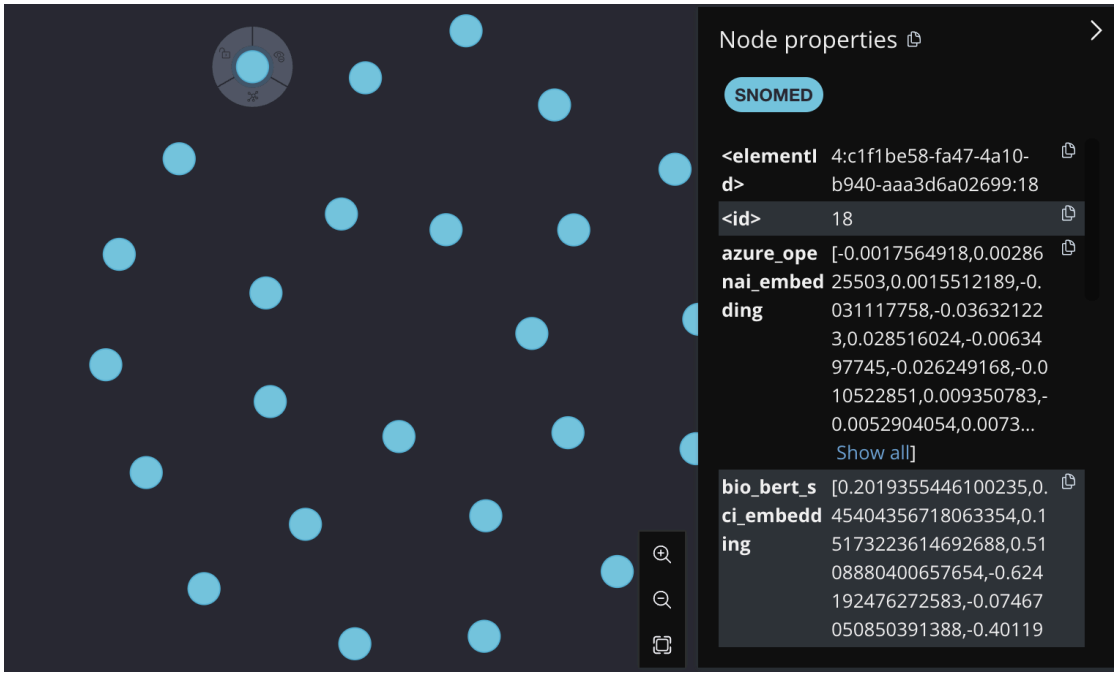


Illustration of SNOMED terms

5. Setup for Query Embedding

Purpose: For query embedding, BioBERT and OpenAI models were used to generate embeddings for unseen SNOMED terms, which were then compared to existing embeddings in the knowledge graph using GDS cosine similarity.

BioBERT:

For testing of user queries, the code ensures that the biobert_embeddings are available for the Snomed term queries to create vector index:

```
def ensure_vector_index_exists():
    with driver.session() as session:
        print("🔍 Checking for BioBERT vector indexes...")
        indexes = session.run("SHOW INDEXES").data()
        index_names = [i["name"] for i in indexes]

        if "biobert_embedding_index" not in index_names:
            print("⚙️ Creating BioBERT vector index for SNOMED...")
            session.run("""
                CREATE VECTOR INDEX biobert_embedding_index
                FOR (s:SNOMED) ON (s.biobert_embedding)
                OPTIONS {
                    indexConfig: {
                        `vector.dimensions`: 768,
                        `vector.similarity_function`: 'cosine'
                    }
                }
            """)
```

else it will undergo the embedding process:

```
# === Generate BioBERT embedding ===
def get_embeddings(texts):
    print(f"🧠 Generating embedding for input...")
    start = time.time()
    embeddings = BIOBERT_MODEL.encode(texts, show_progress_bar=True).tolist()
    print(f"✅ Embedding completed in {time.time() - start:.2f} sec")
    return embeddings
```

before moving on to using GDS cosine similarity function to extract the top 10 similar ICD codes.

```
# === Query ICD directly ===
def query_direct_icds(session, embedding, top_k):
    results = session.run("""
        CALL db.index.vector.queryNodes('biobert_embedding_index_icd', $topK, $embedding)
        YIELD node, score
        RETURN node.description AS term, node.code AS id, score,
               node.code AS icd_code, node.description AS icd_description,
               'ICD' AS source
        ORDER BY score DESC
        LIMIT $topK
    """, {"topK": top_k, "embedding": embedding})
    return results.data()
```

The embeddings generated for the queried Snoemed term can be compared to the embeddings of the Snomed term in the knowledge graph or directly to the ICD codes embeddings itself.

Results:
The accuracy of this model is 72%. There are specific chapters that cannot be predicted such as T43 and S40. Also, for subchapters most of the “others” chapters (eg .8/.9) are predicted wrongly as well.

OpenAI:

For testing of user queries, the code ensures that the azure_openai_embeddings are available for the Snomed term queries to create vector index:

```
# === Step 2: Fetch SNOMED and ICD data using paging to handle large datasets ===
def fetch_all_snomed_and_icd_data():
    all_records = []
    page_size = 10000
    skip = 0

    with driver.session() as session:
        while True:
            # Use paging to fetch data in chunks
            query = """
                MATCH (s:SNOMED)
                WHERE s.description IS NOT NULL AND s.preferred_term IS NOT NULL
                OPTIONAL MATCH (s)-[:MAPS_TO]->(i:ICD)
                WHERE i.description IS NOT NULL
                RETURN s.concept_id AS snomed_id, s.description AS snomed_text,
                       s.preferred_term AS snomed_term, i.code AS icd_code,
                       i.description AS icd_description
                SKIP $skip LIMIT $limit
            """

            results = session.run(query, skip=skip, limit=page_size).data()
            all_records.extend(results)

            if len(results) < page_size:
                break

            skip += page_size
            print(f"Fetched {len(all_records)} records so far...")
```

else it will undergo the embedding process:

```python
# === Step 3: Async function to get Azure OpenAI embeddings ===
async def get_embeddings_async(texts, batch_size=EMBEDDING_BATCH_SIZE):
    """Asynchronously get embeddings from Azure OpenAI API"""
    all_embeddings = []

    # Filter None values
    filtered_texts = [text for text in texts if text]

    # Prepare batches
    batches = [filtered_texts[i:i+batch_size] for i in range(0, len(filtered_texts), batch_size)]
    print(f"🧠 Processing {len(filtered_texts)} texts in {len(batches)} batches")

    # Set up headers once
    headers = {
        "Content-Type": "application/json",
        "api-key": AZURE_OPENAI_API_KEY,
        "Accept": "application/json"
    }
```

before moving on to comparing with the embeddings in the knowledge graph

The embeddings generated for the queried Snoemed term can be compared to the embeddings of the Snomed term in the knowledge graph or directly to the ICD code embeddings itself.

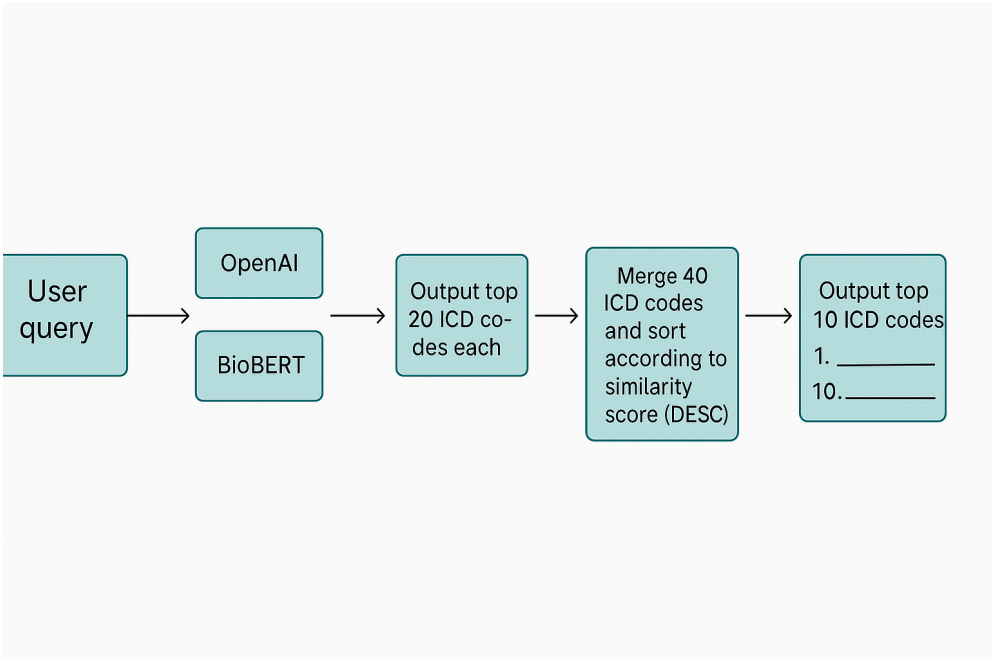
Results:
The accuracy of this model is 74%. There are still specific chapters that cannot be predicted such as T43 and S40. For subchapters most of the “others” chapters (eg .8/.9) are predicted correctly. However subchapters like (eg .42) are still incorrect.

6. Query hybrid model

Purpose: After doing random sampling of queries using unseen data for the 2 individual models above, I discovered that some outputs were successfully retrieved from only one of the models. Hence, I attempt to use the hybrid method to obtain higher accuracy .

Approach 1: Merge Top-K from Both Models

I tried to obtain the top 20 ICD codes from each model and sort them individually to obtain the top 5 ICD codes from each model to get the total of 10 icd codes output to users as visualized from the bottom.



Firstly, it will query from both models to retrieve top 20 ICD codes respectively:

```
# Step 1: Query both models with a larger pool (top 20 each)
try:
    openai_results = query_openai(query_text, top_k=20)
    biobert_results = query_biobert(query_text, top_k=20)
except Exception as e:
    print(f"❌ Error fetching results: {e}")
    return []

print(f"📩 OpenAI returned {len(openai_results)} results")
print(f"📩 BioBERT returned {len(biobert_results)} results")
```

After which, it will merge the top 40 ICD codes and remove duplicates of ICD codes to keep the highest score:

```python
# Step 3: Deduplicate by ICD code, keep highest score
merged = {}
for entry in combined:
    code = entry["icd_code"]
    if code not in merged or entry["score"] > merged[code]["score"]:
        merged[code] = entry

unique_predictions = list(merged.values())
unique_predictions.sort(key=lambda x: -x["score"])
final_top = unique_predictions[:top_k]
```

Lastly, it will output the top 10 ICD codes that are sorted based on confidence score:

```python
# Step 4: Output
print(f"\n🔟 Final Top-{top_k} ICD Predictions (Expanded Pool, Deduped):")
for i, res in enumerate(final_top):
    print(f"{i+1}. [{res['source']}] ICD: {res['icd_code']} | {res['icd_description']} (Score: {round(res['score'], 4)})")

return final_top
```

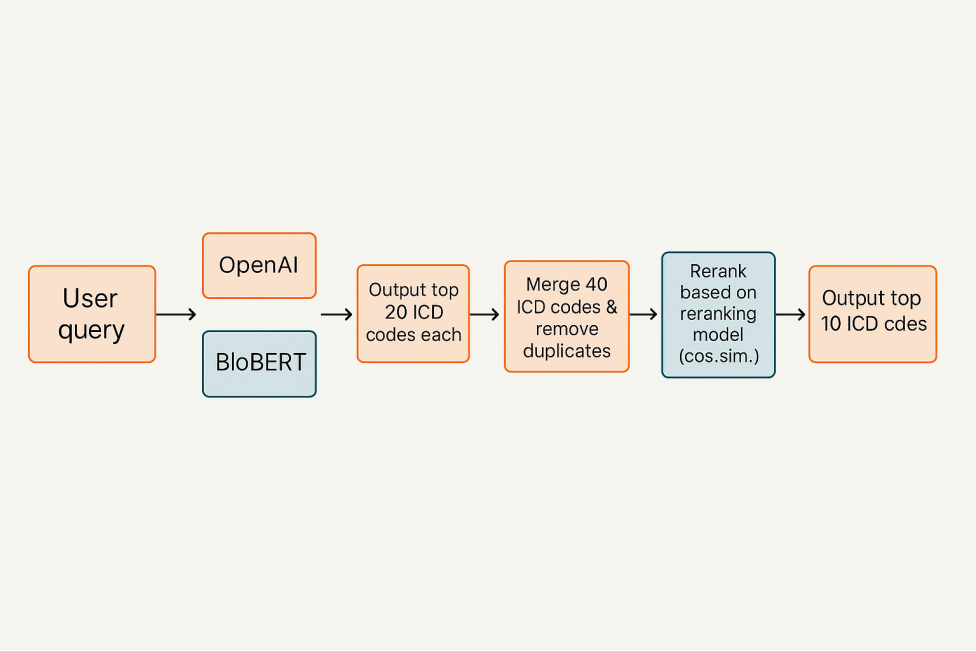
Results:

The accuracy increased up to 84% with the hybrid method, also solving the issue of (.8 and .9) chapters. However, for chapters that are in the ICD code masterlist sheet, it is still very hard to predict it as there are no Snomed terms linked to it and comparing the embeddings it differs by quite a bit.

The reason for choosing K = 20 is after testing out 10,30,40,and 50 where K increases from 82% (K=10) to 84% followed by a dip back to 82% when it reaches K=30 and further on. Hence, I took the highest performing K for the following attempts.

Approach 2: Hybrid + Reranking using open source model

I tried including a reranking model to it using different dimensions as well (384, 768, 1536). Hypothetically, higher dimensions should allow for embeddings to capture more nuanced semantic relationships. However, the following results will prove otherwise.



I went to look up the available open source reranking models of the various dimensions. The three dimensions above are the one compatible with the neo4j database.

Model Name	Dimensions	Use With	Notes
`pritamdeka/S-BiomedBERT-MS-MARCO`	768	`SentenceTransformer`	Biomedical domain. Trained for MS MARCO, suitable for retrieval tasks. ✅
`pritamdeka/BioBERT-mnli-snli-scinli-scitail-mednli`	768	`SentenceTransformer`	Biomedical + natural language inference fine-tuned
`intfloat/e5-large-v2`	1536	`SentenceTransformer`	State-of-the-art reranker model, very high quality. ✅
`sentence-transformers/all-MiniLM-L6-v2`	384	`SentenceTransformer`	Lightweight and fast general-purpose model
`sentence-transformers/paraphrase-MiniLM-L12-v2`	768	`SentenceTransformer`	Good general semantic similarity baseline
`microsoft/BiomedVLP-CXR-BERT-general`	1024	`transformers`	Medical vision-language pretrained. Need custom pooling for embeddings ❗
`GanjinZero/biobert-nli`	768	`transformers`	NLI fine-tuned BioBERT, needs pooling logic manually

I have attempted to include all the models except the 1024 dimension one and 1536 dimension as the params were too big and my computer crashed.

Firstly, it will query both the models once the query is passed. Each model will retrieve the top 20 ICD codes:

```python
# Step 2: Query both models
try:
    openai_results = query_openai(enriched_query, top_k=20)
    biobert_results = query_biobert(enriched_query, top_k=20)
except Exception as e:
    print(f"❌ Error fetching results: {e}")
    return []

print(f"📩 OpenAI returned {len(openai_results)}")
print(f"📩 BioBERT returned {len(biobert_results)}")
```

(variable) enriched_query: str | Any

After which, it will remove the duplicates of ICD codes and keep the highest score and merge the top 40 ICD codes:

```python
# Step 4: Deduplicate by ICD code (keep highest scoring)
merged = {}
for entry in combined:
    code = entry["icd_code"]
    if code not in merged or entry["score"] > merged[code]["score"]:
        merged[code] = entry

unique_predictions = list(merged.values())
```

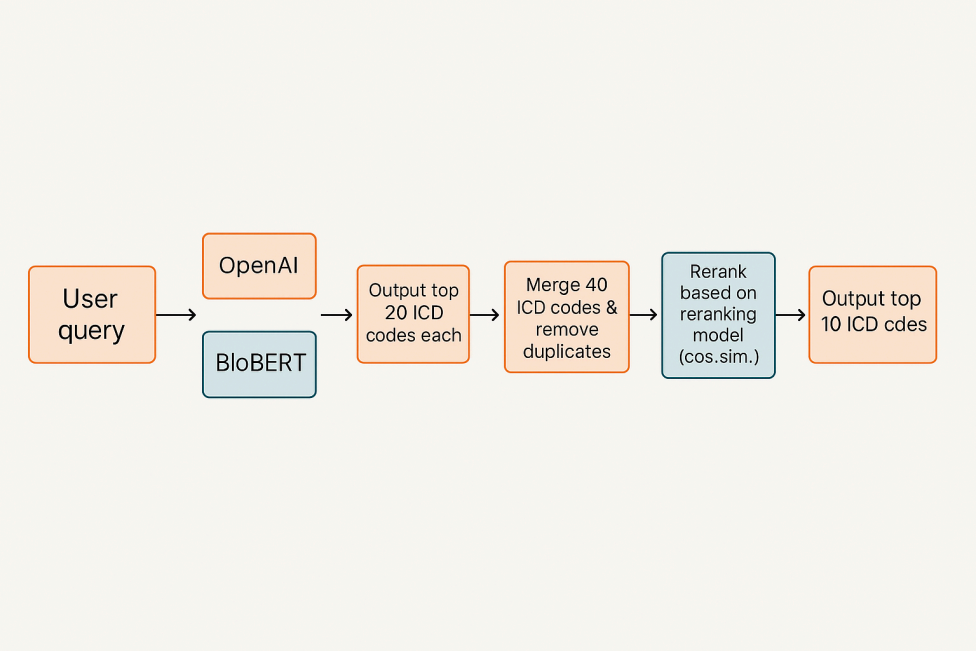
It will then pass through the Reranking model to get sorted based on cosine similarity and output the top 10 ICD codes:

```python
# Step 5: Reranking with different dimensions of model (formatted input)
print("🧠 Reranking by semantic similarity with ICD descriptions...")
query_embed = rerank_model.encode(f"query: {query_text}", convert_to_tensor=True)

for entry in unique_predictions:
    entry_desc = entry["icd_description"] or ""
    entry_embed = rerank_model.encode(f"passage: {entry_desc}", convert_to_tensor=True)
    entry["rerank_score"] = util.cos_sim(query_embed, entry_embed).item()

    # Bonus if first 3 or 4 digits match
    if icd_chapter_hint and entry["icd_code"].startswith(icd_chapter_hint):
        entry["rerank_score"] += 0.05
    if enrichment_result.get("icd_code", "")[:4] == entry["icd_code"][:4]:
        entry["rerank_score"] += 0.1

final_top = sorted(unique_predictions, key=lambda x: -x["rerank_score"])[:top_k]

# Step 6: Output
print(f"\n🔟 Final Top-{top_k} ICD Predictions (Reranked):")
for i, res in enumerate(final_top):
    print(f"{i+1}. [{res['source']}] ICD: {res['icd_code']} | {res['icd_description']} (Score: {round(res['rerank_score'], 4)}")

return final_top
```

Results:

Based on the evaluation, the model that output the highest accuracy of 86% was the second one "pritamdeka/BioBERT-mnli-snli-scinli-scitail-mednli". This is hence the highest accuracy I could obtain so far after trying out many different methods. However, there are still chapters starting with "T", "S" and "L" that are predicted wrongly as well as the niche subchapters that were previously mentioned. The biggest problem was with "S63.82" which heavily affects the accuracy since it took up a large proportion of the unseen test data.

Approach 3: Hybrid + Reranking using GDS cosine similarity (inbuilt function)



Firstly, I retrieve 10000 embeddings from each embedding model:

```
# === Boosted Hybrid ICD Query (with GDS enhancement) ===
def smart_hybrid_query_simple_boosted(query_text, groundtruth_code=None, top_k=10):
    print(f"\n🔍 ICD Hybrid Query (Boosted) for: '{query_text}'")

    try:
        openai_results = query_openai(query_text, top_k=10000)
        biobert_results = query_biobert(query_text, top_k=10000)
    except Exception as e:
        print(f"❌ Error fetching results: {e}")
        return []
```

Based on error analysis, I used boosting logic to reward high error clusters like (eg. S60, T89) - serves as a tiebreaker.

If a predicted code shares the same first 4 or 3 characters, or belongs to the same chapter as the correct code, it receives a higher score—because these usually indicate codes from the same diagnostic family. Additionally, codes from known high-error clusters, get an extra score boost to

help recover common past mistakes. This approach complements semantic similarity by adding domain knowledge, ensuring that structurally relevant codes are more likely to appear in the top predictions:

```
# High-error ICD prefixes (still used for general boosting, not based on groundtruth)
high_error_prefixes = {"S60", "S63", "T89", "S90", "S93", "C79", "L03", "T43", "K08"}
error_suffix_boost = {
    "88": 0.20, "89": 0.08, "50": 0.03, "51": 0.08,
    "60": 0.03, "61": 0.08, "62": 0.03, "01": 0.15, "81": 0.10
}
```

```
# General boosting without using groundtruth
for entry in merged.values():
    icd_code = entry["icd_code"]
    icd_desc = entry["icd_description"] or ""

    # Boost known error-prone prefix clusters
    if icd_code[:3] in high_error_prefixes:
        entry["score"] += 0.05

    # Boost for suffix-based errors
    if '.' in icd_code:
        suffix = icd_code.split('.')[-1]
        if suffix in error_suffix_boost:
            entry["score"] += error_suffix_boost[suffix]

    # Boost for keyword overlap (query ↔ description)
    query_keywords = set(query_text.lower().split())
    desc_keywords = set(icd_desc.lower().split())
    if query_keywords & desc_keywords:
        entry["score"] += 0.03
```

Rationale:
High-error clusters (prefix boost):

- These are known problematic ICD chapters where models tend to underperform.

- Boosting helps prioritize commonly mistaken groups (like injury codes `S60`, cancer codes `C79`, etc.).

Suffix boost:

- Codes like `.88`, `.89`, `.01` often mean “other” or “unspecified” and frequently occur in ambiguous descriptions.

- These are known to be common fallback answers — boosting them helps reduce severe misses.

Keyword overlap:

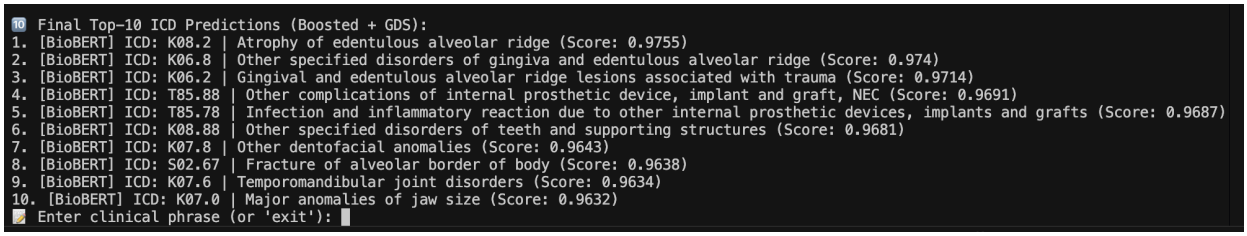
- If query mentions a term like “finger” and the ICD description also contains “finger”, it gets a small lift.

Boost Type	Description	Score Boost
Prefix-based cluster boost	If the ICD code’s prefix (first 3 characters) is in the `high_error_prefixes` set (e.g., `S60`, `T89`, `C79`, etc.), apply a boost.	`+0.05`
Suffix-based boost	If the ICD code has a decimal (e.g., `T85.88`), check the suffix (`88`, `89`, etc.) and boost according to the `error_suffix_boost` dictionary.	Varies (e.g., `+0.20` for `88`, `+0.08` for `89`
Keyword overlap between query and ICD description	If any words from the input query (`query_text`) appear in the ICD description (case-insensitive), apply a small boost.	`+0.03`

After which, I use GDS cosine similarity as my reranking function to output the top 10 ICD codes:

```
# === Optional: Rerank using GDS cosine similarity from filtered graph ===
try:
    with driver.session() as session:
        query = """
        MATCH (s:SNOMED {preferred_term: $term})
        WITH id(s) AS snomedId
        CALL gds.knn.stream('filtered_embedding_graph', {
            nodeProperties: ['azure_openai_embedding'],
            topK: $topK,
            similarityMetric: 'COSINE'
        })
        YIELD node1, node2, similarity
        WHERE node1 = snomedId
        RETURN gds.util.asNode(node2).code AS icd_code,
               gds.util.asNode(node2).description AS icd_description,
               similarity
        ORDER BY similarity DESC
        LIMIT $topK
        """
        gds_preds = session.run(query, {"term": query_text, "topK": top_k}).data()
        for gp in gds_preds:
            gds_code = gp["icd_code"]
            if gds_code not in merged:
                merged[gds_code] = {
                    "source": "GDS",
                    "icd_code": gds_code,
                    "icd_description": gp["icd_description"],
                    "score": gp["similarity"]
```

Results:
Based on the evaluation, this gives the highest accuracy of 95.74% while I am still attempting to find the optimised K for retrieval of embeddings. The main clusters of errors come from chapters S60, S50 and S40. Specifically S60.88 took up 20% of the error.



7. Evaluation Methodology

Accuracy = percentage of queries where the ground truth ICD code is present in the top 10 predicted codes.

I used a gradual implementation of methods to slowly improve the accuracy of the model.

I first tested the individual models where BioBert is a medical specific embedding model and OpenAI is a general embedding model. Both had its pros and cons, I decided to use the ensemble method by combining both models so that it can take in a wider diversity of SnoMed terms depending of how specific the Snomed term is.

Furthermore, I tried different reranking models which help to reorder predicted ICD codes based on deeper semantic understanding, improving the chances of correct predictions.

The inbuilt GDS cosine similarity in Neo4j is often better than using an open-source model for retrieval tasks because it's tightly integrated into the graph engine, allowing high-speed, memory-efficient similarity searches directly on stored embeddings, without needing to export data or build custom indexing pipelines. It leverages optimized native graph structures, parallel computation, and supports dynamic queries on live graph data, making it faster, more scalable, and easier to maintain than managing external models or embedding indexes separately.

Approach	Accuracy
BioBert model	72%
OpenAI model	74%
Hybrid (Merge K)	84%
Hybrid + Rerank using open source model	86%

Hybrid + GDS cosine similarity	95%

8. Challenges Identified

There are a few challenges along the way when trying to achieve the goal of 90% accuracy for top 10 ICD codes.

1. Issue with .8/.9 chapters
    a. Initially there were alot of .8/.9 subchapters that was not able to be predicted due to the limited number of Snomed terms that formed relations with the ICD codes in the knowledge graph
    b. This was probably due to the limitations of LLM generation when it was classifying or generating description as after I did direct insertion of the Snomed terms there was an accuracy boost as about 302 unseen data contains such chapters
    c. Currently, it is left with 38% (116/302) that are still predicted inaccurately

2. Wrong predictions of subchapters
    a. The above is an example of the wrongly predicted subchapters. This is a current problem that is affecting accuracy. 18/220 errors are derived from the wrongly predicted subchapters

A	B	C
query	gold_icd	top1_pred
ligament of left wrist joint	S63.50	S63.3
Traumatic rupture of collateral ligament of joint of left little finger	S63.68	S63.4
Traumatic rupture of volar plate of joint of left little finger	S63.68	S63.4
Traumatic rupture of volar plate of joint of left ring finger	S63.68	S63.4
Traumatic rupture of ligament of joint of right middle finger	S63.68	S63.4
Traumatic rupture of ligament of right wrist joint	S63.50	S63.3
Traumatic rupture of ligament of joint of right index finger	S63.68	S63.4
Traumatic rupture of collateral ligament of joint of right little finger	S63.68	S63.4
Traumatic rupture of volar plate of joint of right little finger	S63.68	S63.4
Traumatic rupture of volar plate of joint of right index finger	S63.68	S63.4
Traumatic rupture of collateral ligament of joint of right index finger	S63.68	S63.4

3. Uncertainty of the predictions against ground truth
    a. After some random sampling, it is realised that some of the ICD codes generated by the model are more specific than what the grountruth asked for. I have compiled a list for verification and the following is an example.

SCT_Preferred_term	Stds ICD-10AM	ICD-10AM Description	biobert_check	Remark	Note
Traumatic rupture of collateral ulnar ligament of right elbow	S53.42	Sprain and strain of ulna collateral ligament		1. ICD: S53.3 | Traumatic rupture of ulnar collateral ligament (Score: 0.9931)	
Traumatic rupture of collateral ulnar ligament of left elbow	S53.42	Sprain and strain of ulna collateral ligament		1. ICD: S53.3 | Traumatic rupture of ulnar collateral ligament (Score: 0.9936)	
Haematoma of right ankle	S90.88	Other superficial injuries of ankle and foot		1. ICD: S90.3 | Contusion of other and unspecified parts of foot (Score: 0.9925)	suggested with reference to Haematoma of right foot parked under S90.3
Haematoma of left ankle	S90.88	Other superficial injuries of ankle and foot		1. ICD: S90.3 | Contusion of other and unspecified parts of foot (Score: 0.9925)	suggested with reference to Haematoma of left foot parked under S90.3

4. ICD codes not found in Knowledge graph
    a. Previously, I found that a few of the ICD codes are not found in the SNOMED to ICD code mapping file and hence not inserted into the Knowledge graph
    b. After which, I got the ICD code master sheet and updated the knowledge graph
    c. Currently, the output of the predicted ICD codes for such a situation is still wrong. Perhaps, it is due to it existing as individual nodes and there are no relations and also the embeddings may be far fetched in comparison.

9. Conclusion and Future Work

The above are all the attempts I have made to reach the current accuracy of 95%. However, there are still improvements to be made since it is still unsatisfactory for now. One of the methods is to customize cypher queries (similar to customising prompts) to obtain the desired output.

However, given the number of edge cases above, it might take awhile to continue improving the code such that the cases can be solved.