UCLouvain

École polytechnique de Louvain

Towards Semantic Interoperability in Resource-Limited Settings: A Free and Open-Source ICD-10 Coding Tool for OpenMRS

Authors: **Nora De Vleeschouwer, Emilie Gouthière**
Supervisor: **Sébastien Jodogne**
Readers: **Amaury Fierens, Jean Vanderdonckt**
Academic year 2025–2026
Master [120] in Biomedical Engineering, Master [120] in Computer Science and Engineering

Abstract

ICD coding is the process of assigning standardised diagnostic codes to clinical documents, such as discharge notes. It is essential for billing, epidemiological surveillance, and health data interoperability; however, it remains a largely manual process that is time-consuming, costly, and error-prone. This thesis investigates automated ICD-10-CM coding along two complementary axes: machine learning model development and open-source clinical deployment.

On the modelling side, two coding scenarios are addressed. For the short excerpt task, four lexical baselines and four neural models are evaluated on a new synthetic dataset of 2,567 examples generated from UMLS and MIMIC-IV, as no annotated dataset exists for this subtask. The results indicate that self-alignment pretraining is the primary driver of performance: SapBERT achieves a Mean Reciprocal Rank of 0.477, compared to 0.181 for the unaligned PubMedBERT and 0.373 for the best lexical baseline. For the full discharge note task, a progressive series of models culminates in a multi-task PubMedBERT encoder that jointly predicts ICD-10-CM codes at three granularity levels (chapter, 3-character, and full code), using label attention over chunk representations and a hierarchical prediction strategy. Despite operating at chunk level rather than word level, the model achieves a macro-F1 of 0.123 and a recall@50 of 0.708 on the MIMIC-IV dataset. This means 70% of correct codes appear in the top-50 suggestions, which is the most relevant outcome for a coding-assistance workflow.

On the deployment side, we integrate both models into an OpenMRS module: ICDHelper. The module performs inference entirely offline using ONNX Runtime, resolves predictions against the CIEL concept dictionary, and surfaces code suggestions within the existing clinical workflow. It has been released under the Mozilla Public License v2 and has already been reviewed by members of the OpenMRS community, including Partners in Health and the director of the CIEL dictionary.

Disclaimer about generative AI usage

AI tools were used in the following aspects of this work: code documentation, debugging, scientific literature search (Undermind), and refinement of phrasing and formulation.

AI was not used for: drafting the original text, developing the core ideas and scientific reasoning, interpreting results, or drawing conclusions. All key ideas, arguments, and scientific judgments are grounded in the reviewed literature and reflect our own thinking.

Acknowledgement

First, we would like to thank Pr. Sébastien Jodogne, who initiated this thesis and supervised us throughout the year, for giving us the opportunity to work on this interesting subject combining health interoperability and machine learning. We would also like to thank him for his availability to answer our many questions, as well as for his advice and feedback, along with the freedom he gave us to explore this field.

Second, we would like to thank Pr. Jean Vanderdonckt for his interest in our early presentation during the November poster session, and for having accepted to be part of our jury.

Third, we would like to thank Amaury Fierens for his detailed answers to our questions and his involvement in our work throughout this year. His support was of great help, and his comments on the first version of this manuscript provided valuable insights.

Fourth, we would like to thank the OpenMRS community as a whole for welcoming us, and show our gratitude to all the members who kindly took the time to give constructive feedback on our module.

Finally, we would like to thank both of our families for their constant support. We would also like to specially thank Nora's dad and Emilie's aunts Claire and Anne-Marie for proofreading.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

Contents

Chapter 1

Introduction

The International Classification of Diseases and Related Health Problems, commonly referred to as ICD, is the global standard used to record diagnoses and health conditions for clinical, administrative, and epidemiological purposes. Established under the authority of the World Health Organization (WHO), ICD provides a common language for the reporting and monitoring of diseases. It allows health information to be shared and compared consistently across institutions and countries [1].

ICD coding plays a central role in modern healthcare systems. It supports billing and reimbursement, along with hospital activity reporting. It also enables epidemiological surveillance and ensures data interoperability. ICD coding contributes to the use of electronic health records for medical research and public health analysis [1]. In practice, it consists of reading a clinical document and assigning the ICD codes that correspond to the diagnoses documented in the text, as illustrated in Figure 1.1.[2]
However, despite its importance, ICD coding is still often performed manually by trained coders. This process is labour intensive and time consuming. It is also prone to variability and human error, especially when dealing with long and complex discharge summaries [3]. These limitations explain why automatic ICD coding has been an active research topic for more than two decades. The goal is to reduce administrative burden while improving coding consistency and scalability. It is important to note that the objective is not to completely replace human coders, but rather to facilitate their work by integrating prediction models into their workflow.

From a machine learning perspective, automated ICD coding is generally formulated as an extreme multi-label text classification problem. A single clinical note, often a discharge summary, must be mapped to a subset of relevant codes among thousands of possible labels [4]. This task is particularly difficult for several reasons. Discharge summaries are long, heterogeneous documents that combine current diagnoses, past medical history, procedures, medications, and discharge recommendations within the same note. This means the signal relevant for coding is often mixed with substantial noise. Medical concepts can also be expressed through abbreviations, synonyms, spelling variants, and highly domain-specific phrasing. Beyond the text itself, ICD labels are highly imbalanced, with many rare codes appearing only infrequently in training data [5]. Finally, ICD itself follows a hierarchical structure. It moves from broad chapters to specific categories and then to full diagnostic codes. This means that the output space is not flat but medically organised.

Because of these challenges, the literature has progressively moved from statistical and feature-based approaches toward deep learning and transformer-based architectures. Earlier methods, such as TF-IDF and CNN, could exploit lexical regularities in clinical notes, but they struggled to capture context, long range dependencies, and fine semantic distinctions between related diagnoses. More recent models based on domain-specific pretrained language models, such as BioBERT, ClinicalBERT, and PubMedBERT, have improved the representation of biomedical and clinical language [6, 7, 8]. At the same time, studies in ICD coding have shown that domain-specific encoders alone are not sufficient. Strong performance also depends on handling long clinical documents, large label spaces, and label-aware prediction mechanisms [4]. These limitations have motivated the use of architectures that combine pretrained biomedical encoders with attention mechanisms and long

[1] World Health Organization. International classification of diseases (ICD). https://www.who.int/standards/classifications/classification-of-diseases, 2026. Accessed: 2026-02-13.

[2] This illustration was inspired by a figure in an article from Kang et al. [2]

document processing strategies.

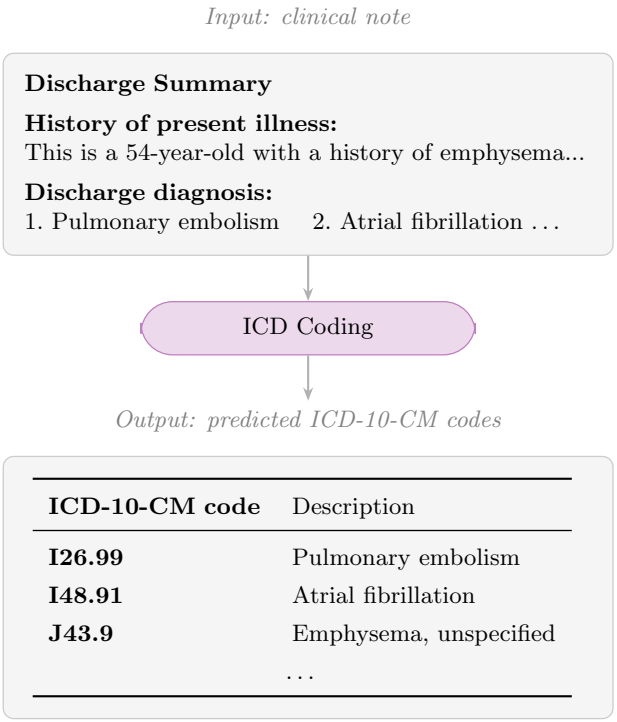


Figure 1.1: Overview of the ICD-10-CM coding task.

Central to all of these approaches is the label space: the International Classification of Diseases (ICD). Over time, the classification system has undergone several revisions. These revisions reflect medical progress and changes in healthcare practice. A major transition occurred with the publication of ICD-10 in 1990. It introduced fundamental structural changes to replace the aging ICD-9 framework. Beyond expanding the code space, ICD-10 shifted from a strictly numeric to an alpha-numeric coding format, which allowed for a more flexible and extensible hierarchy. This revision also improved the clinical logic of disease grouping [9]. The latest version of ICD-10 (2019) lists approximately 14,000 codes [10]. While ICD-10 remains the current standard in many regions, an 11th Revision (ICD-11) was adopted in 2019. It marks a definitive shift toward a fully digital, integrated healthcare framework, with even more codes and a global reorganisation of the hierarchical structure.

The structure of ICD-10 also shapes how the coding task is defined. In this work, we distinguish three levels of granularity as shown in Figure 1.2. The coarsest level is the chapter, one of the 22 broad disease families. A more precise level uses a 3-character code, which identifies a specific subcategory within a chapter. Finally, the full ICD-10 code provides the most detailed classification, pinpointing an exact diagnosis [10].
A notable adaptation of ICD-10 was curated by the National Center for Health Statistics (NCHS) to specifically respond to the United States healthcare needs [11]. It is called ICD-10-CM, for Clinical Modifications, and it stands out for having a much higher granularity, which was necessary in order to accommodate to the complex U.S. reimbursement system. The NCHS largely preserved the chapter and category structure of ICD-10, extending it downward rather than reorganizing upward. Contrarily to ICD-10, it is updated every year in October.

(a) All the chapters in ICD-10 (b) Example of 3-char codes in ICD-10 (c) Example of full codes in ICD-10

Figure 1.2: The three levels of granularity in ICD-10

This thesis is situated within that context. It focuses on automatic full ICD-10-CM coding from discharge notes in the MIMIC-IV dataset [12]. MIMIC-IV (Medical Information Mart for Intensive Care) serves as the global benchmark for clinical research. It is a massive, de-identified database containing real world electronic health records. It includes clinical notes, laboratory results, and diagnoses from thousands of patients admitted to intensive care units. This work addresses several major challenges in medical coding. It tackles the difficulty of mapping long clinical narratives to thousands of different codes. It also manages a highly imbalanced dataset where some diseases appear much more often than others. Beyond technical performance, there is a practical need for an open source solution. Therefore, this thesis also tackles integration of the models into OpenMRS based workflows. OpenMRS is an open source electronic medical record platform. It is used extensively by humanitarian organisations and healthcare providers in resource constrained settings worldwide. This thesis aims to bridge the gap between high level artificial intelligence research and accessible technology. By creating a tool compatible with the OpenMRS ecosystem, our study provides a field ready solution. Ultimately, this design would help clinicians perform accurate medical coding in real world settings.

Technically, the proposed approach relies on fine-tuned PubMedBERT encoders. Standard language models are trained on general text like Wikipedia. But PubMedBERT has been "pre-trained" on millions of biomedical articles, which makes it able to grasp the nuances of professional medical terminology. The first model is specialised in predicting the ICD-10-CM code associated with a concise description. It uses SapBERT, a fine-tuned version of PubMedBERT that aligns synonymous medical terms into a unified vector space. The second model processes long clinical notes by breaking them into manageable segments, called "chunks". Then, they are analysed through a multi-label attention mechanism. This mechanism acts as a selective lens. It learns to identify and weigh specific fragments of a document that provide the most relevant evidence for each ICD-10-CM code.

The main contributions of this thesis are the following:

- **A comparative evaluation framework for ICD code retrieval from short description.** We introduce a synthetic evaluation dataset covering all ICD-10 chapters. This dataset is used to benchmark several models, that given a brief description such as *type 2 diabetes with complications*, retrieve the most likely ICD-10 code by comparing it against all code descriptions in a shared semantic space.

- **An ICD coding model handling full discharge notes.** The model reads the entire clinical document and predicts all relevant ICD-10 codes simultaneously, exploiting the hierarchical structure of ICD-10 to improve prediction of rare codes. This work prioritises ease of deployment in resource-constrained hospital settings over state-of-the-art predictive

performance. The goal is a practical, locally-runnable tool that coders can rely on, not a benchmark leader.

- **A new evaluation benchmark for MIMIC-IV ICD-10.** Existing benchmarks either risk data leakage by splitting at the admission level, or exclude rare codes through aggressive filtering. We introduce a patient-level split with a less restrictive filtering criterion, retaining 9,685 codes and ensuring near-complete code coverage across all splits.
- **ICDHelper, an open-source OpenMRS module.** Models for both tasks are integrated into a Java module that runs directly on the hospital server, with no internet connection or external API required at inference time. The module suggests ICD codes to the clinician, who can review and confirm them within the existing patient record interface.

The remainder of this thesis is organised as follows. Chapter 2 reviews the state of the art, covering approaches to both short excerpt coding and full note coding, as well as existing clinical deployment solutions. Chapter 3 defines the automated ICD coding task tackled and the objectives pursued throughout this thesis. Chapter 4 introduces the technical background required to understand the proposed architectures and the set of metrics used to evaluate the models. Chapter 5 details the datasets, models, and evaluation protocols used in this work. Chapter 6 presents and analyses the experimental results for both tasks. Chapter 7 describes the architecture and implementation of the ICDHelper module and its integration into OpenMRS. Chapter 8 discusses the limitations of this work and the most promising directions for future research. Chapter 9 concludes the thesis.

Chapter 2

State of the art

Research into automated ICD coding began gaining significant interest in the late 1990s. Early studies demonstrated that predicting codes from discharge notes was feasible using statistical methods. These initial approaches relied on feature-based models, such as k-nearest neighbor and Bayesian algorithms [13, 14]. The landscape shifted around 2017-2018 with the rise of deep learning. Researchers started applying recurrent and convolutional networks with attention mechanisms to full discharge summaries, followed by transformer-based models that further improved the performance [1, 15, 16].
In the following sections, we first examine the task of mapping short groups of words to ICD codes. We then focus on the more complex goal of predicting codes from full length clinical discharge summaries.

2.1 Group of words to ICD codes

A first step in mastering ICD coding is to focus on short groups of words rather than complete discharge notes. The goal is to predict the specific ICD-10 code associated with a brief phrase.
In this task, the input is usually a diagnosis term or a problem list entry. The system must normalise these compact phrases into the correct medical coding concept.
A practical exemple of this is similar to searching in a medical dictionary. A physician might identify a disease but describe it using different terms. This system normalises those variations into a standardised diagnosis in the patient's record. By doing so, it ensures that healthcare professionals everywhere understand the medical data and avoids any misinterpretation.

The following sections examine three families of approaches to this problem, ordered by increasing conceptual sophistication. Lexical methods treat coding as a string matching problem and require no external knowledge. Terminology guided systems instead use clinical ontologies such as SNOMED CT and UMLS to connect medical language with administrative codes. These structures act as a bridge between the informal words doctors use and the official ICD-10 labels. Finally, transformer based models abandon handcrafted resources. They learn semantic proximity from data alone. This progression highlights a major challenge in the field. To achieve better language coverage, models must rely more heavily on either expert made databases or massive amounts of labeled data.

2.1.1 WHO coding tool

The WHO (World Health Organization) developed a dynamic search interface designed to simplify medical indexing, called ICD-11 Coding Tool. Its functionality is based on three main pillars, the first two visible in Figure 2.1.
First, Predictive Input generates a word list on the left as the user types. It completes terms and suggests related words based on their frequency in the ICD-11.
Second, the central section displays Destination Entities, such as codes and labels. They are ranked by relevance. Specific icons on the side signal additional details, like coding notes or the availability/need for "post-coordination" (combining multiple codes). More details are provided when clicking on a result, along with the possibility to see the suggested code inside the hierarchy.

Finally, if no exact result is found, a "Flexible" mode suggests broader variants or synonyms. For example, it might replace a specific term with a parent category. A side panel also allows users to view results by chapter or restrict the search to specific medical fields [17].

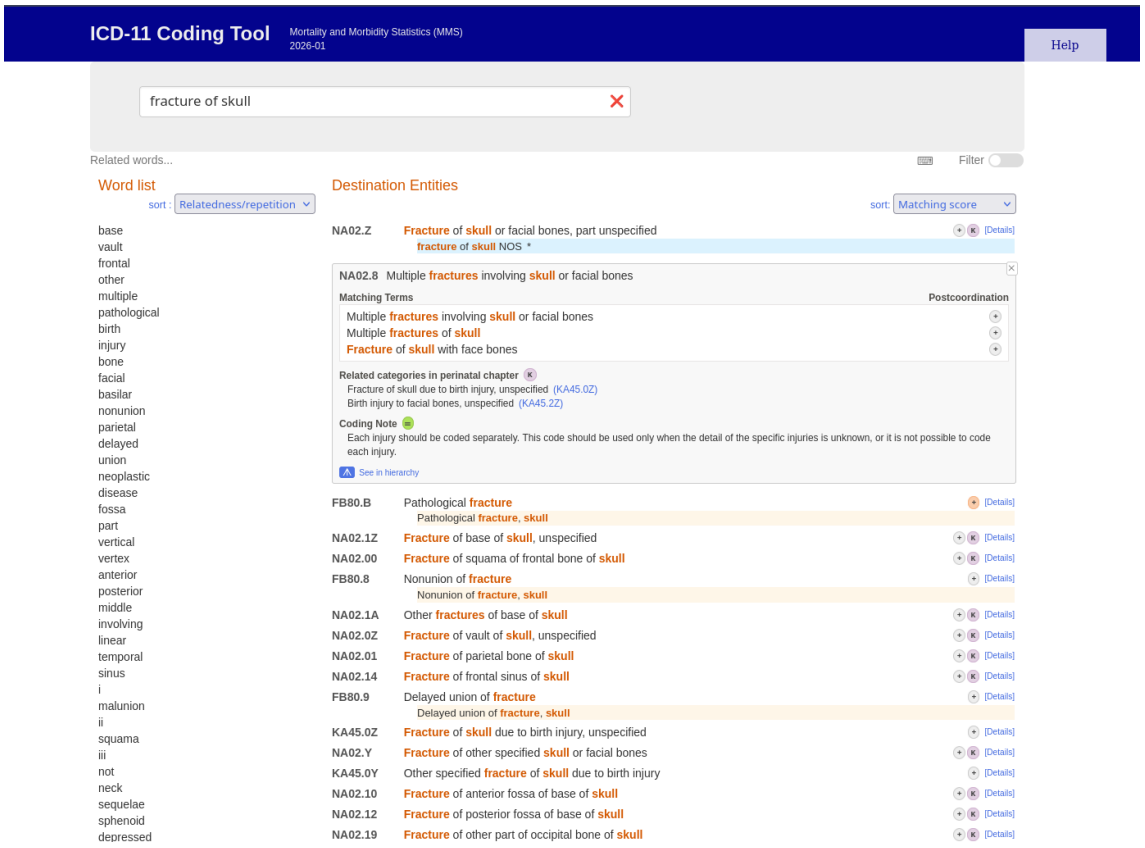


Figure 2.1: Example search in WHO ICD-11 coding tool

The WHO also offers a REST API exposing the same search capabilities as the Coding Tool, including word completion, flexible search, and chapter filtering. These search features are only available for ICD-11; the API's ICD-10 support is limited to direct code lookup by URI.[1]
The ICD-11 Coding Tool and API are freely accessible but unfortunately not released as open-source software. Therefore, we do not know what algorithms and models are used behind the screen.

2.1.2 Lexical similarity

As a first approach to the group of words to ICD code task, we examine methods based on lexical similarity. Rather than understanding the meaning of a phrase, these techniques treat text as a sequence of characters or tokens. They measure how similar two strings look on the surface. In other words, they ask: how much do these two pieces of text resemble each other visually? This family of methods is known as string matching.

One common approach is the Levenshtein distance. This metric counts the minimum number of character changes (insertions, deletions, or substitutions) needed to turn one string into another. Another popular method is the Dice-Sørensen coefficient. This measure calculates the overlap between two strings by comparing the specific pairs of characters they share. In practice, this family also includes n-gram similarity. A n-gram is a contiguous sequence of n items from a given sequence of text. This is based on the assumption that, in a sequence of text, the n-th word depends only on the last n-1 word that precedes it. If the appearance of one term merely relies on the last one word that precedes it, we call it bigrams ($n = 2$) [18]. These tools are effective for finding exact or near-exact matches in medical terminology.

[1] World Health Organization. ICD API *v2 Documentation (Swagger Index)*. https://id.who.int/swagger/index.html.Accessed:2026-04-16, 2024. Accessed: 2026-04-17

A more structured variant of string matching is the Longest Common Subsequence (LCS). Rather than comparing adjacent characters, LCS identifies the longest series of characters that appear in the same order in both strings, even if they are not contiguous. This makes it more robust to insertions or rewordings than simpler character level metrics.
A subsequent study improved this approach by integrating semantic similarity into the calculation, using a medical ontology to recognise synonyms and near-synonyms within sequences (2017) [19]. The authors also introduced a new weighting formula called W-LCS. This formula increases the weight of the common subsequence relative to the total string length, creating more distinct scores between different candidate diagnoses.

To evaluate the model, the authors used 2,036 real clinical documents from a partner hospital. Two professional coders manually assigned ICD-10 codes to ensure a reliable test base. A code was only included in the reference set if both experts agreed on the result.
The study compared three similarity measures on a sample of 1,000 hepatitis-related diagnoses. The goal was to test the algorithm's ability to produce a "strong" score for a true match while ignoring "noise." The results were the following:

- **Classic LCS**: A simple ratio between the common sequence and the longest string. It achieved an average similarity score of **0.60**, which was considered too conservative as it penalises length differences heavily.
- **Twofold LCS (T-LCS)**: A ratio based on the combined lengths of both strings. It improved the average score to **0.72**, but it remained insufficient for many complex diagnoses.
- **Weighted LCS (W-LCS)**: The authors' proposed method. It achieved the highest average similarity score of **0.78**, providing the most discriminant results between candidate diagnoses.

The table 2.1 compares the performance of W-LCS against other standard NLP methods. Precision captures the cost of false alarms (predicting codes that are not relevant), while recall captures the cost of omissions (missing codes that should have been predicted). F-score is their harmonic mean, which penalises models that sacrifice one for another. And here word matching means if two words are exactly the same, their similarity is 1, otherwise the similarity is 0 [19].

Table 2.1: Performance comparison of the algorithms (F-score) on 2,036 real clinical documents from a partner hospital

Method	Precision	Recall	F-score
W-LCS	**82,1%**	**80,2%**	**81,1%**
Bigrams (n=2)[18]	81,9%	79,3%	80,6%
Word Matching (Exact)	78,1%	74,1%	76%

As the results demonstrate, exact Word Matching methods show the weakest performance. This confirms that ICD coding cannot rely on simple text searches. Clinicians frequently use synonyms, abbreviations, or make typing errors that these methods cannot catch.
The Bigram approach ($n = 2$) achieves a respectable score. However, it remains limited because it only captures local dependencies between two adjacent words. The W-LCS algorithm outperforms these methods by combining sentence structure with targeted medical expertise. It uses a medical ontology to resolve synonyms during preprocessing, paired with a quadratic length-normalisation formula.

2.1.3 Terminology-guided normalisation

Clinical notes rarely use the exact terms found in ICD-10. Clinicians write in natural language, using abbreviations, synonyms, medical jargon, and regional variants that do not map directly to administrative code labels. Terminology-guided normalisation addresses this gap by first mapping free text to a structured clinical vocabulary, such as SNOMED CT[2] or UMLS before linking those normalised concepts to ICD codes. SNOMED CT is a comprehensive clinical terminology used to encode health information and improve patient care [20], while UMLS integrates many biomedical vocabularies to enable interoperability [21]. Both contain hundreds of synonyms for a

[2]SNOMED International. What is SNOMED CT. https://www.snomed.org/what-is-snomed-ct. Accessed: 2026-03-21.

single concept, making them far better suited to capture the variability of clinical language than the rigid labels of ICD-10.
The system proposed by Nguyen et al. (2018) uses this approach by transforming word groups into normalised concepts from SNOMED CT and UMLS, which are then mapped to ICD-10 codes using the NLM UMLS Metathesaurus and CSIRO Inhouse Maps [22].

Table 2.2 details the system's effectiveness based on the terminologies used and the code granularity. Mapping through SNOMED CT offers higher sensitivity compared to UMLS. For full codes, it reaches 54.1% versus 41.9%. This is crucial for capturing all diagnoses, even if it slightly increases false positives. Performance improves by approximately 10% when targeting 3-character disease groups. This shows the system identifies general pathologies well but struggles with the extreme precision of 4th or 5th characters. Finally, filtering out long or non-acute stays significantly improves the F-measure. This proves that coding quality depends heavily on the relevance of the processed notes.

Table 2.2: Overall effectiveness of the CAC (Computer assisted coding) system (Extracts from Tables 3 and 4) on a private dataset of the Gold Coast Hospital and Health Service (GCHHS) in Australia

Model and Filter	**UMLS to ICD-10-AM**			**SNOMED CT to ICD-10-AM**		
	Precision	**Recall**	**F1-score**	**PPV**	**Sens.**	**F-mes.**
Full codes						
All stays	72,6%	36,5%	48,6%	69,6%	47,0%	56,1%
Complete Filter*	73,1%	41,9%	53,3%	70,2%	54,1%	61,1%
3-char codes						
All stays	76,4%	46,0%	57,4%	73,2%	58,4%	65,0%
Complete Filter*	76,8%	51,9%	61,9%	73,7%	65,9%	69,6%

*Note : Full filtering includes Care Type, Length of Stay (LOS), and Admission Type. The baseline sensitivity for manual validation is 76.3%.

2.1.4 Static word embeddings and semantic similarity

Before the rise of transformer-based models, semantic similarity in biomedical NLP was approached through static word embeddings. The core idea, introduced by Word2Vec (2013) [23] and later refined by GloVe (2014) [24], is to represent each word as a dense vector, after training on large text corpora, such that semantically related words end up close together in the vector space.
In the biomedical domain, this approach was adapted to capture the specificity of clinical and scientific language. BioWordVec (2019) [25] and BioSentVec (2019) [26] extended these ideas by training on PubMed abstracts and clinical notes from MIMIC-III, producing embeddings that better reflect biomedical terminology. BioSentVec in particular introduced sentence-level representations, allowing the comparison of short clinical phrases rather than isolated words.
However, static embeddings have a fundamental limitation: each word has a single fixed vector, regardless of context. This context insensitivity makes them poorly suited for clinical text, where meaning often depends heavily on surrounding context. This limitation motivated the shift toward contextual, transformer-based representations.

2.1.5 SapBert

Transformer-based models address both limitations at once. Unlike static embeddings, they produce contextual representations where the meaning of a word depends on its surrounding text. Unlike terminology-based approaches [22], they do not rely on rigid manual mappings or exhaustive synonym lists.
SapBERT manages synonyms by learning the semantic proximity between words (2021) [27]. It builds an abstract representation space through a learning procedure that positions semantically similar entities close to each other. Consequently, SapBERT understands that "infarct" and "heart attack" are related, even if exact matches are missing. This model captures deep semantic similarity where lexical methods fail.
SapBERT acts as an encoder to produce mathematical embeddings (2024) [28]. It transforms word groups and terminology descriptions into dense vectors. While originally used for SNOMED-CT,

this approach can be applied to ICD-10 by encoding official code descriptions.

Once all 350,000 SNOMED CT descriptions have been encoded into vectors using SapBERT, the system encodes the input word group in the same way. FAISS is then used to find the code description whose embedding is closest to the input embedding [28]. This similarity search is performed across all codes at once, making it both exhaustive and efficient.
FAISS is an open source library developed by Meta for efficient vector clustering and searching. [3]. Specifically, the IndexFlatL2 approach stores all vectors in memory and uses brute-force search to find the nearest neighbors via Euclidean distance.

Results show significant efficiency, especially on the English n2c2 dataset. The system outperforms classic dictionary methods. For entity recognition, it achieves an F1-score of 58,2% compared to a 50,6% baseline. In normalisation (MCN), it reaches 35,3% versus 28%. The model performs exceptionally well in the "Disorders" category, with a normalisation F1-score of 87,1% [28].

The primary advantage of this approach is that as there is no need for fine-tuning; it requires no additional annotated training data. SapBERT and FAISS are both readily available, making the system straightforward to deploy in a hospital setting. This drastically lowers the barrier to adopting coding assistance tools. Overall, the combination of SapBERT and FAISS demonstrates that deep medical meaning can be captured without expensive supervised learning.

2.2 Discharge notes to ICD codes

This section now addresses a more complex task: predicting ICD codes from full discharge notes. A discharge note, also called a discharge summary, is a clinical document written by a physician at the end of a hospital stay. It summarises the patient's reason for admission, the diagnoses made, the procedures performed, the treatments administered, and the follow up plan. These documents are typically long and contain a mix of structured and unstructured information.

Following the same progression as in the previous section, we move from simple statistical approaches to increasingly sophisticated models. For this task, the evolution is particularly pronounced: early statistical methods are quickly outpaced by neural approaches. We will trace this trajectory from the first sequence based neural models up to label aware transformer architectures.

Two structural properties of ICD coding are essential. They motivate many architectural choices. The first is the extreme length of discharge summaries. Clinical notes often contain thousands of words, spread across sections of very different relevance. Relevant evidence for a given code may appear anywhere in the note, from past medical history to the final discharge diagnosis. This makes it difficult for any model to process the full document at once without losing important information.

The second is the hierarchical structure of the label space. ICD codes are not flat and independent. They are organised from broad diagnostic chapters down to highly specific full codes. This structure is medically meaningful. A model that predicts the wrong full code but stays within the correct diagnostic family makes a less severe error than one predicting a completely unrelated code. Moreover, rare codes can benefit from the broader and more stable patterns of their parent categories. Several studies have shown that exploiting this hierarchy improves performance on rare codes and makes predictions more clinically coherent [15, 29, 30].
Both challenges are addressed progressively across the models reviewed in this section.

2.2.1 Early statistical approaches to automated ICD coding of discharge notes

A first approach consists of analyzing the statistical properties of discharge notes. One well-known method is **TF-IDF** (Term Frequency–Inverse Document Frequency). It measures how important a word is within a document relative to a larger corpus [31]. It combines two complementary components: term frequency, which measures how often a word appears in a document, and inverse document frequency, which penalises words that appear in many documents and are therefore less distinctive. This balance allows TF-IDF to highlight terms that are both frequent within a specific

[3]Meta Fundamentals of AI Research Welcome to Faiss documentation. https://faiss.ai/. Accessed: 2026-03-26.

document and distinctive across the corpus. It is a widely used tool for tasks such as search ranking, text classification, and keyword extraction. However, despite its computational efficiency, TF-IDF relies purely on word frequency and therefore presents three critical limitations when applied to ICD-10 coding.

Semantic shallowness. TF-IDF produces a representation that treats each term as an isolated token. This representation ignores the surrounding context. As a result, it cannot recognise that semantically equivalent clinical expressions refer to the same diagnostic concept [32].

Sensitivity to label skewness. ICD-10 codes follow a highly skewed distribution: in practice, over 80% of codes appear in fewer than 1% of clinical reports. Since TF-IDF relies entirely on term frequency statistics, it lacks sufficient signal to produce stable feature weights for these rare tail labels. It leads to systematically poor performance on infrequent but clinically important codes [32].

Insensitivity to document structure and negation. Discharge summaries are typically lengthy, heterogeneous documents. They combine sections of very different clinical relevance (e.g., Principal Diagnosis, Past Medical History, Social History). TF-IDF assigns weights uniformly across the entire document. It fails to distinguish a confirmed diagnosis from a negated finding (e.g., *"no sign of pneumonia"*), and treats clinically critical sections identically to peripheral background information.

To address these limitations, subsequent works moved towards deep distributed representations, such as CNN, RNN or LSTM.

2.2.2 Neural networks

In 2016, Nigam compared several architectures proposed to address the limitations of statistical approaches (TF-IDF): Logistic Regression (baseline), Feed-Forward Networks, RNN, LSTM, and GRU [33]. The key innovation, common to all those methods, was to treat clinical notes sequentially. They used the 20 most recent notes per patient to capture the evolution of the medical record over time. The results clearly favored deep learning. The GRU (Gated Recurrent Unit) achieved the best performance, reaching an F1-score of 0.42 on the 10 most frequent ICD-9 codes for the MIMIC-III dataset. Both LSTM (Long Short-Term Memory) and GRU are recurrent neural networks designed to process sequences step by step while selectively remembering or forgetting information over time. The GRU is a simplified variant of the LSTM: it achieves similar results with fewer parameters and faster training.

However, this result comes with a significant limitation: the problem was deliberately simplified to only 10 codes, whereas real world ICD coding involves over 14,000 codes. GRU also suffered from marked overfitting and struggled when scaled to larger code sets. This limited performance is not surprising given the nature of the input. Discharge summaries are extremely long and written in telegraphic style. They are full of abbreviations and symbols that do not follow standard grammatical structure.
Sequential models such as LSTM and GRU are designed to capture word order and syntax. When the input lacks meaningful order, they lose their advantage over simpler approaches.

The importance of this paper therefore lies less in its absolute scores than in its conceptual shift [33]. It framed ICD coding as a sequence modeling problem rather than a pure feature engineering task. It opened the way to models that could exploit word order and local context. This was especially relevant for clinical notes, where phrases such as "acute renal failure", "ruled out pneumonia" or "history of diabetes" could not be reliably interpreted through isolated token counts alone. The models fail in these cases because they ignore context, negation, and medical history, which completely change the meaning of a clinical term.

In parallel, convolutional models gained popularity as a simpler and faster alternative to recurrent networks. CNN (Convolutional Neural Networks) apply small sliding filters over the text to detect local patterns. They detect patterns such as clinically meaningful word combinations, regardless of their position in the document. This made CNNs faster and more scalable than LSTM chains. Chung-Chian Hsu et al. (2020) benchmarked several architectures, including SVM, CNN, LSTM,

GRU, and HAN [34]. CNN outperformed all others. It reached a micro-F1 of 76% on the label-to-chapter task, and between 57.5% and 67.4% on the top-50 most frequent ICD-9 codes from the dataset MIMIC-III, clearly surpassing more complex models such as LSTM and GRU.

However, both recurrent and convolutional models had structural limits. LSTMs struggled to retain useful information over very long notes. CNNs captured local patterns well but missed evidence scattered across distant sections of the document. In both cases, compressing a full note into a single vector was too crude for a task where one document might support 10 or more codes, each grounded in a different part of the text. These limitations set the stage for the rise of attention mechanisms.

2.2.3 Attention

The key limitation of both recurrent and convolutional models was their reliance on a single document vector. In a long discharge summary, not every sentence is equally relevant, and different diagnoses are supported by different parts of the text. Compressing everything into one vector forces the model to mix signals that should remain separate.

Attention mechanisms were introduced to address this problem. Instead of treating all words equally, an attention mechanism assigns a weight to each word or sentence, reflecting how relevant that part of the text is for a given prediction. In practice, the model learns to focus on the most informative fragments and ignore the rest. This is comparable to a human reader who skims a long report and only carefully reads the sections that matter for a specific question.
However, a single attention mechanism still produces one document representation shared across all codes. In the multi-label setting of ICD coding, this is insufficient. A code for heart failure should not attend to the same evidence as a code for urinary tract infection or post operative complication. Label-specific attention was introduced to solve this issue. Each ICD code is assigned its own attention weights, allowing the model to read the same document differently for each diagnosis

The most influential formulation of this idea is CAML (2018) [16]. It combines a CNN encoder with label-dependent attention, allowing the model to select the most relevant text segments for each code independently.
At the time of publication, CAML outperformed all existing models on MIMIC-II and MIMIC-III. On MIMIC-III (Full), it reached a Micro-F1 of 53,9% and a Macro-AUC of 89,5%.
Beyond performance, CAML introduced a first practical form of explainability by highlighting which parts of the note were responsible for each prediction. In a clinical setting, this is a fundamental requirement. A model that assigns a code without justification is a black box that clinicians cannot audit, correct, or trust. Regulatory frameworks and medical practice both demand that automated coding decisions be traceable to specific evidence in the source document. The attention weights produced by CAML provide a direct link between each predicted code and the text fragments that support it.
LAAT (2020) [35] pushed this idea further. Instead of a CNN, it used a bidirectional LSTM (BiLSTM) to better capture long range dependencies in lengthy discharge summaries. Its key contribution was a label specific attention matrix, where each ICD code has its own attention weights. This allows the model to read the same document differently for each diagnosis. On MIMIC-III (Full), LAAT achieved a Micro-F1 of 57.5%, a P@5 of 81.3%, and a P@8 of 73.8%.

AttentionXML [36] pushed this idea even further by tackling a more extreme version of the problem: label spaces that can reach hundreds of thousands of codes. Its two key contributions build directly on the label-specific attention principle established by CAML and LAAT. First, it applies label-specific attention directly on top of the text encoder, allowing each label to independently focus on the most relevant parts of the document. Second, to handle such massive label spaces efficiently, it organises labels hierarchically through a probabilistic label tree. It groups related codes together before making final predictions.
Together, these models redefined how ICD coding should be structured. Rather than first summarizing a note and then predicting all codes from that summary, the model should learn a separate view of the document for each label. This shift made attention the central organizing principle of the field, and directly set the stage for transformer based approaches.

2.2.4 Transformers

The transition to transformer-based architectures did not abandon the attention logic developed in the previous section. It strengthened it. Earlier models such as CAML and LAAT introduced label-specific attention, but relied on relatively weak encoders like CNNs or LSTMs.
Transformers replaced these encoders with much richer representations. A transformer is a neural network architecture that processes an entire sequence of words simultaneously. It uses attention to weigh the relationship between every pair of words in the text. This allows it to capture the meaning of a clinical term based on its surrounding context, the section it appears in, and the way diagnoses are phrased.
One of the most popular transformer model is BERT (Bidirectional Encoder Representations from Transformers), which reads text in both directions at once and was pretrained on large general purpose corpora such as Wikipedia and books.

A key difference from earlier approaches is how words are represented. Previous models used static embeddings such as Word2Vec (2013) [23] or GloVe (2014) [24], where each word always has the same vector regardless of context. Transformers produce contextual representations: the word *cold* means something different in "the patient has a cold" and "cold extremities". A transformer can capture this distinction from context

However, applying transformers to ICD coding was not straightforward. Three specific challenges required dedicated solutions: the *length problem* — discharge summaries far exceed the 512 token limit of standard encoders; the *label space problem* — ICD-10 contains over 155,000 codes, many of which are extremely rare; and the *vocabulary problem* — general purpose models were not trained on clinical or biomedical text. The following papers each tackled one or more of these challenges.
The *length problem* became particularly acute with transformer encoders. Standard encoders such as BERT are constrained to a maximum of 512 tokens, making it impossible to feed it the full note directly.
The literature has converged toward two main strategies. The first splits the note into independent chunks. Then, it encodes each chunk separately and aggregates the results. The second replaces standard encoders with long context transformers such as Longformer or BigBird. These encoders can directly process much longer sequences.
The models presented below illustrate how the field progressively refined the first strategy. It moves from simple max pooling aggregation toward more structured label aware mechanisms.

Trans-ICD (2021) [37] primarily addresses the *length problem* and the *label space problem*. The note is split into chunks and encoded by a transformer combined with label wise attention. This paper makes two important points. First, the label specific attention idea from CAML and LAAT remained valuable even after moving to transformers. Second, better contextual representations alone are not sufficient: the model still needs a separate view of the document for each code.
The results on MIMIC-III Top-50 confirm this clearly. As shown in Table 2.3, a plain transformer without attention had micro-F1 performance below CAML. Adding attention brought the score up by only 3%. The full TransICD model, which also includes an LDAM loss to handle rare codes, reached Micro-F1 of 64,4%, outperforming all previous models. LDAM (Label-Distribution-Aware Margin) is a loss function that imposes a larger penalty for misclassifying rare codes during training, forcing the model to remain accurate even on infrequent diagnoses.

However, TransICD was trained from scratch on MIMIC-III, which is expensive and requires large amounts of data. A more efficient strategy is to start from a pretrained language model and adapt it to the clinical domain.

BERT-XML (2020) [38] addressed this matter by starting from a pretrained BERT encoder. Then the model is based on AttentionXML to handle the *label space problem*[36], see Section 2.2.3.
BERT-XML adapted this framework to long clinical documents by addressing the *length problem* through chunking. The document is first split into independent chunks. Each chunk is encoded separately by BERT. Then, for each code, a label specific attention mechanism identifies the most relevant tokens within each chunk.
A max pooling step then aggregates the predictions across all chunks to produce the final score.

This approach has one structural limitation. By processing chunks independently, the model cannot capture relationships that span across different parts of the document. For example, if a

condition is suggested in one chunk but only confirmed in a later one, BERT-XML treats these as separate pieces of evidence and may miss the connection.

Table 2.3: CAML versus transformer-based models on MIMIC-III Top-50. Micro metrics are dominated by frequent codes; macro metrics give equal weight to all codes and better reflect performance on rare ones.

Models	**AUC**		**F1**		**P@5**
	Macro	*Micro*	*Macro*	*Micro*	
CAML [16]	87,5%	90,9%	53,2%	61,4%	60,9%
Transformer	85,2%	88,9%	47,8%	56,3%	56,5%
Transformer + Attention	88,2%	91,1%	49,4%	59,3%	59,6%
TransICD (Best)	**89,4%**	**92,3%**	**56,2%**	**64,4%**	**61,7%**

PLM-ICD (2022) [4] addressed this limitation with segment pooling. The note is still split into chunks to address the *length problem* and each chunk is encoded separately by BERT. However, instead of making a prediction from each chunk independently, all chunk representations are concatenated into a single sequence before the label aware attention is applied.
This means the attention mechanism can now look across the entire note at once, rather than being confined to a single chunk. A condition mentioned in the first chunk and confirmed in the last chunk can therefore contribute jointly to the same code prediction.
For the *label space problem*, it used a label aware attention layer inspired by LAAT. Each code independently searches for its own evidence across the full document.

For the *vocabulary problem,*, it replaced general purpose BERT with domain specific encoders. An example of domain specific encoders is BioBERT (2020), which is based on BERT and fine-tuned on PubMed abstracts and PMC data. This improves its understanding of medical language [8].
ClinicalBERT (2019) adapted BERT to hospital text using MIMIC-III clinical notes, making it more sensitive to the writing style found in discharge summaries [7].
PubMedBERT (2021) went one step further by training an entirely new model from scratch on biomedical corpora. It shows that full domain pretraining can outperform simple adaptations [6]. RoBERTa (Robustly Optimised BERT Pretraining Approach) is a variant of BERT that uses the same architecture but was trained on more data, for longer, and without the next-sentence prediction objective that BERT originally used. These changes consistently improve downstream performance. RoBERTa-PM is a version of RoBERTa pretrained specifically on PubMed abstracts, combining the architectural improvements of RoBERTa with biomedical domain knowledge.
PLM-ICD used PubMedBERT and RoBERTa-PM as its encoders. The combination of these three improvements pushed PLM-ICD to state-of-the-art performance, as shown in Table 2.4: a Micro-F1 of 59.8% and a P@5 of 84.4% on MIMIC-III Full, outperforming both CAML and LAAT across all metrics.

Table 2.4: Evolution of ICD coding performance on MIMIC-III Full (CAML, LAAT, PLM-ICD)

Model	**F1-score (%)**		**Precision@k (%)**		
	Macro	*Micro*	*P@5*	*P@8*	*P@15*
CAML (2018)	8,8	53,9	70,9	56,1	-
LAAT (2020)	9,9	57,5	81,3	73,8	59,1
PLM-ICD (2022)	**10,4**	**59,8**	**84,4**	**77,1**	**61,3**

More recent work pushed this line further by injecting structured medical knowledge directly into the model. GKI-ICD (2025) [39] addresses this by injecting three complementary sources of structured knowledge directly into the training process.
For each clinical note, the model constructs a synthetic guideline sequence. It aggregates the official textual description of each ICD code, clinical synonyms retrieved from UMLS, and the descriptions of parent codes extracted from the ICD hierarchy. This guideline captures the medical meaning of each label in a clean, noise free form.

The architecture follows the same general pattern as PLM-ICD. The key innovation lies in the

training objective. Instead of a single classification loss, the model optimises three terms simultaneously. The first is a standard classification loss on the real clinical note. The second applies the same classification loss on the synthetic guideline, forcing the model to anchor its label representations to official medical definitions. The third is a cosine similarity loss that pushes the evidence vector extracted from the clinical note closer to the evidence vector extracted from the guideline. This alignment acts as a regulariser. It prevents the model from learning false correlations in clinical text, and provides rich supervision for rare codes that have very few real training examples. Importantly, this knowledge is fully absorbed into the model weights during training, so no additional parameters or external resources are needed at inference time.

2.2.5 Pipeline

The previous sections focused on two core questions: how to encode a clinical note into a meaningful representation, and how to identify which parts of that note support each specific ICD code. A third and equally important question is how these components are assembled into a complete coding pipeline. The literature shows three broad architectural strategies, each reflecting a different trade off between performance, explainability, and complexity.

Direct classification. The simplest strategy is to feed the full note into a transformer based model and directly predict all codes from the output representations. Models such as PLM-ICD follow this pattern [4].
The encoder produces contextual token representations. Then, label aware attention selects the relevant fragments for each code, and a classifier produces the final predictions. This approach is computationally efficient and achieves strong performance on standard benchmarks. In practice, the top predicted codes can be selected by keeping those with the highest probability scores above a fixed threshold.

NER and NEL pipelines. A second strategy decomposes the task into two explicit steps. Named Entity Recognition (NER) first identifies clinically meaningful spans in the note, such as diagnoses, symptoms, or procedures. Named Entity Linking (NEL) then maps each detected span to a normalised medical concept or a candidate ICD code. NEL is closely related to the terminology-guided normalisation introduced earlier (Section 2.1.3). They both aim to bridge the gap between the free form language of clinical notes excerpts and the structured vocabulary of a reference terminology. The key difference is that terminology-guided normalisation operates on the full document using predefined mappings, whereas NEL operates on specific text spans identified by NER, making each prediction directly traceable to a fragment of the note. This pipeline is attractive because it makes the coding path easier to inspect. Instead of relying on opaque document level attention, the model explicitly shows which text spans support which codes. Hierarchical NER then NEL strategies with in-domain transformers have shown strong results on explainable coding benchmarks (2023) [40].
More recent work goes one step further by framing the entire problem as entity linking (2025) [41]. Rather than classifying a full document into a set of codes, the model is given a specific clinical mention, such as "type 2 diabetes", and must link it to its corresponding ICD-10 code. This makes each prediction directly traceable to a specific fragment of the note. The authors tested three models based on BioMistral, a transformer decoder pretrained on biomedical text. The first model constructed from BioMistral uses in context learning, where a few examples are provided in the prompt without any retraining [41]. The second model uses supervised fine-tuning on thousands of mention-to-code pairs. These two models work on individual mentions that are already identified in the text. The third model, named INSGENEL, is different. It processes the entire document at once and rewrites it by inserting ICD codes next to each detected condition. It handles both detection and linking in a single pass. This makes it the closest to a true end-to-end coding system. To prevent the model from generating invalid codes, a constrained decoding mechanism forces the output to always form a valid ICD-10 code. The models were evaluated on CodiEsp, a dataset of Spanish clinical notes annotated with ICD-10 codes. Fine-tuned models clearly outperformed the prompt only approach, reaching a micro-accuracy of 66.85% on the dataset CodiEsp for INSGENEL, while the first model with the in context learning baseline scored only around 6-10%, confirming that ICD coding is too complex to solve with a few prompt examples alone.

However, NER and NEL are not the obvious first choice for broad full note ICD coding. Many ICD assignments depend on the entire document rather than a single mention. A discharge summary

contains dispersed evidence across history, assessment, procedures, and discharge plans. Some codes are inferred from combinations of findings or temporal context rather than a neat one-to-one mention mapping. This is one reason why direct concept extraction did not consistently improve document level coding performance in controlled experiments [42].

Candidate generation and reranking. A third strategy tackles the large label space directly through a two-stage pipeline. Rather than predicting a code among thousands of possibilities at once, a retriever first narrows the space of possible codes to a manageable set of candidates. A reranker then scores each candidate more carefully using richer interactions between the note and the code description. This design is well suited to ICD coding because the label space is very large and strongly long tailed. Most codes appear rarely, and a single stage classifier struggles to recover them.

A hybrid retrieval approach (2024) [43] combines three sources of evidence to reduce the space from approximately 9,000 codes to around 1,300 candidates. First, auxiliary clinical data already present in the record are used to prioritise relevant code families. For example, if a patient is prescribed insulin, the retriever promotes diabetes related codes. Second, BM25, a standard text search engine, compares the words of the note against official ICD code descriptions.

The reranking step then uses three complementary components. Clinical Longformer, a transformer capable of processing up to 4,000 tokens, replaces standard BERT. It captures evidence scattered across long discharge summaries. A graph based module called Graphormer, models relationships between diagnoses, such as the link between obesity and hypertension. Finally, contrastive learning trains the model to bring the note representation closer to correct codes and push it away from incorrect ones in the vector space. This last point is conceptually close to the SapBERT and FAISS approach described in Section 2.1.5, which also relies on vector similarity to match clinical text to code descriptions.

The results on MIMIC-III Full confirm the value of this design. The model reaches a Precision@15 of 62.3%, compared to 56.1% for CAML, and shows particularly strong gains on rare codes that appear fewer than ten times in the training data. The study also shows that switching from ClinicalBERT (512 tokens) to Clinical Longformer (4,000 tokens) significantly improves performance, confirming that important diagnostic evidence is often found late in discharge summaries.

The FLASH-ICD framework (2023) [44] follows the same "retrieve then rerank" pattern with contrastive learning. Their retrieval step uses contrastive learning to shortlist the most probable codes. Then, the reranker processes the full note against each candidate using an efficient transformer variant designed for long documents.
On MIMIC-III Full, their pipeline reaches a Micro-F1 of 59%. They also show that adding this two stage structure improves existing base models. This suggests that candidate filtering is broadly beneficial, regardless of the underlying architecture.

Conclusion. These three strategies reflect a clear evolution in how the field has approached the structural challenges of ICD coding. Direct classification models are simple and effective but treat the entire label space at once.
NER and NEL pipelines offer better explainability but struggle with document level synthesis.
"Retrieve then rerank" systems currently achieve the strongest results on broad coding benchmarks by decomposing the problem into a manageable two stage process.
Across all three strategies, two recurring principles stand out: the importance of handling long documents and of using embeddings to represent both clinical notes and code descriptions. This allows similarity to be measured directly rather than relying on rigid word-by-word matching.

2.3 Clinical deployment and integration

Most of the models reviewed in this chapter were evaluated on benchmark datasets. However, their integration into real clinical workflows raises additional challenges. In practice, automated ICD coding tools must interface with existing hospital information systems to be usable by clinicians. The gap between research performance and clinical usability is a recurring concern in the literature.

A model that achieves strong benchmark results is not automatically ready for deployment. It must be accessible to clinicians, interpretable enough to support their decisions, and integrated into their existing workflow without disruption.

OpenMRS is an open-source electronic medical record platform. It was originally developed in 2004 through a collaboration between Regenstrief Institute and Partners in Health, and is now maintained by a global community of developers and implementers. It is designed with resource-limited healthcare settings in mind, and is actively deployed in over 40 countries across sub-Saharan Africa, Asia, and Latin America. Its architecture is built on the Java Spring framework with a Hibernate persistence layer, and it exposes a rich API that abstracts clinical data operations behind a service layer [45]. Its modular architecture allows external tools and models to be integrated as plugins or services. This makes it a natural deployment target for coding assistance tools.

Prior work (2015) has explored structuring free clinical narratives within OpenMRS through medical concept extraction [46]. The ICD-11 API has also been integrated into an OpenMRS-based national electronic medical record system in Rwanda (2025) [47], demonstrating that OpenMRS can support standardised diagnostic coding at a national scale. Although the literature on fully automatic ICD coding within OpenMRS remains limited, these studies show that OpenMRS is a realistic deployment target for coding assistance tools.

Chapter 3

Problem statement

Clinical coding is the process of translating medical information from patient records into standardised codes from a controlled terminology. It is a critical step in hospital administration, billing, and epidemiological surveillance. In practice, however, it is performed manually by trained coders. It is a process that is slow, costly, and prone to inconsistency. Automating this process is therefore a long standing challenge in clinical NLP.

This thesis addresses two related tasks within this broader problem. The first is linking a **group of words to ICD-10 codes**. Given a short clinical expression, the goal is to identify the most relevant ICD-10 code from a large catalogue. This task serves as an entry point into the broader coding problem. It allows us to explore the structure of the ICD terminology and evaluate retrieval approaches in a controlled, smaller scale setting before tackling the full complexity of discharge note coding.
The second, and more challenging, task is **automatic coding from full discharge notes**. Unlike short expressions, discharge notes are long, noisy, and written in a heterogeneous clinical language. A single note may justify dozens of codes simultaneously. This makes the task both a multi-label classification problem and a text understanding challenge. The hierarchical organisation of ICD codes, from broad chapters down to granular leaf codes is an additional structural property that can be exploited to guide the classification.

As shown in the state of the art, significant progress has been made over the past two decades. Yet most existing work focuses exclusively on benchmark performance, leaving a gap between research models and tools that can realistically be deployed in hospitals. Bridging this gap is a central motivation of our thesis.

We therefore pursue two objectives in parallel. The first is to build and evaluate competitive models for both tasks, addressing known challenges such as note length, large and imbalanced code sets, and the hierarchical structure of ICD.
The second objective is deployability. We aim to produce an **open source system that operates entirely offline**, without external API or cloud dependency at inference time. This system will be integrated into OpenMRS, a widely adopted open source hospital information system, making it accessible to institutions with limited infrastructure.
Particular attention is also given to the user interface. It must be practical and readable for clinical coders in their daily workflow.

In summary, the objectives of this thesis are the following:

- to build and evaluate a robust pipeline for group of words to ICD-10-CM codes;
- to train and compare models for automatic ICD coding from discharge notes, addressing the challenges of note length, large code sets, and clinical vocabulary;
- to exploit the hierarchical structure of ICD codes to improve classification;
- to deliver an offline, open source implementation integrated into OpenMRS, with a practical interface designed for real world clinical coding.

Chapter 4

Technical background

This chapter introduces the technical concepts and methods underlying the approaches developed in this study. We begin with classical text representations based on term statistics, then describe the Transformer architecture and its application to biomedical text through BERT-based models. We then present strategies for handling long clinical documents, and conclude with the evaluation metrics used throughout the experimental chapters.
Before text is fed into a model, words must be converted into numerical representations. Sparse, count based methods such as Bag of Words and TF-IDF represent documents as fixed size vectors over a vocabulary. While simple and interpretable, they ignore word order and treat semantically similar words as unrelated. Static embeddings such as Word2Vec [48] address the latter limitation by mapping words to dense vectors that capture semantic similarity, but a given word always receives the same representation regardless of its context. Contextual embeddings overcome this issue by generating token representations that vary with the surrounding context. This is the approach adopted in this thesis, described in Section 4.5.

4.1 TF-IDF

TF-IDF (Term Frequency–Inverse Document Frequency) is a classical statistical measure used to evaluate the importance of a term within a document relative to a collection of documents, called a corpus [31].
It combines two complementary signals: how frequently a term appears in a given document, and how rarely it appears across the corpus as a whole. A high TF-IDF score indicates that a term is both frequent in the document and distinctive with respect to the corpus.

Term frequency

The term frequency measures how often a term t appears in a document d, under the assumption that higher occurrence indicates greater relevance:

$$\mathrm{TF}(t, d) = \frac{n_{t,d}}{\sum_{t' \in d} n_{t',d}} \tag{4.1}$$

where $n_{t,d}$ denotes the raw count of term t in document d, and the denominator normalises by the total number of terms in d, ensuring that longer documents are not unfairly penalised.

Inverse document frequency

While term frequency captures local importance, common terms such as *the* or *is* tend to appear frequently in almost every document and carry little discriminative information. The inverse document frequency down-weights such terms while amplifying the importance of rare, informative ones:

$$\mathrm{IDF}(t, \mathcal{D}) = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in d\}| + 1} \tag{4.2}$$

where $|\mathcal{D}|$ is the total number of documents in the corpus and $|\{d \in \mathcal{D} : t \in d\}|$ is the number of documents containing term t. The $+1$ in the denominator is a smoothing term that prevents division by zero for terms that do not appear in the corpus.

Combined score

The final TF-IDF weight assigned to term t in document d is the product of both components:

$$\text{TF-IDF}(t, d, \mathcal{D}) = \text{TF}(t, d) \times \text{IDF}(t, \mathcal{D}) \tag{4.3}$$

This score is high when a term appears frequently in a specific document but rarely across the corpus, making it a useful signal for identifying the most characteristic terms of a document.

In practice, a collection of documents is represented as a **TF-IDF matrix** $\mathbf{X} \in \mathbb{R}^{n \times v}$, where n is the number of documents and v is the vocabulary size. Each entry $X_{i,j}$ corresponds to the TF-IDF score of term j in document i. This sparse matrix representation is commonly used as input to machine learning models such as logistic regression or support vector machines. However, TF-IDF treats each term independently and does not capture word order or semantic relationships between terms, which motivates the use of more expressive representations such as word embeddings or contextual models (Section 4.3).

4.2 Bidirectional GRU

The Gated Recurrent Unit (GRU) [49] is a recurrent architecture designed to model sequential dependencies while mitigating the vanishing gradient problem that affects standard RNNs. At each time step t, the GRU computes a hidden state $\mathbf{h}_t \in \mathbb{R}^d$ through two gating mechanisms that control the flow of information.

The **reset gate** $\mathbf{r}_t$ determines how much of the previous hidden state is used to compute the candidate state:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{4.4}$$

The **update gate** $\mathbf{z}_t$ controls how much of the previous hidden state is carried forward versus replaced by new information:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{4.5}$$

A candidate hidden state $\tilde{\mathbf{h}}_t$ is computed from the current input and the selectively reset previous state:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{4.6}$$

The final hidden state interpolates between the previous state and the candidate, controlled by the update gate:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{4.7}$$

where σ is the sigmoid function and $\odot$ denotes element wise multiplication. When $\mathbf{z}_t \approx 1$, the new candidate state dominates; when $\mathbf{z}_t \approx 0$, the previous state is preserved.
This mechanism allows the GRU to selectively forget or retain information over long sequences.
A **Bidirectional GRU** (BiGRU) runs two GRUs in parallel over the same sequence: one in the forward direction and one in the backward direction. At each position t, the two hidden states are concatenated:

$$\mathbf{h}_t^{\text{bi}} = \left[\overrightarrow{\mathbf{h}}_t \,\|\, \overleftarrow{\mathbf{h}}_t\right] \in \mathbb{R}^{2d} \tag{4.8}$$

where $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ are the forward and backward hidden states respectively. This allows each position to integrate context from both past and future elements of the sequence.

4.3 Transformer architecture

Neural models learn to map inputs to outputs through successive layers of parameterised transformations. This builds increasingly abstract representations of the input. In practice, it is common to distinguish between a backbone, the stack of layers responsible for producing rich, general-purpose representations and a task specific head, a lightweight module placed on top of the backbone that maps these representations to a target output space (e.g. a probability distribution over ICD codes). This distinction is central to the approach adopted in this thesis. A pretrained backbone is combined with a task-specific head and the entire model is fine-tuned end to end on the target task [50].

The Transformer [51] is the foundational architecture underlying BERT and its variants. It relies entirely on attention mechanisms, replacing recurrence and convolution. Formally, each Transformer layer takes a matrix $X \in \mathbb{R}^{n \times d}$ as input where n and d denote the sequence length and embedding dimension, respectively. Then, it produces an output $Y = \text{Attention}(Q, K, V)$ as described in Section 4.3.1 below. We describe its core components in turn.

4.3.1 Scaled dot-product Attention

Given a sequence of n tokens, each token is projected into three vectors: a query $\mathbf{q}_i \in \mathbb{R}^{d_k}$, a key $\mathbf{k}_i \in \mathbb{R}^{d_k}$, and a value $\mathbf{v}_i \in \mathbb{R}^{d_v}$. These are obtained by linear projections of the input:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q \tag{4.9}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^K \tag{4.10}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^V \tag{4.11}$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the input matrix and $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$ are learned weight matrices.
The attention output is then computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{4.12}$$

The dot product $\mathbf{Q}\mathbf{K}^\top$ computes a compatibility score between every pair of tokens. Dividing by $\sqrt{d_k}$ stabilises gradients by preventing the dot products from growing too large in high dimensions. The softmax normalises these scores into a probability distribution over the sequence, producing attention weights $\mathbf{A} \in \mathbb{R}^{n \times n}$:

$$A_{ij} = \frac{\exp\left(\mathbf{q}_i \cdot \mathbf{k}_j / \sqrt{d_k}\right)}{\sum_{l=1}^{n} \exp\left(\mathbf{q}_i \cdot \mathbf{k}_l / \sqrt{d_k}\right)} \tag{4.13}$$

Each output token representation is then a weighted sum of value vectors: $\mathbf{z}_i = \sum_j A_{ij}\mathbf{v}_j$. This allows every token to attend to every other token in the sequence, regardless of distance.

4.3.2 Multi-head Attention

Rather than computing a single attention function, the Transformer applies h attention heads in parallel, each operating in a lower-dimensional subspace:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \tag{4.14}$$

where $\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$ is a learned output projection. Each head can specialise in capturing different types of dependencies: syntactic, semantic, or positional. This specialisation is not architecturally enforced but emerges from training. Since each head operates in its own learned subspace with independent projection matrices, gradient descent naturally drives different heads to attend to different patterns when this leads to a lower loss [52].

4.3.3 Positional Encoding

Since the attention mechanism is permutation invariant, the Transformer architecture has no inherent notion of token order. Processing the sequence"the cat sat" and "sat cat the" would have identical attention outputs without additional information. To remedy this, positional information is explicitly injected by adding a positional encoding matrix $\mathbf{P} \in \mathbb{R}^{n \times d}$ to the input embeddings before they enter the model:

$$\mathbf{X}' = \mathbf{X} + \mathbf{P}, \tag{4.15}$$

where n is the sequence length and d the embedding dimension.
The approach uses fixed, deterministic encodings based on sinusoidal functions of varying frequency [51]:

$$P_{pos,\,2i} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \tag{4.16}$$

$$P_{pos,\,2i+1} = \cos\left(\frac{pos}{10000^{2i/d}}\right), \tag{4.17}$$

where $pos \in \{0, \ldots, n-1\}$ denotes the token position in the sequence and $i \in \{0, \ldots, d/2-1\}$ the dimension index within the embedding. Each dimension thus oscillates at a different frequency, allowing the model to distinguish both absolute positions and, through linear combinations, relative offsets between tokens.

An alternative approach, adopted by BERT [50], replaces these encodings with learned positional embeddings: a trainable matrix of the same shape $\mathbb{R}^{n \times d}$ is optimised jointly with the rest of the model. While both strategies yield comparable performance in practice, sinusoidal encodings have the advantage of generalising, in principle, to sequence lengths unseen during training.

4.4 Transformer variants

The Transformer architecture comes in three main variants, differing in how attention is applied across the sequence. Encoder only models such as BERT [50] process the full input sequence bidirectionally. Each token attends to all others simultaneously, producing rich contextual representations well suited to classification and retrieval tasks. Decoder-only models such as GPT [53] apply causal (left-to-right) attention, where each token can only attend to preceding tokens. This autoregressive constraint makes them natural text generators but limits their ability to build full-context representations. Finally, encoder-decoder models such as T5 [54] combine both approaches. The encoder processes the input bidirectionally, while the decoder generates the output token by token, attending to both its own previous outputs and the encoder representations. This architecture is particularly suited to sequence to sequence tasks such as translation or summarisation.

4.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) [50] is a Transformer encoder pre-trained on large text corpora.
BERT uses a pure encoder stack with full bidirectional self-attention. Every token attends to all other tokens in both directions simultaneously. This makes BERT particularly well suited to classification tasks where the full context of a sequence is available at inference time.

4.5.1 Pre-training and fine-tuning

BERT is pre-trained on unlabelled text using self-supervised objectives, most notably Masked Language Modelling (MLM), where randomly masked tokens must be predicted from their surrounding context. This allows BERT to learn deep bidirectional representations without requiring labelled data. After pre-training, BERT can be fine-tuned on a downstream task by adding a task-specific head and updating all parameters jointly.

4.5.2 Input representation

BERT uses a **WordPiece** [50] tokeniser to convert raw text into a sequence of subword tokens. Rather than operating on whole words, WordPiece splits rare or unknown words into smaller known subunits. For example, the clinical term *tachycardia* may be tokenised as *tachy* + `##`*cardia*, where `##` indicates a continuation of the previous token. This allows the model to handle out-of-vocabulary terms gracefully, which is particularly important in biomedical text where specialised terminology is frequent.

A special `[CLS]` token precedes the sequence, and a `[SEP]` token marks segment boundaries. A segment is a contiguous span of text treated as a unit, typically a sentence or a paragraph. This allows the model to distinguish between two segments when the input consists of a pair of texts, as in next-sentence prediction or textual entailment.
The embedding of each token is the sum of three learned components:

$$\mathbf{e}_i = \mathbf{e}_i^{\text{token}} + \mathbf{e}_i^{\text{pos}} + \mathbf{e}_i^{\text{seg}} \tag{4.18}$$

where $\mathbf{e}_i^{\text{token}}$ is the token embedding, $\mathbf{e}_i^{\text{pos}}$ is the positional embedding, and $\mathbf{e}_i^{\text{seg}}$ identifies the segment the token belongs to.

4.5.3 The [CLS] Token as a sequence representation

The [CLS] token precedes every input sequence. Unlike other tokens, it has no inherent lexical meaning. However, because BERT uses full bidirectional self-attention, the [CLS] token attends to every other token in the sequence across all layers. By the final layer, its hidden state $\mathbf{h}_{\texttt{[CLS]}} \in \mathbb{R}^d$ has aggregated information from the entire input sequence and serves as a sequence-level representation.

At layer l, the representation of [CLS] is updated as follows:

$$\mathbf{h}_{\texttt{[CLS]}}^{(l)} = \text{TransformerLayer}^{(l)}\left(\mathbf{h}_{\texttt{[CLS]}}^{(l-1)}, \mathbf{h}_1^{(l-1)}, \ldots, \mathbf{h}_n^{(l-1)}\right) \tag{4.19}$$

At every layer, [CLS] attends to all token representations from the previous layer. After L layers, $\mathbf{h}_{\texttt{[CLS]}}^{(L)}$ encodes a global summary of the sequence and is typically used as input to a classification head.

BERT's maximum input length is 512 tokens. Clinical notes such as discharge summaries, however, routinely exceed this limit: a typical discharge summary contains several thousand tokens, covering admission history, diagnoses, procedures, and follow-up instructions. Since ICD-10 codes may be supported by evidence scattered across different sections of the document, truncating the input risks discarding clinically relevant information. The text must therefore be split into shorter chunks, each encoded independently. Each chunk produces its own [CLS] representation $\mathbf{c}_k \in \mathbb{R}^d$. Aggregating these chunk-level representations into a single document-level vector is a non-trivial problem, and several strategies are described in Section 4.7.

4.6 Domain-specific BERT models

General-purpose BERT models are pre-trained on broad corpora such as Wikipedia and BookCorpus. However, biomedical and clinical texts exhibit a specialised vocabulary and writing style that differs substantially from general English. Domain-specific pre-training has been shown to yield significant performance gains on biomedical NLP tasks.

4.6.1 PubMedBERT

PubMedBERT [6] is pre-trained from scratch exclusively on PubMed abstracts and PubMed Central full-text articles, without initialising from a general-domain checkpoint. Its vocabulary is learned entirely from biomedical text, which avoids the fragmentation of domain specific terms that occurs when a general-purpose tokeniser is applied to biomedical content. PubMedBERT follows the BERT architecture: 12 Transformer layers, 12 attention heads, a hidden dimension of 768, and approximately 110M parameters[50].

4.6.2 Clinical ModernBERT

ClinicalModernBERT builds on ModernBERT [55], an updated encoder architecture that introduces two key improvements over the original BERT design.

First, absolute positional embeddings are replaced by Rotary Position Embeddings (RoPE) [56], which encode relative position information directly into the attention computation by rotating query and key vectors:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{(\mathbf{RQ})(\mathbf{RK})^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{4.20}$$

where $\mathbf{R}$ is a block-diagonal rotation matrix whose angle depends on the token position. RoPE generalises naturally to sequence lengths longer than those seen during training.

Second, ModernBERT alternates between local and global attention layers. Local layers restrict attention to a fixed window of neighbouring tokens, reducing the quadratic complexity of full self-attention, while global layers maintain full sequence-level context. This makes the architecture more efficient on long sequences. ModernBERT breaks the classic 512-token barrier. It allows for the processing of long-form documents with a context window of 8192 tokens.

ClinicalModernBERT is further pre-trained on clinical notes, making it particularly suited to discharge summaries and other clinical documents that contain abbreviations, structured sections, and non-standard language.

4.7 Aggregation strategies for long documents

When a document is split into K chunks, each independently encoded by a BERT model, the result is a sequence of chunk representations $(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K)$ where $\mathbf{c}_k \in \mathbb{R}^d$. These must be combined into a single document level vector $\mathbf{h} \in \mathbb{R}^d$ before classification. Several strategies exist for this aggregation step.
Once a document level representation $\mathbf{h}$ has been obtained through one of the strategies described above, it is passed to a classification head that produces a probability score for each ICD-10 code. The design of this classification head and the associated training objectives are described in Chapter 5.

4.7.1 Mean pooling

The simplest approach averages the chunk representations:

$$\mathbf{h} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{c}_k \tag{4.21}$$

This treats all chunks as equally important. It is computationally cheap and requires no additional parameters, but cannot focus on the most relevant parts of the document.

4.7.2 Recurrent aggregation

A recurrent network can be applied over the sequence of chunk representations to capture their ordering. Unlike mean pooling, this approach treats the chunks as a narrative sequence. Each chunk is processed in the context of those that precede and follow it.

A Bidirectional GRU (Section 4.2) processes the sequence in both directions. The forward pass reads from the first chunk to the last, while the backward pass reads in the reverse direction.
At each step k, the GRU updates its hidden state based on the current chunk representation $\mathbf{c}_k$ and the previous hidden state. This accumulates information from all previously seen chunks. The output at each position is the concatenation of the forward and backward hidden states:

$$\mathbf{h}_k^{\text{bi}} = \left[\overrightarrow{\mathbf{h}}_k \parallel \overleftarrow{\mathbf{h}}_k \right] \tag{4.22}$$

The final document representation is obtained by concatenating the last hidden states of both directions:

$$\mathbf{h}_{\text{doc}} = \left[\overrightarrow{\mathbf{h}}_K \parallel \overleftarrow{\mathbf{h}}_1 \right] \in \mathbb{R}^{2d_{\text{GRU}}} \tag{4.23}$$

where $\overrightarrow{\mathbf{h}}_K$ is the final hidden state of the forward GRU after reading all K chunks, and $\overleftarrow{\mathbf{h}}_1$ is the final hidden state of the backward GRU. This vector summarises the entire sequence of chunks and is passed to the classification head.

4.7.3 Label-wise Attention

Label wise attention [16] is a mechanism that computes a different document representation for each target label c. Each label is associated with a learned query vector $\mathbf{u}_c \in \mathbb{R}^d$, which is used to attend over the chunk representations:

$$\alpha_{ck} = \frac{\exp(\mathbf{u}_c^\top \mathbf{c}_k)}{\sum_{j=1}^{K} \exp(\mathbf{u}_c^\top \mathbf{c}_j)} \tag{4.24}$$

where α_{ck} is the attention weight assigned to chunk k for label c, reflecting how relevant that chunk is for predicting code c. The label specific document representation is then the weighted sum of chunk vectors:

$$\mathbf{h}c = \sum k = 1^K \alpha_{ck} \mathbf{c}_k \tag{4.25}$$

Rather than applying a simple linear projection on $\mathbf{h}_c$, the classification score for label c is computed through a two layer non linear transformation:

$$\hat{y}_c = \sigma\left(\mathbf{W}_a \mathrm{ReLU}(\mathbf{W}_b, \mathbf{h}_c)\right) \tag{4.26}$$

where $\mathbf{W}_b \in \mathbb{R}^{d \times d}$ projects the label-specific representation into an intermediate space, ReLU introduces non linearity, and $\mathbf{W}_a \in \mathbb{R}^{d \times 1}$ produces the final scalar score, which is passed through a sigmoid σ to obtain a probability. This two layer design follows the original CAML formulation [16] and gives the model more expressive power than a direct dot product.
This approach allows the model to focus on different parts of the document for different labels. For instance, the representation used to predict a respiratory diagnosis may attend more strongly to chunks describing pulmonary symptoms, while the representation for a metabolic condition attends to different chunks. This is particularly well suited to multi label settings where different labels are supported by different parts of the document.

4.8 Evaluation metrics

ICD-10 coding is a multi-label classification problem over a large and highly imbalanced label space. Standard accuracy is not informative in this setting: a model that predicts no codes at all would achieve high accuracy simply because most labels are absent for any given document.
The metrics reported in this thesis are chosen to align with those used in the automatic ICD coding literature [16]. This ensures direct comparability with prior and concurrent work. We follow their protocol in reporting micro- and macro-averaged F1 and precision at k, and extend it with recall at k to better reflect our assistive use case, where surfacing the correct codes within the top candidates is of primary importance.

4.8.1 Precision, Recall and F1-score

For a single label c, precision and recall are defined as:

$$P_c = \frac{TP_c}{TP_c + FP_c}, \qquad R_c = \frac{TP_c}{TP_c + FN_c} \tag{4.27}$$

where TP_c, FP_c, and FN_c denote true positives, false positives, and false negatives for label c respectively.
Precision measures the fraction of predicted labels that are correct.
Recall measures the fraction of true labels that are retrieved. The F1-score is their harmonic mean:

$$F1_c = \frac{2 \cdot P_c \cdot R_c}{P_c + R_c} \tag{4.28}$$

4.8.2 Micro and Macro Averaging

When evaluating over C labels simultaneously, two averaging strategies are commonly used.
Micro-F1 aggregates TP, FP, and FN across all labels before computing the score:

$$F1_{\text{micro}} = \frac{2\sum_{c=1}^{C} TP_c}{2\sum_{c=1}^{C} TP_c + \sum_{c=1}^{C} FP_c + \sum_{c=1}^{C} FN_c} \tag{4.29}$$

Micro-F1 is dominated by frequent labels, since they contribute more counts to the numerator and denominator.
Macro-F1 computes the F1-score for each label independently and then averages:

$$F1_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} F1_c \tag{4.30}$$

Macro-F1 treats all labels equally regardless of their frequency. It is therefore more sensitive to performance on rare codes, which is particularly relevant in ICD-10 coding where the label distribution is highly skewed. A small number of common codes account for the vast majority of assignments, while most codes appear only rarely.

Both metrics are reported in this thesis. Micro-F1 reflects overall system performance on frequent codes, while Macro-F1 captures the ability of the model to generalise across the full label space including rare codes.

4.8.3 Precision and Recall at k

In this work, the goal is not to produce a fully automated coding decision, but rather to assist human coders by surfacing the most relevant candidate codes from a large label space. Concretely, the model ranks all possible ICD-10 codes for a given clinical note, and the coder selects the correct ones from the top-k candidates. Precision at k (P@k) and Recall at k (R@k) are therefore particularly relevant metrics, as they evaluate whether the correct codes appear among the top-k predictions regardless of the exact decision threshold.
Given the top-k predicted codes for a document, P@k measures the fraction of those predictions that are correct:

$$\text{P@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{\mathcal{Y}}_i^{(k)} \cap \mathcal{Y}_i|}{k} \tag{4.31}$$

R@k measures the fraction of true codes that appear in the top-k predictions:

$$\text{R@}k = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{\mathcal{Y}}_i^{(k)} \cap \mathcal{Y}_i|}{|\mathcal{Y}_i|} \tag{4.32}$$

where $\hat{\mathcal{Y}}_i^{(k)}$ is the set of top-k predicted codes for document i, $\mathcal{Y}_i$ is the set of true codes, and N is the number of documents.
These metrics are threshold free: they evaluate the ranking quality of the model regardless of the decision threshold τ. R@k is particularly informative in clinical settings where missing a relevant code (false negative) is more costly than predicting a spurious one.

4.8.4 Accuracy at k and Mean Reciprocal Rank

The two metrics above, P@k and R@k, are defined for the multi-label classification setting of Task 2, where each document is associated with a set of correct codes. Task 1 is instead formulated as a single-label retrieval problem: each query has exactly one correct ICD-10-CM code and the models produce a ranked list of candidates. Two additional metrics are therefore used to evaluate this task.

Accuracy at k (Acc@k) is a binary hit metric. It equals 1 if the correct code appears anywhere in the top-k retrieved candidates, and 0 otherwise. Averaged over all queries:

$$\text{Acc@}k = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\hat{\mathcal{Y}}_i^{(k)}}[c_i] \tag{4.33}$$

where c_i is the single correct code for the query i, $\hat{\mathcal{Y}}_i^{(k)}$ is the set of top-k retrieved codes, and $\mathbf{1}_A[\cdot]$ is the set membership indicator function. We note that in this single-label setting, Acc@k is equivalent to R@k defined in equation 4.32: the denominator is always 1 since each query has a single correct answer. We choose to only report Acc@k rather than R@k to avoid confusion.

Mean Reciprocal Rank (MRR) captures not only whether the correct code is retrieved, but also how highly it is ranked within the candidate list. For each query, the reciprocal rank is the inverse of the position at which the correct code first appears. Over all queries:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i} \tag{4.34}$$

where rank_i is the position of the correct code c_i in the ranked list for query i. If c_i does not appear in the ranking, its contribution to the sum is 0. A model achieving MRR = 1.0 always places the correct code first; MRR = 0.5 implies the correct code is on average ranked second.

MRR is particularly relevant in a clinical coding assistance setting. A human reviewer inspecting the top suggestions benefits more from a model that ranks the correct code first than from one that retrieves it somewhere in the top 10 but buries it below several incorrect candidates. MRR directly captures this distinction, whereas Acc@k treats all positions within the top-k equally.

Chapter 5

Methodology

This chapter describes the experimental pipeline developed to address the tasks defined in the Problem Statement (Chapter 3): linking short clinical expressions to ICD-10-CM codes and predicting ICD-10-CM codes from full discharge notes.
The chapter is organised as follows: Section 5.1 describes the datasets used in this work. The first source is the UMLS MRCONSO.RRF file, used as a knowledge base. A synthetic dataset was also generated for the first task. For the second task, 2 benchmark setups from the literature are presented, based on MIMIC-IV. Finally, a clinical dataset that we constructed from MIMIC-IV is introduced, along with the preprocessing and splitting strategies we applied to it.

Section 5.2 describes the methodology developed for the first task, formulated as an information retrieval problem. Two categories of approaches are compared: lexical string-matching baselines and neural embedding based models built on top of biomedical BERT encoders.

Section 5.3 describes the methodology developed for the second task, formulated as a multi-label classification problem. We follow a progressive experimental design, starting from a TF-IDF baseline and incrementally introducing more expressive architectures. We explore chunk-based neural models with various aggregation strategies, including mean pooling, Bi-GRU, and label-wise attention. We investigate the effect of chunking strategy, encoder choice, and context window size. Finally, we explore two approaches to exploit the natural hierarchy of ICD-10 codes. The first one is a top-down masking procedure applied at inference time. The second option is a multi-task architecture that jointly predicts all three granularity levels during training. Section 5.4 describes the hardware environment used.

5.1 Datasets description

5.1.1 UMLS: MRCONSO.RRF

The Unified Medical Language System (UMLS) is "a set of files and software that brings together many biomedical and health vocabularies and standards to enable interoperability between computer systems"[57, 21]. The MRCONSO.RFF file is widely used, as it represents the entire metathesaurus concept structure. It provides the mapping between a Concept Unique Identifier (CUI) and the specific names, synonyms, and abbreviations provided by over 200 different source vocabularies.

5.1.2 Large patient records dataset: MIMIC-IV

MIMIC-IV (Medical Information Mart for Intensive Care IV) is a large de-identified clinical database built from patient records at the Beth Israel Deaconess Medical Center in Boston. It covers the period from 2008 to 2022 [58]. Access to the dataset requires completing a training course and signing a data use agreement on PhysioNet.
The database is organised into two main modules. The *hosp* module contains hospital-level data, including patient demographics, admissions, diagnoses, and procedures. The *icu* module contains high-frequency intensive care data such as vital signs and monitoring events [58].
Clinical notes are available through a linked resource called MIMIC-IV-Note, which contains 331 793 de-identified discharge summaries from 145 914 patients [12]. These discharge notes are the primary input used in this work. Diagnostic codes associated with these notes are provided in both

ICD-9-CM and ICD-10-CM formats, in the `diagnoses_icd.csv` file. This makes the dataset directly suitable for multi-label ICD coding tasks. A summary of the key statistics of the MIMIC-IV database is provided in Table 5.1.

Table 5.1: General statistics of the MIMIC-IV database (v3.1) [58, 12]

Property	**Value**
Hospital	Beth Israel Deaconess Medical Center, Boston
Coverage period	2008 – 2022
Total hospital admissions	546 028
Unique patients	223 452
MIMIC-IV-Note (discharge summaries)	
Total discharge summaries	331 793
Patients with discharge summaries	145 914
Admissions linked to ICD-9 codes (with notes)	209 323
Admissions linked to ICD-10 codes (with notes)	122 288
Annotation	
Coding systems	ICD-9-CM and ICD-10-CM
Annotation source	Professional medical coders
Access	PhysioNet (training + data use agreement)

Advantages for ICD coding MIMIC-IV presents several properties that make it particularly well suited for this work. First, its scale is a major asset: over 330 000 annotated discharge summaries provide sufficient data to train large neural models. Second, the codes are assigned by professional medical coders, which ensures a high level of annotation quality. Third, MIMIC-IV has become the de facto benchmark in the ICD coding literature, making it possible to compare results directly against a large number of published models.

Limitations for ICD coding Despite its strengths, MIMIC-IV has several limitations that are important to acknowledge. First, all notes come from a single hospital in the United States. The writing style, clinical workflows, and coding practices of the Beth Israel Deaconess Medical Center may not generalise to other institutions, countries, or healthcare systems. A model trained on MIMIC-IV may therefore struggle when deployed in a different clinical environment.
Second, discharge summaries in MIMIC-IV follow a relatively structured and consistent format, with clearly delimited sections such as chief complaint, history of present illness, and discharge diagnosis among others. This is not representative of all real-world clinical notes, which are often more informal, fragmented, or inconsistently structured.
Third, the dataset uses ICD-9-CM codes for admissions before 2015 and ICD-10-CM codes for more recent admissions. This version shift introduces heterogeneity in the coding system, which is why this work is restricted to ICD-10-CM codes only.

Benchmark setups in the literature based on MIMIC-IV

While MIMIC-IV has emerged as the dominant dataset in the field, the absence of a standardised benchmark setup means that comparisons remain difficult. Many studies use private datasets, and even those using MIMIC-IV differ in how they filter codes, split documents, and define the label space. Some studies evaluate only on the top 50 most frequent codes, others on the full code set. These choices have a direct impact on reported metrics, making cross-study comparisons unreliable without careful contextualisation.
To situate our experimental choices, we identify two reference setups from the literature that will serve as comparison points in the results (Chapter 6).

Setup A The first reference setup [59] uses a random admission-level split. This means the same patient can appear in both the training and test sets. The model may then memorise patient-specific patterns rather than generalise. This makes Setup A a less rigorous evaluation setting.

Setup B The second reference setup [5, 60] adopt a patient-level split, preventing leakage. However, their frequency threshold requires a code to appear in at least 10 admissions, which retains only 7,942 codes. Furthermore, their splitting strategy does not guarantee that every code appears in every split: around 0.5% of codes are absent from the validation set and 0.1% from the test set, making it impossible to evaluate the model on those codes.
The limitations identified in these two setups motivated us to design our own setup, described in the following section. To improve over existing configurations, we adopt a patient-level split to prevent data leakage, lower the frequency threshold to at least 3 patients to retain more rare codes, and enforce near-complete code coverage across all splits.

Data preparation for our setup

Data construction The dataset is constructed by joining two tables from MIMIC-IV. The first consists of the discharge notes from MIMIC-IV-Note, which contain the full clinical text of each hospital stay. The second is the diagnostic codes table, which contains the ICD codes assigned to each admission. Only ICD-10-CM codes are retained, as ICD-9 codes follow a different structure and would require a separate label space. The two tables are merged on the hospital admission identifier (`hadm_id`), producing one record per admission with its associated discharge note and list of ICD-10 codes.

Code filtering ICD-10 codes follow a highly uneven distribution: many codes appear only once or twice in the entire dataset. A model cannot learn a reliable representation for a code that appears in a single patient record. And such codes would be impossible to evaluate properly. To address this, only codes that appear in at least three distinct patients are retained. This filtering step ensures that every code in the final label space is carried by enough distinct patients to be representable across all three splits : training, validation, and test.

Train, validation and test split The dataset is split at the patient level rather than the admission level [16]. This choice is essential to prevent data leakage. If two admissions from the same patient were placed in different splits, the model could implicitly learn patient-specific patterns rather than generalisable coding rules.
The split follows a strategy designed to maximise the chances that every valid code appears in all three sets. Codes are processed from the rarest to the most frequent.
For each code, if it is not yet represented in one of the three sets, a patient associated with that code is assigned to the missing set. The remaining patients, whose codes are already well represented, are distributed randomly with a 70/15/15 train/validation/test ratio. In practice, this results in near-complete coverage: of the 9,685 valid codes, 9,683 appear in the training set, 9,678 in validation, and 9,670 in test. At most 15 codes (0.16%) are absent from any single split.

Text preprocessing Transformer-based models such as PubMedBERT handle raw text well and do not require heavy linguistic preprocessing. The only transformation applied is the normalisation of whitespace: multiple consecutive spaces, tabs, and newlines are collapsed into a single space. No tokenisation, stopword removal, or vocabulary filtering is performed, as these operations could remove clinically relevant information.

Label encoding Each discharge note is associated with a list of ICD-10 codes. These lists are converted into binary vectors using a MultiLabelBinarizer fitted on the training set only. Each position in the vector corresponds to one ICD-10 code in the label space. A value of 1 indicates that the code was assigned to that admission, and 0 otherwise.
The binariser is saved and reused at inference time to decode model predictions back into ICD-10 codes.

Dataset statistics The final dataset contains 122,275 discharge summaries, split into 75,133 training notes, 23,608 validation notes, and 23,534 test notes. Following our filtering step, the final training set includes 9,685 distinct ICD-10 codes. This represents a reduction from the 16,155 codes present in admissions with discharge notes. This filtering was intentional: we only kept codes with a minimum frequency of three patients to ensure that each label has sufficient representation across the train, validation, and test sets. A detailed summary of these statistics, along with the key methodological choices that shaped this dataset, is provided in Table 5.2.
The code distribution is strongly imbalanced, as illustrated in Figure 5.1. The most frequent code

(I10, Essential Hypertension) appears 27,307 times in the training set, while many codes appear only a few times. This long-tail distribution is a central challenge of the ICD coding task.

Table 5.2: Dataset statistics after our preprocessing pipeline (ICD-10-CM diagnosis codes only).

Property	**Value**
Filtering choices	
Code type retained	ICD-10-CM diagnosis only
Minimum frequency threshold	≥ 3 distinct patients
Split strategy	Patient-level
Label space	
Unique ICD-10 codes in MIMIC-IV (diagnosis only)	19,440
Unique ICD-10 codes linked to discharge notes	16,155
Unique ICD-10 codes after frequency filtering	9,685
Dataset size	
Total discharge summaries	122,275
Training set	75,133
Validation set	23,608
Test set	23,534
Note and label statistics	
Average words per note	1,736
Average ICD-10 codes per note	14.09
Most frequent code (train)	I10 – Essential hypertension (27,307)

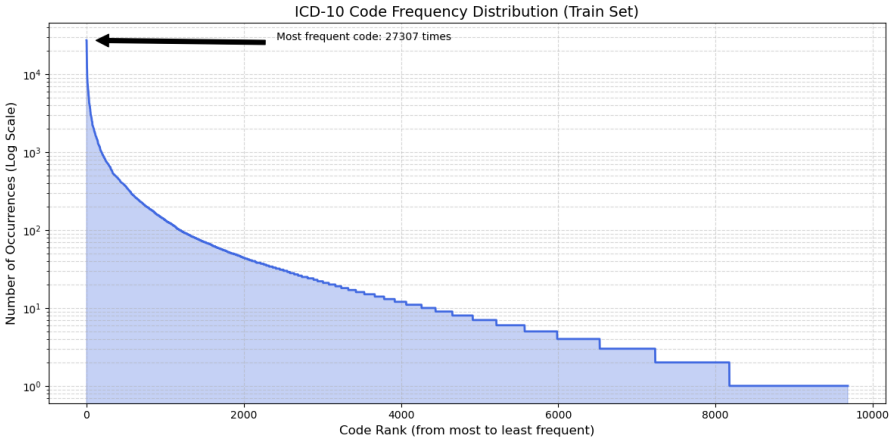


Figure 5.1: Frequency distribution of ICD-10 codes in the training set (log scale).

5.1.3 Short clinical description generation

No suitable benchmark dataset was found for evaluating the task of matching short clinical descriptions to ICD-10-CM codes. Therefore, wr decided to generate a synthetic dataset using a large language model (LLM). The target dataset should satisfy three properties. First, it must cover all ICD-10-CM chapters in a balanced manner, to allow for a chapter-level evaluation. Second, it must contain a variety of description styles, including abbreviations, informal phrasings, and slight misspellings, to be representative of the lexical variability encountered in real clinical notes. Third, examples should span a range of retrieval difficulties, from straightforward paraphrases to semantically more distant but valid reformulations, to provide a meaningful discriminative benchmark for comparing models.
We retrieved the ICD-10-CM codes and their associated descriptions from the `d_icd_diagnoses.csv` file, contained in the MIMIC-IV dataset presented just above. It represents 97,441 ICD pairs.

The generation process relies on two small open-source LLMs rather than a single model, so as to minimise model-specific bias and increase the lexical diversity of the generated ICD description variations. First, we tried Llama 3.1 8B, a general-purpose open-weights language model developed by Meta and released in July 2024[61]. Second, we chose Biomistral 7B, a domain-specific model described by its authors as "a collection of open-source pretrained large language models for medical domains"[62]. Built upon the Mistral architecture, it was further pre-trained on biomedical literature from PubMed Central Open Access and also released in 2024. The use of a domain-specific model alongside a general-purpose one also allows a post-hoc analysis of whether medical pretraining improves the quality of generated clinical variations.

We performed the model inference with Ollama. This framework allowed us to easily download and run open-source models locally, through a command-line interface and API. The same prompt was applied for both models, with precise instructions to generate five stylistically distinct variations per ICD code-description pair:

```
f"""
You are a medical expert.
ICD-10 Code: {code}
Official Description: {description}

Generate 5 distinct variations of this diagnosis as they might appear in a doctor's clinical note.
Include:
1. A formal medical term (synonym)
2. A common abbreviation
3. A short, informal description of the condition
4. A version with a slight typo
5. A concise phrase (fast/sloppy note format)
Prefer challenging variations that are lexically distinct from the official description.

CRITICAL FORMATTING INSTRUCTION:
- Output strictly as a JSON LIST OF STRINGS.
- Do NOT include the ICD code in the output.

Example Output:
["Heart attack", "Acute MI", "Myocardial infarction", "AMI", "Severe chest pain (MI)"]
"""
```

We conducted two iterations of the generation process. The first iteration operated on a limited batch with the objectives of validating the pipeline and obtaining preliminary results. We then performed a second iteration to expand the dataset and make the data more significant in terms of size. For the later evaluation of the models, we will use the combination of examples from both iteration, forming the complete combined dataset.

The dataset generation pipeline followed these steps, as summarised in Figure 5.2 flowchart:

1. Dictionary filtering The ICD dictionary from MIMIC-IV contains both ICD-10-CM and ICD-9-CM codes. Since our work only concerns ICD-10-CM, the first step was to filter out of the dictionary any code referring to ICD-9. Additionally for the second iteration, we identified and excluded form the available pool the codes that were already sampled during the first iteration. That way, we ensured both produced non-overlapping samples.

2. Stratified sampling: Each code was assigned to its corresponding ICD-10-CM chapter, the highest level of the hierarchy (grouping codes by broad disease category) . Within each chapter, we sampled $\min(5, |C_i|)$ code–description pairs, where $|C_i|$ denotes the number of codes in chapter i, using a fixed random seed (`random_state=42`) to ensure reproducibility. For the second iteration, we increased the number of sampled pairs to $\min(10, |C_i|)$.

3. Generation of description variants: For each sampled code-description pair, the defined prompt was submitted to each model with `format='json'` to enforce a structured output. The returned list, ideally containing five variations of the description, was parsed recursively to account for occasional formatting inconsistencies. Duplicate entries were removed.

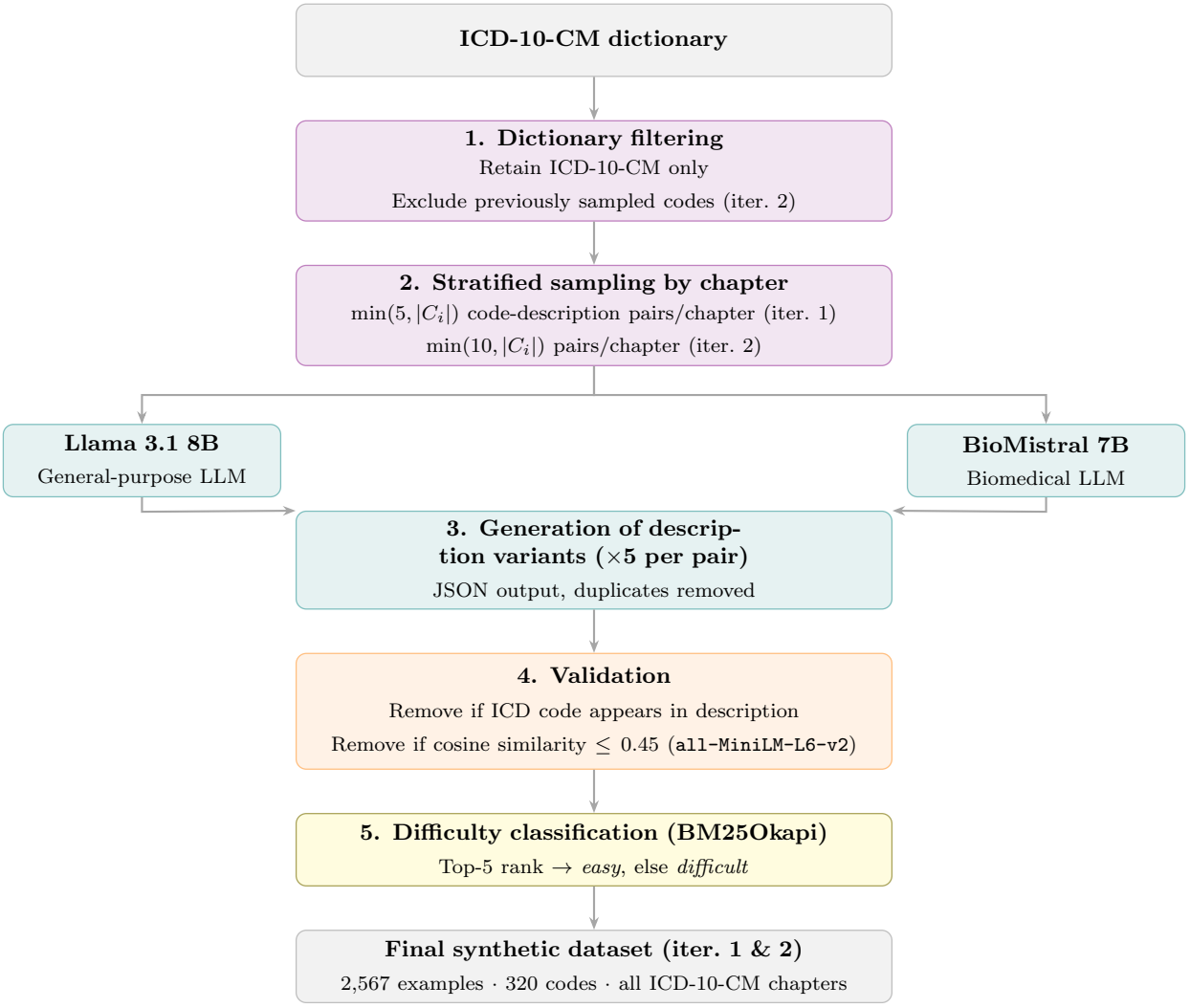


Figure 5.2: Overview of the synthetic dataset generation pipeline

4. Synthetic dataset validation: Two filtering criteria were applied on generated variants to ensure dataset quality. First, we discarded every example in which the ICD code appeared directly within the generated description, as they do not represent typical examples for this task. Second, a semantic similarity analysis was conducted. Both the official and generated descriptions were encoded using the `all-MiniLM-L6-v2` sentence embedding model from the Sentence Transformers library [63], based on the MiniLM architecture [64]. This lightweight model is transformer-based, and has been extensively validated for sentence-pair tasks. We only retained pairs with a cosine similarity strictly greater than 0.45. This threshold was chosen arbitrarily, with the objective in mind to be permissive enough to accept abbreviations while still rejecting semantically different or erroneous generations.

5. Difficulty classification of codes: The examples were categorised by difficulty to distinguish trivial variations (e.g. simple word reordering, or punctuation changes) from difficult ones (e.g. use of abbreviations and synonymous terms). For this step, we employed the BM25Okapi model from the `rank_bm25` library [65]. It is a probabilistic ranking algorithm based on TF-IDF scoring (which was explained in Section 4.1), and typically used for information retrieval. We obtained a similarity ranking of the official descriptions for each artificial description. If the correct ICD description appeared in the top-5 retrieved result, the example was labelled as easy; else it was labelled as difficult.

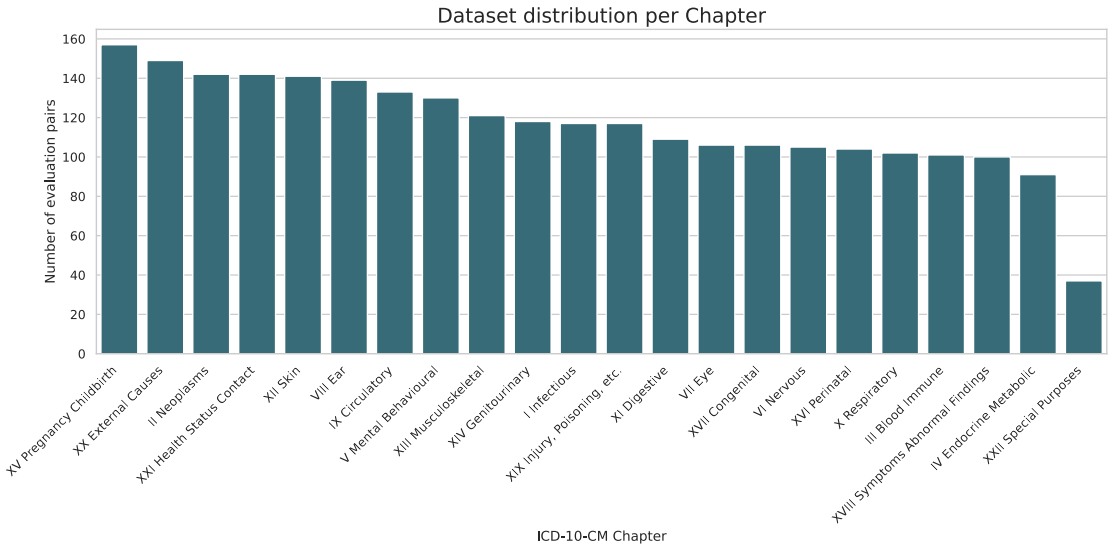
	Iteration 1	Iteration 2	Combined
Codes sampled per chapter	$\min(5, \lvert C_i \rvert)$	$\min(10, \lvert C_i \rvert)$	—
Total codes sampled	110	211	**320**
Examples generated (raw)	1,252	2,384	—
After ICD-in-description filter	1,225 (-27)	2,336 (-48)	—
After semantic similarity filter	910 (-315)	1,657 (-679)	—
Final validated examples	**910**	**1,657**	**2,567**
Retention rate (vs. raw generation)	72.7%	69.5%	**72.7%**
Mean cosine similarity (final set)	0.742	0.749	**0.747**
Difficult examples (rank > 5)	63.6%	62.2%	**62.7%**

Table 5.3: Summary of the dataset generation pipeline across both iterations and the combined total dataset.

Table 5.3 summarises for each iteration, the exact statistics and numbers associated with the generation steps. The two iterations are broadly consistent: retention rates are similar, as are mean cosine similarity scores and proportion of difficult examples. Around 7% of the rejections were attributed to the model including the ICD code verbatim in the generated description, even though the exclusion instruction was clearly stated in the prompt. This is a recurring failure mode of instruction-following in smaller language models. The final combined dataset contains 2,567 examples, covering 320 unique codes across all chapters. It is made available for future uses in Appendix E.

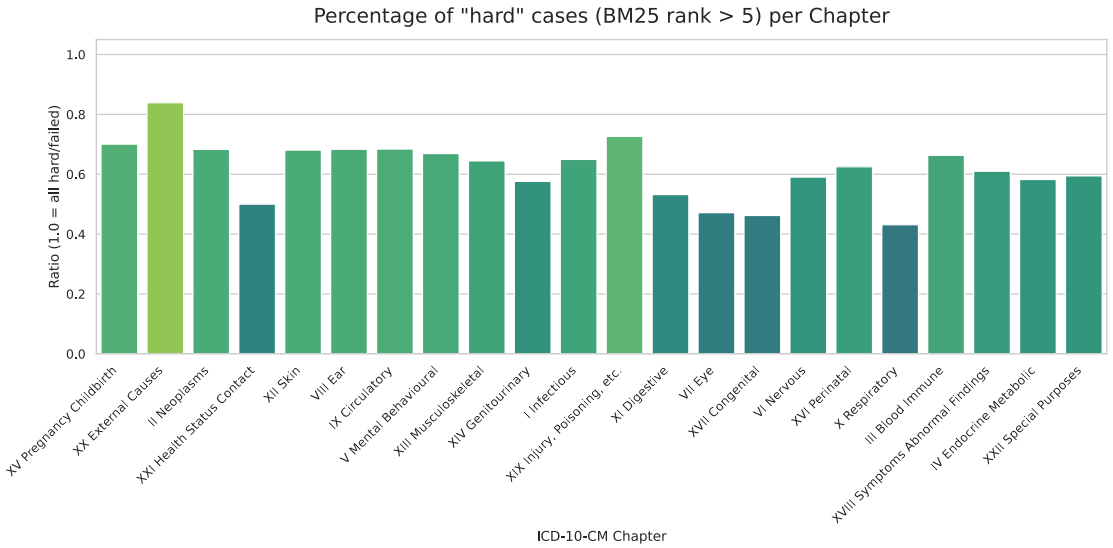
Figures 5.3a, 5.3c and 5.3b present the distribution of the combined dataset across ICD-10-CM chapters. It also displays two quality indicators: the semantic similarity between generated and official descriptions, computed using `all-MiniLM-L6-v2`, and the proportion of difficult examples.

As shown in Figure 5.3a, the dataset is relatively balanced across most chapters, with the majority contributing between 100 and 150 evaluation pairs for around 15 ICD codes. The notable exception is Chapter XXII (Special Purposes), which contains only 37 examples. This is a direct consequence of the limited number of codes in this chapter (five codes only in total), which constrained the sampling in both iterations. This class imbalance should be kept in mind when interpreting per-chapter evaluation results.

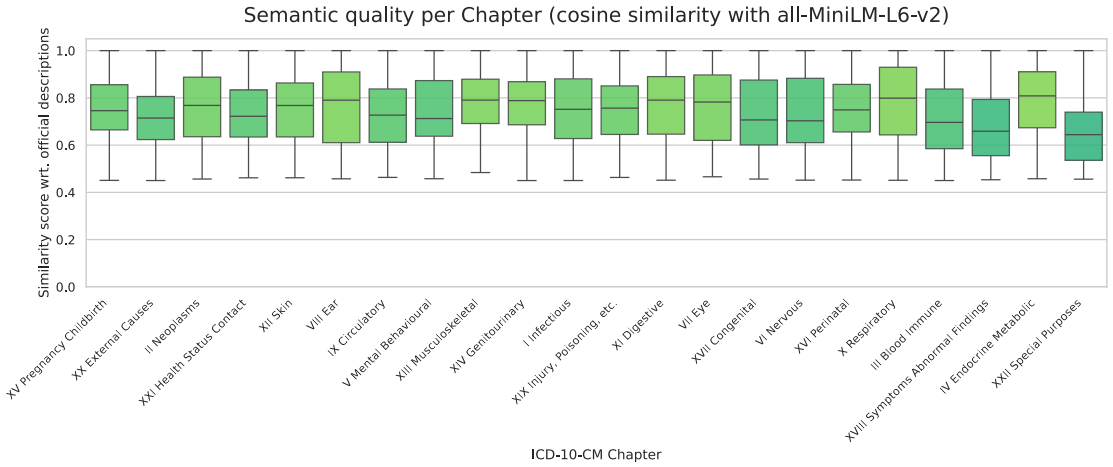


(a) Examples count per Chapter. The dataset is relatively balanced across chapters, even if some domains have more generated examples than others. Chapter XXII has very few examples, because the chapter itself contains only 5 codes.

Figure 5.3: Per Chapter statistics of the generated complete dataset



(b) Semantic difficulty per Chapter. The proportion of difficult examples are broadly consistent. For the majority of chapters, more than half examples are complicated ones.



(c) Semantic score distribution per Chapter. No chapter has an abnormally low median semantic score, almost all falling between 0.70–0.75. The exclusion of examples with semantic score lower than 0.4 was correctly performed.

Figure 5.3: Per Chapter statistics of the generated complete dataset

Figure 5.3c displays the distribution of cosine similarity scores between generated and official descriptions. Median similarity scores are consistently high across all chapters, generally falling in the range [0.68, 0.78], which confirms that the generated variations are semantically grounded in their corresponding official descriptions. Chapter XXII again exhibits the lowest median, reflecting the smaller sample available for that chapter.

Finally, Figure 5.3b reports the proportion of examples classified as difficult within each chapter. Difficulty is broadly consistent across all chapters, mostly ranging from 0.5 to 0.75, which suggests that the generation pipeline often succeeded in producing challenging variations across medical domains. Chapter XX (External Causes) and Chapter XIX (Injury, Poisoning, etc.) exhibit the highest difficulty ratios, whereas Chapters XVII (Congenital) and X (Respiratory) appear somewhat easier under the BM25 metric. No strong correlation is observed between chapter size and difficulty level.

Visually, there appears to be no strong correlation either between the percentage of hard examples and the median semantic similarity between official and generated descriptions in a chapter. To further examine this relationship, we conducted a correlation analysis at two levels of granularity. At the chapter level, we used the Spearman rank correlation, which operates on ranked values. It indicates whether an increase in one variable consistently correlates with an increase

or decrease in the other, regardless of whether that relationship is linear. The test yielded a coefficient $\rho = -0.154$ ($p = 0.493$), indicating no statistically significant association. This suggests that chapters with higher difficulty ratios do not systematically produce lower-quality generations, and that the BM25-based difficulty classification is not simply a proxy for poor semantic alignment.

However, the statistical test at individual-example level tells a different story. We performed a Point-Biserial correlation between the binary difficulty label and the cosine similarity score, which is the typical choice when one variable is dichotomous. It yielded $r_{pb} = -0.559$ ($p < 0.001$), revealing a moderate and highly significant negative relationship. Difficult examples tend to exhibit lower semantic similarity to their official descriptions than easy ones, which can be expected. Those examples that are harder to retrieve are precisely those whose generated descriptions are more lexically and semantically distant from the official formulation, making them more representative of the challenging cases encountered in real clinical settings.
The observed divergence between the two granularity levels is actually not so surprising, and reflects an aggregation effect: the within-chapter variance is smoothed out by considering only the chapter median.

A comparison of the two generative models reveals that BioMistral produces marginally higher semantic similarity scores than Llama 3 among easy examples (median around 0.91 vs. 0.87), while the two models perform near-identically on difficult examples (median around 0.69 for both). Given this negligible difference in the harder and more informative subset of the data, both models were retained in the final dataset to maximise coverage and diversity of generated variations.

5.2 Group of words to ICD codes methodology

This first task is formulated as an information retrieval problem. Given a short clinical expression q such as a diagnostic sentence, an abbreviation, or a clinical note fragment, the goal is to retrieve the correct ICD-10-CM code c^* from a fixed candidate set $\mathcal{C}$ containing all codes present in the ICD-10-CM dictionary extracted from MIMIC-IV. This dictionary comprises 97,441 codes, and was also used for our synthetic dataset generation (Section 5.1.3). Each model produces a ranked list of candidates from $\mathcal{C}$, and evaluation measures whether c^* appears within the top k positions of that ranking.

We compared seven models spanning two methodological families: lexical string-matching baselines — Levenshtein distance, Dice-Sørensen coefficient, T-LCS, and W-LCS — and neural embedding-based approaches — PubMedBERT, SapBERT, and ClinicalModernBERT. This selection is motivated by the need to cover a spectrum ranging from simple, interpretable methods to state-of-the-art biomedical encoders.

5.2.1 Lexical baselines

Four character-level string similarity measures are included as lower-bound baselines. All operate directly on the text, requiring no training and no external resources outside of the ICD-10-CM dictionary. They provide a reference point for assessing the added value of neural networks approaches. The only preprocessing applied to all models is lowercasing. This is a more defensive choice, as we wanted to preserve clinically meaningful abbreviations and acronyms that stopword removal or stemmimg would have distorted.

Levenshtein distance As introduced in the State of the art (Section 2.1.2), the Levenshtein distance measures the minimum number of single-character edits (insertions, deletions, and substitutions) required to transform one string into another. It is the most straightforward lexical baseline. The distance is normalised by the length of the longest string, in order to obtain a similarity score between 0 and 1. At inference time, the query is compared against all official descriptions in $\mathcal{C}$ and candidates are ranked by decreasing similarity (increasing Levenshtein distance).

Dice-Sørensen coefficient The Dice-Sørensen coefficient measures the overlap between two sets of character bigrams. For two strings s_1 and s_2, the sets of character bigrams are defined as:

$$\mathcal{B}(s) = \{s[i : i + 2] \mid i \in \{0, \ldots, |s| - 2\}\} \tag{5.1}$$

and the coefficient is computed as

$$\text{Dice}(s_1, s_2) = \frac{2 \cdot |\mathcal{B}(s_1) \cap \mathcal{B}(s_2)|}{|\mathcal{B}(s_1)| + |\mathcal{B}(s_2)|} \tag{5.2}$$

Compared to Levenshtein distance, the Dice-Sørensen coefficient is more robust to local morphological variation and word-order differences, as it operates on an unordered set of overlapping character patterns rather than a global sequence. The obtained score is greater when the two strings are similar, contrarily to the Levenshtein score where it would be smaller. Candidate ranking follow the same procedure as for Levenshtein distance.

Longest Common Subsequence Two additional lexical baselines are included based on the Longest Common Subsequence (LCS), as introduced in Section 2.1.2. The Twofold LCS (T-LCS) improves the basic LCS with a more balanced weighting of longest common subsequence length (LCSL): dividing it by the average sequence length instead of the longest string length. The similarity coefficient is defined as:

$$\text{T-LCS}(s_1, s_2) = \frac{2 \cdot \text{LCSL}(s_1, s_2)}{|s_1| + |s_2|} \tag{5.3}$$

The Weighted LCS (W-LCS) was designed to increase the proportion of LCSL in the computation and to improve the score in the case where one string is mostly contained by the other. It is computed with:

$$\text{W-LCS}(s_1, s_2) = \frac{(\text{LCSL}(s_1, s_2) + 1) \cdot \text{LCSL}(s_1, s_2)}{|s_1| \cdot \text{LCSL}(s_1, s_2) + |s_2|} \quad \text{where} \quad |s_1| \leq |s_2| \tag{5.4}$$

Practically, we find the longest common subsequence length by constructing a matrix where every character of the first sequence is compared against every character of the second sequence. An entry m_{ij} stores the length of the longest subsequence found up to the $i-th$ character of s_1 and the $j-th$ character of s_2. LCSL is entry $m_{|s_1||s_2|}$.
Both T-LCS and W-LCS computations produce a similarity score between 0 and 1, which reflects how close the compared strings are. Candidate ranking still follow the same procedure.

5.2.2 Neural embedding-based models

We evaluated four neural models based on BERT pre-trained encoder. All four follow the same inference procedure: the full ICD-10-CM dictionary is encoded once at build time to produce a static index of description embeddings, stored using FAISS [66]. Specifically, a `IndexFlatIP` index is used after L2-normalisation of all embeddings, which is equivalent to performing exact cosine similarity search. ach query is encoded and compared against the index to retrieve the top-k most similar candidates. For all models, the `[CLS]` token representation of the final hidden layer is used as the sentence embedding.

SapBERT SapBERT represents the very established state of the art, that was most used in the literature for biomedical concept representation/linking. This model was presented previously in the State of the art Chapter, in the Section 2.1.5. The pretrained checkpoint `cambridgeltl/SapBERT-from-PubMedBERT-fulltext` was loaded from HuggingFace. Descriptions are tokenised with padding to a maximum length of 25 tokens, following the encoding strategy recommended for this model. For the index dictionary building, we encode all official ICD-10-CM descriptions in batches of 128.

SapBERT reimplementation We reimplemented SapBERT from scratch, starting from the PubMedBERT base model and reproducing the self-alignment pre-training procedure described in the original paper [27]. This reimplementation serves three purposes: verifying the reproducibility of the original approach in the context of ICD linking, gaining a first practical knowledge of the models, and providing a controlled base for potential domain-specific future adaptations.

First, we extracted the knowledge about positive associations, using the MRCONSO.RFF file from UMLS. All terms were grouped by their associated CUI, and for each concept we generated all

pairwise synonym combinations: the positive pairs. To prevent common concepts with many synonyms from overwhelming the training, we limited the number of positive pairs per concept to 50.

The base model is `microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext`, loaded from HuggingFace. Training was performed using a Multi-similarity miner (`epsilon=0.1`) to identify "hard" pairs in each batch (that is, the pairs that are currently not well placed in the embedding space), and a Multi-similarity loss (`alpha=2, beta=50, base=0.5`) to simultaneously pull synonyms closer together and push unrelated terms apart. The model was optimised with `AdamW` (`lr=2e-5`, `weight_decay=0.01`), with a batch size of 64 and input sequences truncated to a maximum of 40 tokens. Each training step feeded two strings into the PubMedBERT encoder, which generated a vector for each, then the loss function compared the two vectors with regards to their CUI index. The model finally updated its weight to fix the alignment.
At inference time, it follows the identical procedure to the original SapBERT checkpoint described above.

PubMedBERT Plain PubMedBERT is included as the primary neural baseline. It uses the same pretrained checkpoint as the SapBERT reimplementation (`microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext`), but without any further alignment pre-training. Its role is to highlight the contribution of the backbone architecture from that of the self-alignment objective: any performance gap between PubMedBERT and SapBERT can be attributed specifically to the synonym-aware training procedure rather than to the underlying language model. The inference procedure is identical to the other neural models, using the `[CLS]` representation and a FAISS index, constructed using the exact same parameters as SapBERT models.

Clinical ModernBERT Clinical ModernBERT, which was previously introduced in Section 4.6.2, is included to assess whether the architectural improvements introduced in the ModernBERT family translate into gains on this specific retrieval task. The model was pre-trained on a large corpus of clinical text and it benefits from a longer context window and updated attention mechanisms relative to earlier BERT variants. The pretrained checkpoint `Simonlee711/Clinical_ModernBERT` was loaded from HuggingFace. Descriptions are tokenised with dynamic padding and a maximum length of 64 tokens. The dictionary index is built using the same batched GPU encoding procedure as the other neural models.

5.2.3 Evaluation protocol

All models are evaluated on the combined generated dataset of 2,567 examples presented in Section 5.1.3. Since neither lexical models or pretrained neural models require training, no train/test split was applied; the entire dataset serves as evaluation set. The only training exception is the reimplemented SapBERT, but it used UMLS synonym pairs as training set and is therefore independant from the generated evaluation set. Each model is evaluated globally, and results are also broken down by example difficulty (easy vs. hard, as defined in section 5.1.3). We performed retrieval for $k \in \{1, 5, 10\}$.

5.3 Full discharge note to ICD codes methodology

This section describes the sequence of models developed for the full discharge note to ICD-10 codes classification task. Rather than simply searching for the best architecture, our goal is to understand what each design choice actually contributes, at every level of the ICD-10 hierarchy. For this reason, every architecture is systematically evaluated at three levels of granularity: chapter level prediction, 3 character code prediction, and full ICD-10 code prediction. All models are trained and evaluated on the MIMIC-IV dataset described in Section 5.1.2

We follow a progressive approach, where each step builds on the previous one and attempts to push the model further, see Figure 5.4. We start with a simple TF-IDF baseline to establish a reference point. If a more complex model fails to outperform it, this signals an issue in data preparation, metrics, or tuning.
We then move to chunk-based neural architectures using PubMedBERT, where each discharge note is split into overlapping chunks that are independently encoded and then aggregated. We begin with a simple aggregation strategy, mean pooling to verify that the use of PubMedBert

yields a clear gain over the baseline. For all chunk-based models, a separate model is trained per granularity level: one for chapters, one for 3-character codes, and one for full codes, each with its own classification head and label binariser.
Building on this, we investigate more expressive aggregation mechanisms, replacing simple pooling with attention based aggregation over chunk representations. We also explore a Bi-GRU as an alternative aggregator, which processes the sequence of chunk representations in both directions to capture ordering dependencies across chunks. We also compare two chunking strategies to measure their respective impact. To further push the model, we test a full document attention approach, which attends over all token representations simultaneously without chunking, allowing the model to capture long range dependencies across the entire note. We additionally replace PubMedBERT with Clinical ModernBERT, a more recent clinically pretrained encoder, to assess whether a closer domain pretraining brings meaningful gains, see Figure 5.5.
Finally, we move to a multitask architecture where all three granularity levels are trained jointly. The discharge note passes through the encoder only once, and three separate classification heads predict chapters, 3-character codes, and full codes simultaneously, with the hypothesis that higher level predictions can guide and regularise lower level ones. This design also reduces computational cost.

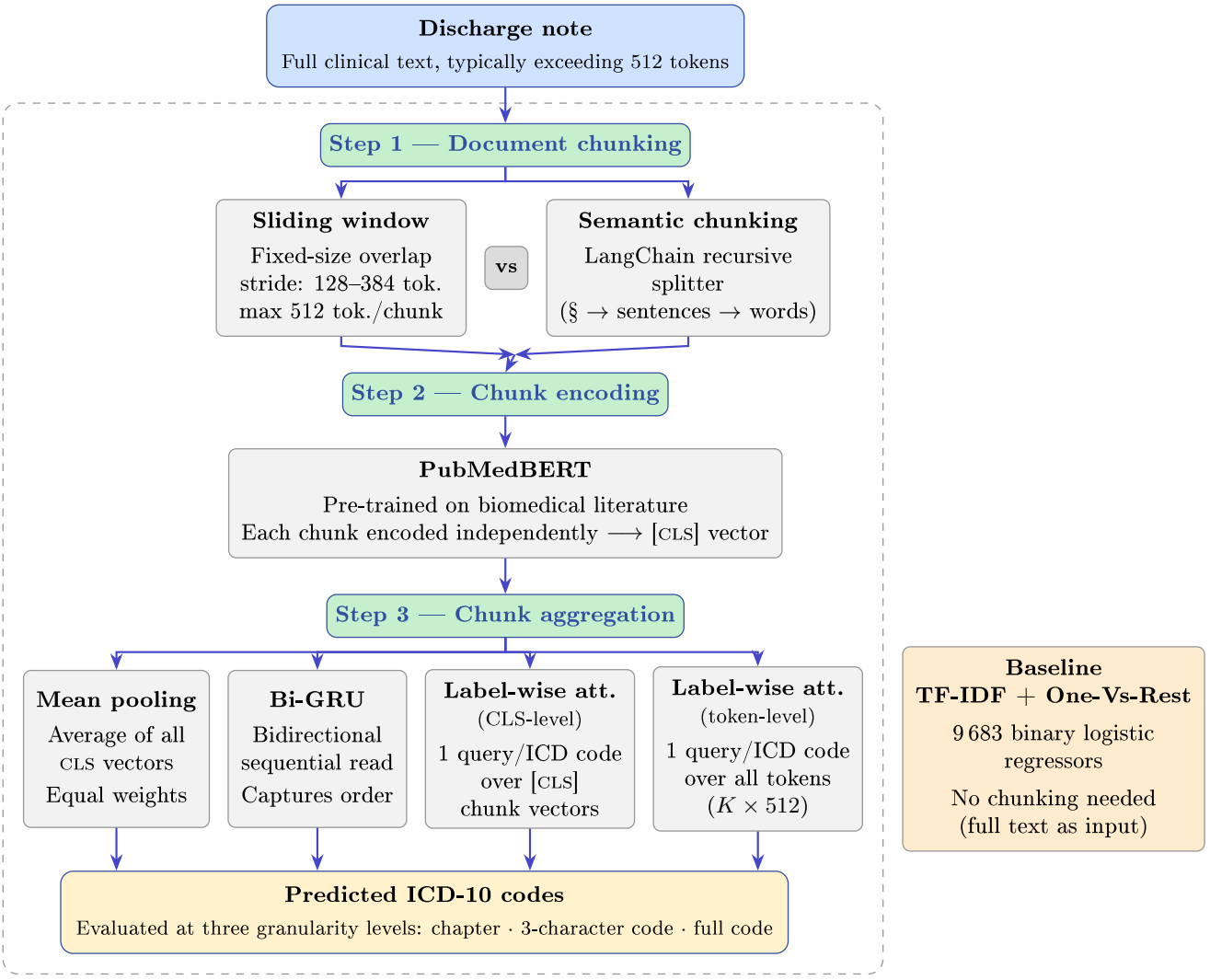


Figure 5.4: Overview of the ICD-10 coding pipeline. Step 1 chunks the discharge note using either a sliding window or semantic chunking. Step 2 encodes each chunk with PubMedBERT. Step 3 aggregates chunk representations before predicting ICD-10 codes at three granularity levels.

Hyperparameter optimisation

All models were tuned using Optuna, an open source hyperparameter optimisation framework that follows a define by run paradigm [67]. The search space is defined dynamically within the objective function using standard Python control flow. This makes it easy to handle conditional and heterogeneous parameter spaces. This is particularly useful, since the optimal configuration varies across architectures and granularity levels.
Sampling is handled by the Tree structured Parzen Estimator (TPE), which builds a probabilistic model of the objective function to prioritise promising regions of the search space. To reduce computational cost, Optuna's asynchronous Successive Halving (ASHA) pruner terminates unpromising trials early based on intermediate validation performance. It is a critical feature given the high training cost of transformer-based models. The specific parameters explored and the number of trials are detailed for each architecture below.
However, due to computational constraints, the number of trials remained limited for several models, making it difficult to reliably explore the full combined space of learning rate, chunk size, and other hyperparameters. In these cases, Optuna failed to converge to a stable configuration, and hyperparameters were instead set to values reported in the literature for similar architectures. For each model, the hyperparameters explored during tuning are described in the corresponding section. The exact parameters used in the final models can be found in Appendix B

5.3.1 Baseline : TF-IDF

We implemented a traditional machine learning pipeline to serve as a performance baseline, see Figure 5.4. This approach transforms medical text into numerical features using TF-IDF. We then process these features through a multi-label classification strategy, see Section 4.1.

The TF-IDF method evaluates the importance of each word. Term Frequency (TF) measures how often a word appears in a specific report. Inverse Document Frequency (IDF) reduces the weight of common words across the entire dataset. This highlights rare and specific clinical terms. To reduce noise, we filter out terms appearing in fewer than three documents or in more than 95 % of reports. We also apply sublinear TF scaling, replacing raw term frequency tf with $1 + \log(tf)$, which diminishes the disproportionate influence of highly repetitive words in long clinical notes.

Hyperparameter optimisation To ensure a fair comparison with deep learning models, we perform hyperparameter tuning using Optuna over 15 trials per granularity level, maximizing Micro-F1. Three key parameters are explored. First, the n-gram range, tested across (1,1), (1,2), and (1,3), controls whether the vocabulary consists of single words only or also includes two and three word phrases. This is critical for capturing complex medical expressions such as "chronic kidney disease" rather than isolated tokens. Second, the maximum vocabulary size is tuned between 20,000 and 80,000 features. This balance the model's ability to recognise rare medical terms against the risk of producing overly sparse and memory intensive feature matrices. Third, the inverse regularisation strength C is searched on a logarithmic scale between 0.1 and 10.0. This value decides the trade off between fitting the training data and generalising to unseen notes.

Multi-label classification The classification task is challenging because ICD-10 coding is multi-label. Each patient report can be associated with several different codes simultaneously. To handle this, we utilise a One-Vs-Rest (OvR) architecture. This strategy decomposes the problem into thousands of independent binary tasks. For our dataset, the system trains 9,685 separate Logistic Regression models. Each individual model specialises in one specific ICD-10 code. It learns to predict a simple "Yes" or "No" for its assigned category. By aggregating the results of all models, the system can predict multiple codes for a single patient. This classic approach provides a transparent and robust benchmark for evaluating our more advanced neural network architectures.

5.3.2 PubMedBert + mean pooling

To evaluate the benefits of a neural architecture over the TF-IDF baseline, we fine tuned PubMedBERT. This model is specifically pre-trained on biomedical literature to better handle clinical terminology 4.6.1.

Handling long documents via chunking Standard BERT architectures are limited to a maximum of 512 tokens per input. However, medical reports in our dataset often exceed this length. To ensure the model has access to the same global document information as the TF-IDF baseline, we adopt a sliding window chunking strategy. Each note is split into overlapping chunks of 512 tokens. For each chunk, the encoder produces a [CLS] vector serving as a compact semantic summary of that segment.

Mean pooling aggregation To obtain a single representation for the entire medical report, we apply mean pooling across all chunk [CLS] vectors 4.7.1. This averaged vector is then passed to a linear classification head to predict ICD-10 codes. This aggregation is intentionally simple and treats all chunks with equal importance. It serves as a transparent intermediate step between the TF-IDF baseline and more expressive aggregation mechanisms explored later. We refer to this model as `modelB_mean` throughout the rest of this thesis.

Hyperparameter optimisation Training is conducted using AdamW, with the learning rate searched on a logarithmic scale between 1×10^{-5} and 5×10^{-5}. The stride between consecutive chunks is tuned over {128, 256, 384} tokens, and the maximum number of chunks per document over {4, 8, 12, 16}, allowing the model to process up to approximately 8,000 tokens. Batch size is tuned over {4, 6, 8}, constrained by GPU memory. We run 30 Optuna trials for the granularity level chapter and 5 trials for 3-character and full because 30 trials took too much time. We maximised Micro-F1 for chapter and 3-character code tasks, and Recall@10 for full code prediction.

5.3.3 Chunk aggregation: from mean pooling to Bi-GRU and label Attention

A key design decision in chunk based models is how to combine the representations of individual chunks into a single document level vector. The simplest approach is mean pooling, as used above. However, this approach has a fundamental limitation for discharge note coding. A discharge note is not a homogeneous document. It contains diverse clinical information spread across sections with very different roles. Averaging all chunk representations collapses this structure into a single undifferentiated vector, diluting the information rather than preserving it. This is particularly problematic for ICD coding, where each note is associated with an average of 14 codes, each potentially grounded in a different section of the document. Prior work has shown that mean pooling is not sufficient to capture the complexity of long clinical documents [68].

Bi-GRU aggregation

To address this, we explored the use of a Bidirectional GRU (Bi-GRU) as an aggregation mechanism 4.2. Instead of averaging the chunk representations, the Bi-GRU processes them as a sequence, reading the chunks from left to right and from right to left simultaneously. This allows the model to capture the narrative structure of discharge notes, where the position of information matters. A reason for admission mentioned in an early chunk carries different clinical weight than a complication reported at the end. Dependencies between distant segments can thus be explicitly modeled, rather than lost through averaging.
This design choice is supported by empirical evidence: comparing mean pooling and Bi-GRU aggregation on top of a BERT-based medical model shows that preserving the sequential order of chunks and learning to weight transitions differently brings a meaningful gain over treating all segments as equally informative [69].

The architecture follows the same two stage design as the mean pooling model. In the first stage, the discharge note is split into overlapping chunks of 512 tokens, with stride and maximum number of chunks tuned as hyperparameters. PubMedBERT encodes each chunk independently and produces a [CLS] vector.

In the second stage, this sequence of CLS vectors is passed through a Bi-GRU. The forward pass reads chunks from beginning to end, while the backward pass reads them in reverse. The final hidden states from both directions are concatenated, producing a document level representation that encodes both the temporal progression and the global context of the note 4.2. This vector is then passed to a linear classification head to predict ICD-10 codes. We refer to this model as `model_gru` throughout the rest of this thesis.

Hyperparameter optimisation We run 15 Optuna trials for chapter and 3-character using Bayesian search (TPE), with a single epoch per trial given the fast convergence of recurrent models. It would take too much time for the full task. The learning rate is searched on a logarithmic scale between $1 \times 10^{-5}$and 5×10^{-5}. The GRU hidden size is tuned over {128, 256, 512} , controlling the capacity of the sequential memory. Stride is searched over {128, 256, 384} tokens and maximum chunks over {4, 8, 12}. Batch size is restricted to {4, 6} due to the additional memory overhead of the recurrent layer. To prevent gradient explosion, we apply gradient clipping with a maximum norm of 1.0.

Due to its sequential nature, the Bi-GRU cannot parallelise computation across chunks and is therefore slower to train than mean pooling. For this reason, this architecture is evaluated only on the chapter and 3-character code tasks, and not on full ICD-10 code prediction.

Label-wise chunk Attention aggregation

Both mean pooling and Bi-GRU aggregation produce a single shared document representation used to predict all ICD codes simultaneously. Yet in a multi label setting, different diagnoses are typically supported by different parts of a discharge note, which a single shared vector cannot capture. Compressing the entire note into a single vector forces the model to mix signals that should remain separate.
To address this, we adopt a label-wise attention mechanism as the aggregation step, directly inspired by CAML [16] and its adaptations in PLM-ICD [4] (see Section 2.2.3 and Section 2.2.4). As described in the technical background 4.7.3, each ICD code is assigned its own query vector that attends independently over the chunk representations, producing a label-specific document vector used for classification. This means the model can focus on chunks describing pulmonary symptoms when predicting a respiratory code, and on entirely different chunks when predicting a metabolic or surgical code. This is a property that neither mean pooling nor the Bi-GRU can provide.
The overall architecture follows the same two stage design as the previous models. The discharge note is split into overlapping chunks of 512 tokens, each independently encoded by PubMedBERT, and the [CLS] vector of each chunk is extracted. The label-wise attention module then operates over this sequence of chunk representations. We refer to this model as `modelC_cls attention` throughout the rest of this thesis.

Hyperparameter optimisation We run 15 Optuna trials per granularity level using Bayesian search (TPE), with 4 epochs per trial. The learning rate is searched on a logarithmic scale between 1×10^{-5} and 3×10^{-5}. Stride is tuned over {256, 384} tokens, maximum number of chunks over {8, 12, 16}, and batch size over {4, 6}. For chapter and 3-character code tasks the objective is Micro-F1. For the full code task it is Recall@10.

Word-level Attention aggregation

All chunk-based models described so far rely on the [CLS] token as a summary of each chunk. While convenient, this design discards all token-level information within each chunk. The [CLS] vector is a single compressed representation of up to 512 tokens, and precise evidence such as a specific clinical term or a negation may be lost in that compression. To investigate whether attending directly over individual word representations brings a meaningful gain, we implement a word level variant of the label-wise attention mechanism. Instead of extracting only the [CLS] vector from each chunk, we retain the full sequence of token representations produced by PubMedBERT.
The K chunks are encoded independently and their token representations are concatenated into a single flat sequence of $K \times 512$ vectors, over which the label-wise attention operates directly. The attention mechanism follows the same label-wise formulation described in Section 4.7.3, where h_k now denotes an individual token representation rather than a chunk level [CLS] vector. Each label can therefore attend to any individual token across the entire document, rather than being limited to chunk-level summaries. The classification head remains identical to the chunk-level attention model.
This design comes at a significantly higher memory cost, as the attention matrix now operates over thousands of tokens rather than a handful of chunk vectors. For this reason, batch size is restricted to $\{4, 8\}$. We refer to this model as `modelC_cls allword` throughout the rest of this thesis.

Hyperparameter optimisation We run 25 Optuna trials for the chapter task and 10 for 3-character and full using Bayesian search (TPE), with 2 epochs per trial. The learning rate is searched on a logarithmic scale between 1×10^{-5} and 5×10^{-5}. Stride is tuned over $\{128, 256, 384\}$ tokens and maximum number of chunks over $\{4, 8, 12, 16\}$. For chapter and 3-character code tasks the objective is Micro-F1; for the full code task it is Recall@10.

5.3.4 Semantic chunking with LangChain

All chunking strategies described so far rely on a token-level sliding window. The discharge note is split into fixed size overlapping segments regardless of its linguistic structure. While simple and reproducible, this approach can cut a sentence midway, separate a diagnosis from its negation, or break a clinical expression across two chunks. Each chunk is therefore not guaranteed to be a semantically coherent unit, which may degrade the quality of the [CLS] representation extracted by PubMedBERT.
To investigate whether more linguistically informed chunking brings a meaningful gain, we test an alternative splitting strategy using LangChain's `RecursiveCharacterTextSplitter`. Rather than cutting at fixed token boundaries, this splitter attempts to split first on paragraph breaks, then on sentence boundaries, and finally on word boundaries, only falling back to character-level splitting if necessary. This ensures that each chunk respects the natural structure of the clinical note as much as possible. The chunk size is still controlled by the tokeniser to guarantee that no chunk exceeds 512 tokens, making it directly comparable to the token-level sliding window approach.
The attention architecture and classification head are kept strictly identical to the chunk level attention model described above, so that any performance difference can be attributed solely to the chunking strategy. We refer to this model as `modelC_cls_langchain` throughout the rest of this thesis.

Hyperparameter optimisation We run 10 Optuna trials per granularity level using Bayesian search (TPE), with 3 epochs per trial. The learning rate is searched on a logarithmic scale between 1×10^{-5} and 3×10^{-5}. The chunk overlap is tuned over {128, 256, 384} tokens, and the maximum number of chunks over {10, 12, 14, 16}. Batch size is tuned over {4, 6}. For chapter and 3-character code tasks, the objective is Micro-F1; for the full code task, it is Recall@10.

5.3.5 Clinical ModernBERT

All neural models described so far use PubMedBERT as the encoder. While PubMedBERT is well suited to biomedical text, it was pretrained on published abstracts and full-text articles rather than on clinical notes. To investigate whether a closer domain match brings meaningful gains, we replace PubMedBERT with Clinical ModernBERT (see Section 4.6.2) and evaluate it under two distinct settings, see Figure 5.5.

Setting 1: 512-token chunks with label-wise attention

In the first setting, the architecture is kept strictly identical to the chunk-level attention model described above. The same token-level sliding window chunking, the same [CLS] extraction, and the same label-wise attention aggregation with PubMedBERT simply replaced by Clinical ModernBERT. This allows a direct, controlled comparison between the two encoders. Any performance difference can be attributed solely to the choice of pretrained model rather than to any architectural change. We refer to this model as `modelC ModernBert` throughout the rest of this thesis.

Setting 2: Long-context chunks

The second setting exploits a key architectural advantage of ClinicalModernBERT over PubMedBERT: its extended context window of up to 8192 tokens (see Section 4.6.2). Rather than splitting the discharge note into 512-token chunks, we allow much larger chunks. This explores whether processing fewer but longer segments is beneficial for ICD-10 coding. In this setting, the chunk size itself becomes a hyperparameter, and the model is therefore exposed to much richer intra-chunk context than any of the previous models. The attention architecture and classification head remain identical.

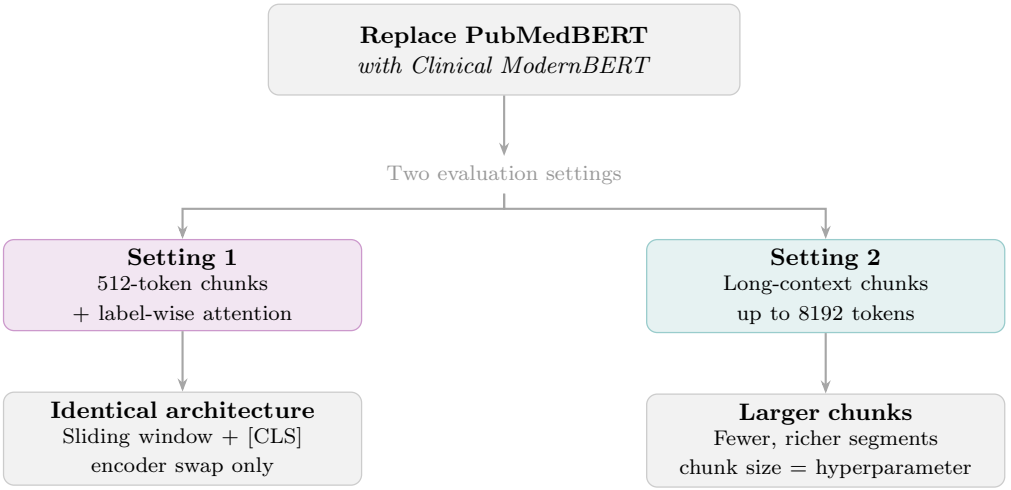


Figure 5.5: Overview of Clinical ModernBert as an encoder

Hyperparameter optimisation — Setting 1

We run 5 Optuna trials per granularity level using Bayesian search (TPE), with 2 epochs per trial. The learning rate is searched on a logarithmic scale between 1×10^{-5} and 5×10^{-5}. Stride is tuned over {128, 256, 384} tokens, maximum number of chunks over {4, 8, 12, 16}, and batch size over {2, 4, 6}, the latter being more constrained than PubMedBERT due to the higher memory footprint of ClinicalModernBERT.

Hyperparameter optimisation — Setting 2

We run 10 Optuna trials per granularity level with 1 epoch per trial. The learning rate is searched on a logarithmic scale between 1×10^{-5} and 5×10^{-5}. The chunk size is tuned over {512, 1024, 2048} tokens, the stride over {256, 512, 1024} tokens, the maximum number of chunks over {2, 4, 6, 8}, and batch size over {1, 2, 4}. Trials where the stride exceeds the chunk size are automatically pruned. For both settings and all granularity levels, the objective is Micro-F1 for chapter and 3-character code tasks, and Recall@10 for full code prediction.

5.3.6 Leveraging the ICD-10 hierarchy

All models described so far treat the three granularity levels independently. Yet the ICD-10 hierarchy provides a natural structure that can be exploited: a full code such as `I21.0` necessarily belongs to the 3-character category `I21`, which itself belongs to Chapter IX. Ignoring this structure means the model can predict a full code without predicting the corresponding chapter. This can produce incoherent outputs. We explore two strategies to incorporate this hierarchy: one is applied at inference time and one integrated directly into training.

Top-down hierarchical masking

The first approach reuses the three independently trained models, one per granularity level, and applies a top-down masking procedure at inference time. Given the predicted chapter probabilities, only the 3-character codes that belong to the predicted chapters are kept as candidates. Similarly, only the full codes that belong to the predicted 3-character categories are retained. This constrains the prediction space at each level using the decisions made at the level above. Coarser levels benefit from fewer and more frequent labels, making them easier to learn reliably. Propagating their predictions downward helps constrain the prediction space at finer levels, where the long-tail distribution of codes makes direct prediction harder. This strategy directly addresses the label imbalance problem: leveraging hierarchical structure has been shown to be particularly effective for rare codes [39].
To avoid errors at the top level propagating down and eliminating correct candidates, we introduce fallback mechanisms: if no chapter exceeds a confidence threshold τ_{chap}, the top-k chapters by score are retained instead. The same fallback applies at the 3-character level. All thresholds and fallback parameters are optimised on the validation set using Optuna over 50 trials, maximising Micro-F1. We refer to this model as `modelD_masking` throughout the rest of this thesis.

Hierarchical multi-task learning with context sharing

The second approach integrates the hierarchy directly into training through a multi-task architecture. Rather than training three separate models and combining them post hoc, a single shared PubMedBERT encoder processes the discharge note once. We use three label-wise attention heads to predict chapters, 3-character codes, and full codes simultaneously. Multi-task learning with a joint weighted loss has been shown to be effective for ICD coding [70, 71], as it allows the model to benefit from the correlations between related tasks rather than optimising each level in isolation. Beyond simply summing three weighted losses, we introduce a hierarchical conditioning mechanism between the attention heads. The intuition is that chapter-level predictions, being easier to learn due to a smaller and more balanced label space, can provide useful context to guide the harder task of predicting fine grained codes. The chapter head produces its label specific document representation $\mathbf{f}_{\text{chap}}$, defined as the mean of the label weighted chunk vectors. This representation is then passed through a bridge module. The bridge is a single linear layer followed by layer normalisation. This layer transforms the chapter-level context vector into the same dimension as the chunk representations so that it can be added to each of them before the 3-character attention head operates:

$$\mathbf{h}_{\text{cat3}} = \mathbf{h} + \text{Bridge}_1(\mathbf{f}_{\text{chap}}) \tag{5.5}$$

The same conditioning is applied between the 3-character and full-code heads:

$$\mathbf{h}_{\text{full}} = \mathbf{h}_{\text{cat3}} + \text{Bridge}_2(\mathbf{f}_{\text{cat3}}) \tag{5.6}$$

Each bridge consists of a linear projection followed by layer normalisation. This residual design ensures that, if the higher-level signal is not informative, the model can learn to ignore it and instead fall back to the original chunk representations. The joint loss is a weighted sum of the three binary cross-entropy losses:

$$\mathcal{L} = w_{\text{chap}} \cdot \mathcal{L}_{\text{chap}} + w_{\text{cat3}} \cdot \mathcal{L}_{\text{cat3}} + \mathcal{L}_{\text{full}} \tag{5.7}$$

where w_{chap} and w_{cat3} are tuned as hyperparameters, and the full-code loss is always weighted at 1.0 as it is the primary objective. To further address class imbalance, each loss uses a positive class weight, also tuned by Optuna. We refer to this model as `modelD_hierarchy` throughout the rest of this thesis.

Training follows a two-stage procedure inspired by prior work showing that selectively unfreezing the upper transformer layers while keeping lower layers frozen stabilises fine-tuning on multi-label classification tasks [72]. In the first stage, the PubMedBERT encoder is fully frozen, and only the attention heads and bridge modules are trained, allowing the classification layers to converge before any encoder weights are updated. The resulting checkpoint is then used to initialise the second stage, where the top n transformer layers are unfrozen and the full model is fine-tuned jointly with a lower learning rate for the encoder (1×10^{-6} to 2×10^{-5}) and a higher learning rate for the classification heads (5×10^{-5} to 5×10^{-4}). This layer wise discriminative strategy balances training stability and adaptation. A low learning rate preserves the encoder's pre-trained knowledge. The higher learning rate ensures rapid convergence of the task specific heads [73].This staged approach stabilises training given the relatively large number of parameters being optimised simultaneously.

Hyperparameter optimisation We run 40 Optuna trials using multivariate TPE with 10 startup trials, maximizing Recall@20 on the validation set over 5 epochs per trial, with early stopping with patience 4 during final training over 15 epochs. The searched parameters include the learning rate for the BERT encoder (1×10^{-6} to 2×10^{-5}) and for the classification heads (5×10^{-5} to 5×10^{-4}), the number of unfrozen BERT layers (2 to 6), dropout (0.0 to 0.4), batch size ({4, 8}), positive class weights for each level, and the loss weights w_{chap} and w_{cat3}.

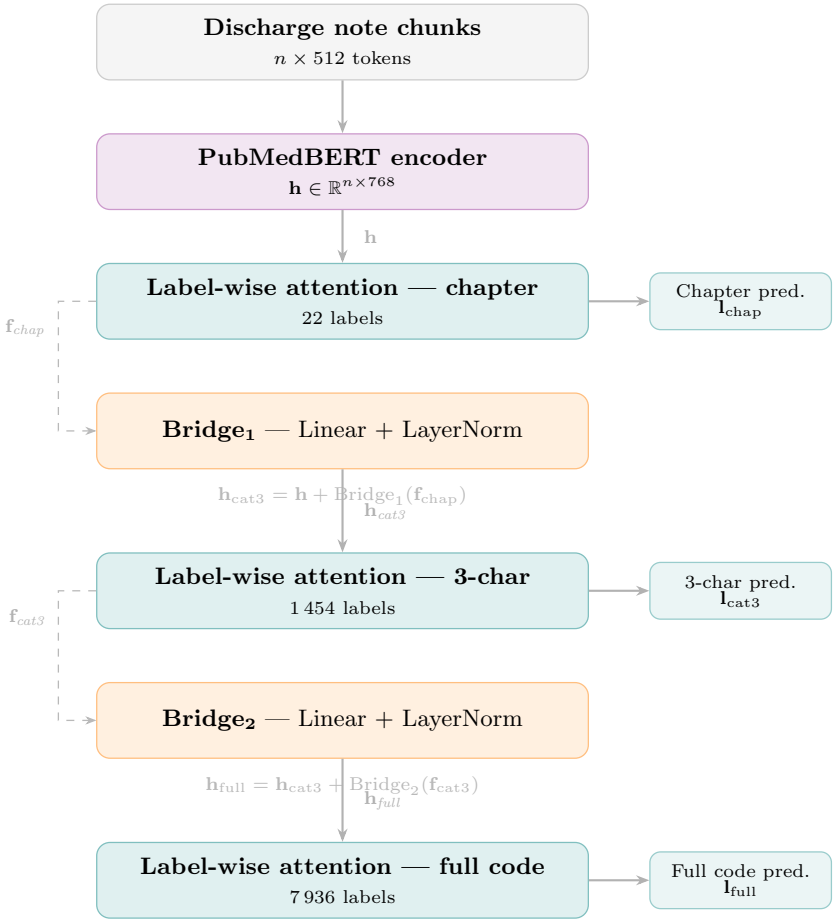


Figure 5.6: Forward pass of `modelD_hierarchy`. A single PubMedBERT encoder is shared across all three granularity levels. Each attention head predicts ICD-10 codes at increasing specificity (chapters → 3-char → full codes). Then, it passes its context vector ($\mathbf{f}_{\text{chap}}$, $\mathbf{f}_{\text{cat3}}$) residually to the next head. This allows to mitigate the long tail distribution of rare codes.

5.4 Hardware and environment

All experiments were conducted on the Lyra cluster, part of the CÉCI infrastructure (Consortium des Équipements de Calcul Intensif). A practical guide to reproduce these experiments is provided in Appendix A. CÉCI provides Belgian university researchers with access to high-performance computing (HPC) resources.
A single account grants SSH access to all CÉCI clusters [74]. Lyra is hosted at ULB. It is designed specifically for Machine Learning, AI, and High-Throughput Computing workloads.
The cluster consists of 40 virtual compute nodes. Each node provides one NVIDIA RTX 6000 ADA GPU (48 GB GDDR6, 18,176 CUDA cores), 32 CPU cores (AMD EPYC Genoa), and 128 GB of RAM. The maximum wall time per job is 5 days. You can find how to use it in the CECI documentation[1].
Lyra was chosen over other CÉCI clusters for two main reasons. First, it provides a dedicated GPU on each node, which is essential for training transformer-based models. Second, other GPU-capable clusters such as Hercules2 or Dragon2 either have fewer GPUs, older GPU generations, or are primarily designed for serial/SMP workloads. Clusters such as Lemaitre4 and NIC5 have no GPUs at all. For ICD-10 code classification using large language models, GPU acceleration is not optional. Training times without GPU would be prohibitive.

The Python environment was managed using a virtual environment activated in every job script. The core framework was PyTorch 2.9.1, combined with Hugging Face `Transformers` 4.57.3 and `Tokenizers` 0.22.1 for model loading, fine-tuning, and tokenisation. Hyperparameter search was

[1] CÉCI Consortium. Documentation and User Guide for High-Performance Computing resources. https://support.ceci-hpc.be/doc/, 2026. Accessed: 2026-03-28. The CÉCI is funded by the F.R.S.-FNRS and the Walloon Region.

performed with Optuna 4.8.0. Standard data processing relied on NumPy 2.3.5, Pandas 2.3.3, and scikit-learn 1.7.2. Scikit also provides evaluation metrics such as F1-score and classification reports.

Chapter 6

Model performance analysis

This chapter provides a comparative analysis of model performances across the two clinical coding objectives defined in the thesis.

First, Section 6.1 examines the "group of words to ICD codes" retrieval task. It evaluates how effectively lexical and deep learning methods resolve paraphrased diagnostic descriptions against exact ICD-10-CM codes. This section also explores near-miss analysis and chapter-level heatmaps to assess the clinical coherence of errors.

Second, Section 6.2 evaluates the document-level multi-label classification task. We compare our custom hierarchical architecture *modelD_hierarchy* against state-of-the-art models on different experimental setups from the literature. The section then analyses the impact of training imbalance, examines inference latency and architectural trade-offs. It concludes with a qualitative dive into model explainability, via chunk-level attention heatmaps and Leave-One-Out token-level analysis.

6.1 Group of words to ICD codes results

This section presents the evaluation results for the retrieval task described in Section 5.2. Eight models are compared on the synthetic dataset of 2,567 examples introduced in Section 5.1.3, covering all ICD-10-CM chapters. Results are organised around three axes: model family (lexical vs. neural), example difficulty (easy vs. hard), and retrieval depth ($k \in \{1, 5, 10\}$). We then perform a near-miss analysis that moves beyond exact code retrieval to examine whether models at least orient towards the correct clinical domain. We conclude with a qualitative chapter-level error analysis.

6.1.1 Global retrieval performance

	Accuracy@1	**Accuracy@5**	**Accuracy@10**	**MRR**
Levenshtein	0.219	0.297	0.330	0.252
Dice-Sørensen coefficient	0.314	0.454	0.517	0.373
T-LCS	0.231	0.334	0.374	0.274
W-LCS	0.219	0.308	0.347	0.258
Original SapBERT	**0.379**	**0.610**	**0.687**	**0.477**
Reimplemented SapBERT	0.368	0.596	0.673	0.467
PubMedBERT	0.147	0.226	0.266	0.181
ClinicalModernBERT	0.181	0.288	0.337	0.227

Table 6.1: Overall results of the models on the exact code retrieval task

Table 6.1 reports Accuracy@k and Mean Reciprocal Rank (MRR) for all eight models on the complete evaluation set. The results reveal three performance levels: the non-aligned neural models, the lexical baselines, and the aligned SapBERT models.

The most striking result in Table 6.1 concerns PubMedBERT and ClinicalModernBERT. **Both pretrained neural models fall well *below* the lexical baselines**. Although this placement is

counterintuitive, it has a clear explanation. Without any synonym-alignment training, the `[CLS]` embeddings produced by these models are not optimised for similarity search. The biomedical pre-training objective (masked language modelling on medical text) produces rich contextual representations, but provides no guarantee that synonymous expressions will be geometrically close to one another in the embedding space, which is exactly what the cosine similarity relies on. This suggests that alignment training on top of either architecture would be a natural next step, and is exactly proven by SapBERT results commented here below. It is nonetheless worth noting that the architectural and domain improvements of ModernBERT provide a measurable performance increase, improving the MRR by nearly 25% over PubMedBERT.

Lexical baselines come next in terms of performance. Among those, the Dice-Sørensen coefficient stands apart from the other three. Its MRR (0.373) is roughly 40% higher than Levenshtein (MRR 0.252), T-LCS (0.274), and W-LCS (0.258), which are barely distinguishable from one another. This gap within the lexical family is larger than what we expected. The failure of LCS methods to outperform plain Levenshtein distance is informative: the principal challenge in the dataset is not substring matching but semantic paraphrase. In this regime, all character-level metrics degrade together regardless of their formulation, leaving Dice's bigram overlap, insensitive to order, as the most forgiving measure.
We further observe that W-LCS underperforms T-LCS despite its more complex formula. This is likely a design mismatch: W-LCS was built to boost scores when one string is mostly contained within the other, with an asymmetric weighting that assumes the query is shorter than the target. A length difference analysis on the dataset shows that even though generated descriptions have a tendency to be shorter than the official ones, both often display comparable length. This means that the containment assumption rarely holds and the asymmetric weighting introduces noise rather than signal.

The original and reimplemented SapBERT clearly dominate all other models, with MRR scores of 0.477 and 0.467 respectively. These results are especially significant, as they imply that on average, the correct ICD-10-CM code is found between the first and second suggestions. The gap over the best lexical model (Dice-Sørensen coefficient) is over 10 percentage points. It is even larger then the unaligned neural models, confirming that self-alignment is the decisive factor, not the biomedical pretraining per se. The reimplemented SapBERT performs within 1–2 percentage points of the original checkpoint across all metrics, confirming the reproducibility of the training procedure described in Section 5.2.2.

To further understand these performance disparities, we examine the easy/hard split defined in Section 5.1.3, which uses BM25 rank as a proxy for lexical accessibility. Figure 6.1 breaks down the MRR by difficulty subset (overall, hard, easy) and confirms different dynamics across model families.

On easy examples, we notice the widening spread within the lexical family. The gap between Dice coefficient and the other baselines is around 15–20 percentage points, compared to a gap of less than 10 points on the hard subset. This is consistent with the earlier explanation of Dice's advantage. On easy examples, surface similarity becomes available and what separates models is how efficiently they exploit it. Dice's order-insensitivity gives it a structural edge over descriptions that are close but not identical. A minor word reordering or inflectional variation that barely changes meaning still disrupts Levenshtein and LCS-based scoring, while Dice absorbs it without penalty. When examples become more complex, the gap collapses because paraphrasing defeats all character-level methods indiscriminately, even though Dice degrades more gracefully than the others.
We observe that the SapBERT variants maintain their lead on the easy queries, but with a reduced margin over Dice-Sørensen. We also see that ClinicalModernBERT surfaces correct codes at higher positions than PubMedBERT, a gap that is more pronounced in this easy subset than when we first identified it within overall results. Both neural baselines nonetheless remain well below the lexical models, clear evidence that their weakness is structural rather than difficulty-dependent.

The hard subset reinforces this conclusion: PubMedBERT and ClinicalModernBERT are now collapsed to the bottom of the chart. The SapBERT variants retain MRR values of approximately 0.271–0.293, while Dice leads the lexical family at 0.15. The hard subset is the relevant evaluation regime and the one most representative of real clinical notes, where diagnostic expressions are rarely verbatim copies of official ICD terminology. Subsequent analysis will therefore focus

exclusively on this subset.

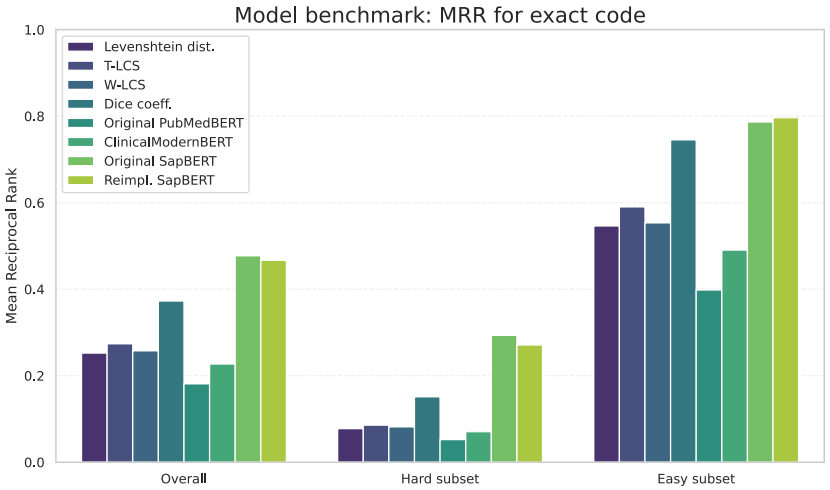


Figure 6.1: Mean Reciprocal Rank (MRR) for the eight models across difficulty subsets. Unaligned models remain at the bottom even on easy examples. Dice-coefficient lead over other lexical baselines is reduced across hard subset, but enhanced on the easy one. The advantage of SapBERT variants is clear on all examples, especially difficult ones.

Finally, a breakdown by generation source (Llama 3 vs. BioMistral) confirms that performance is essentially identical across both generators within each difficulty bin, with no systematic retrieval bias attributable to either model. This suggests that the results reflect fundamental model capabilities rather than generation artifacts. The plot displaying the details about this breakdown is in Appendix C.1.

6.1.2 Near-miss analysis

We decided to measure not only whether the exact code is retrieved, but also whether the model at least identifies the correct clinical domain. This distinction is motivated by the hierarchical structure of ICD-10-CM: a model that retrieves a code from the correct chapter or 3-character category has partially succeeded, in the sense that it has narrowed the search space for a human reviewer even when the precise code falls outside the top-k.
Figure 6.2 presents this analysis on the hard subset, by plotting chapter-level and 3-character accuracies against exact code accuracy, for $k \in \{1, 5\}$. Points above the diagonal indicate models that often identify the correct hierarchical level despite failing on the exact code. Points near the diagonal indicate models that rarely orient towards the right chapter or category without also finding the exact code.

At $k = 1$ (left panels), exact code accuracy is at its lowest for all models, which makes the vertical spread in chapter or category accuracy even more visible. Concerning chapter accuracy (top-left panel), although all models are significantly above the diagonal, the dissociation is largest for the SapBERT variants. Original SapBERT retrieves the exact code correctly only 18.8% of the time at Top-1, yet already points to the correct chapter 82.5% of the time. In the large majority of hard cases where SapBERT's first suggestion is wrong, it is still wrong within the correct clinical domain. Lexical baselines cluster in a tighter band at chapter accuracy 0.47–0.58, showing that even character-level methods carry some chapter-level signal in their top prediction.

As k increases to 5 (right panels), both axes rise, and points shift up and to the right. As more candidates are retrieved, all models accumulate both more exact hits and more correct-chapter and correct-category hits, and the latter consistently outnumber the former. At Top-5, SapBERT reaches a chapter accuracy of approximately 0.904–0.919. This means that **for almost every hard example, SapBERT retrieves at least one code from the right chapter in its first five suggestions**. The gap over the lexical family remains substantial, at roughly 20 percentage points, confirming that the near-miss advantage of neural models holds beyond Top-1.
We also computed the chapter *precision* at Top-5. Although not shown directly in the plots here,

it is visible in Appendix C.2. The precision is notably high for SapBERT (0.768), while the lexical baselines cluster in the 0.42–0.55 range. The best lexical model (Dice coefficient) thus recovers the correct chapter in roughly one in two suggestions for hard examples at Top-5, compared to three in four for SapBERT.

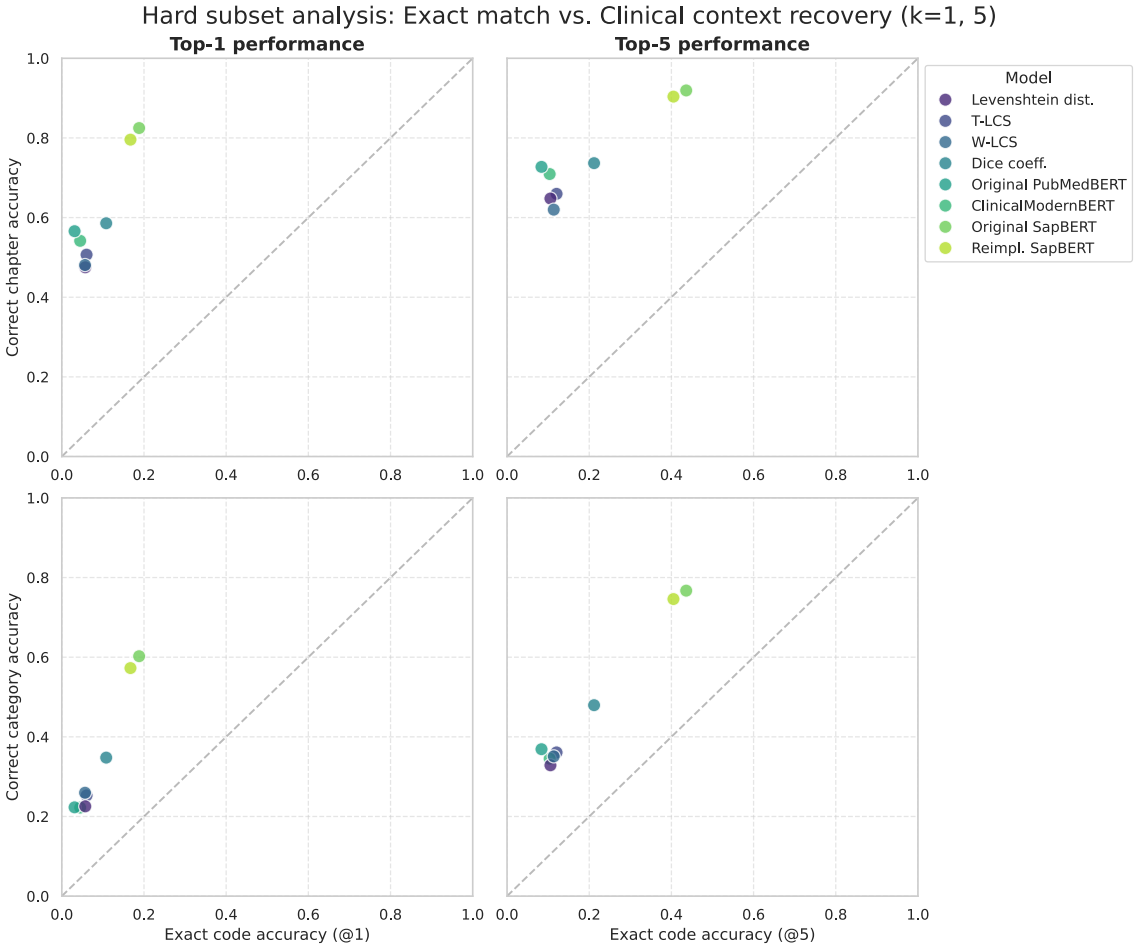


Figure 6.2: Near-miss analysis results for chapter and category precision on the hard subset. The x-axis shows exact code accuracy and the y-axis shows accuracy at a coarser hierarchical level: correct ICD-10-CM chapter (top row) or correct 3-character category (bottom row), at $k = 1$ (left) and $k = 5$ (right). All models lie well above the diagonal, confirming a consistent near-miss signal across model families. The gap is largest for SapBERT variants.

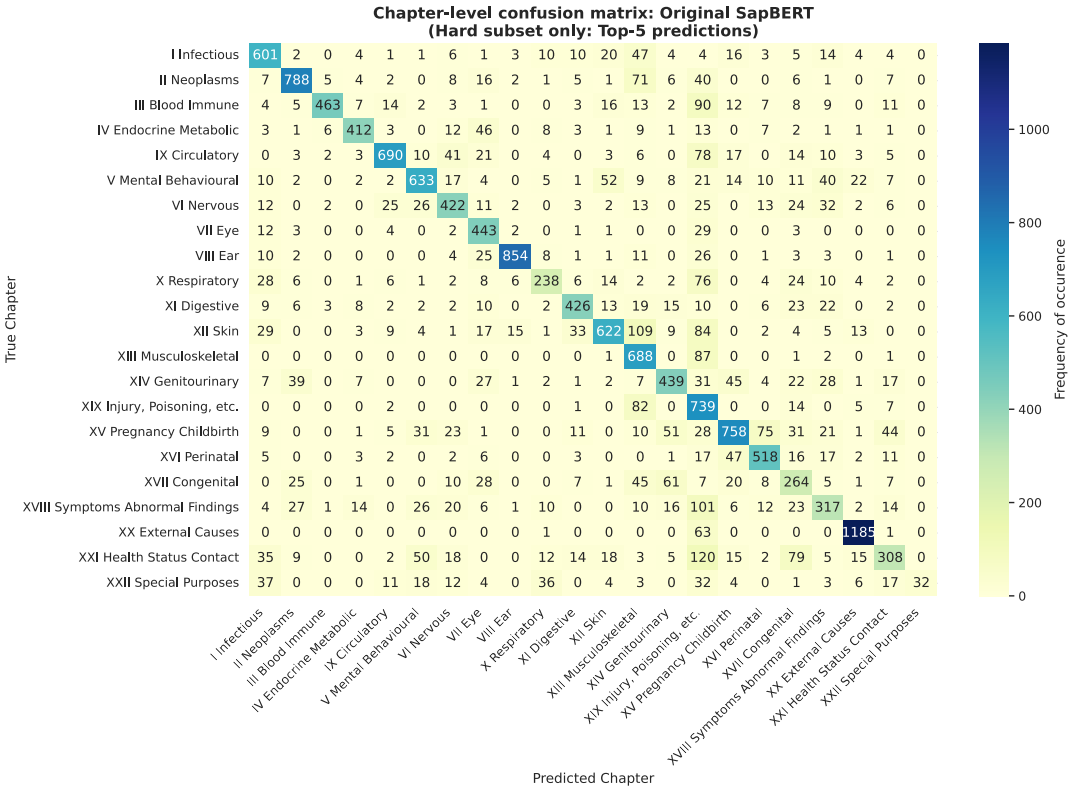
We furthermore flag that the unaligned neural models, despite falling behind the lexical baselines in terms of exact code accuracy, surpass the Levenshtein and LCS-based methods in chapter-level accuracy both at Top-1 and at Top-5, coming close to the performance of the Dice coefficient. This suggests that the MLM pretraining objective does encode enough clinical structure to point roughly in the right direction, even if it cannot resolve the fine-grained ranking needed for exact code retrieval.

The 3-character category analysis (bottom row) tells a consistent but more constrained story. As anticipated, the gap between category accuracy and exact code accuracy is smaller than at the chapter level, reflecting the finer granularity of the 3-character space. At Top-1, SapBERT variants achieve a category accuracy of 0.572–0.602 against an exact accuracy of 0.188, which remains a meaningful near-miss advantage. The Dice coefficient still leads the lexical family at both k values. PubMedBERT and ClinicalModernBERT now cluster with the weakest lexical models, offering a weak near-miss advantage at this 3-character level.

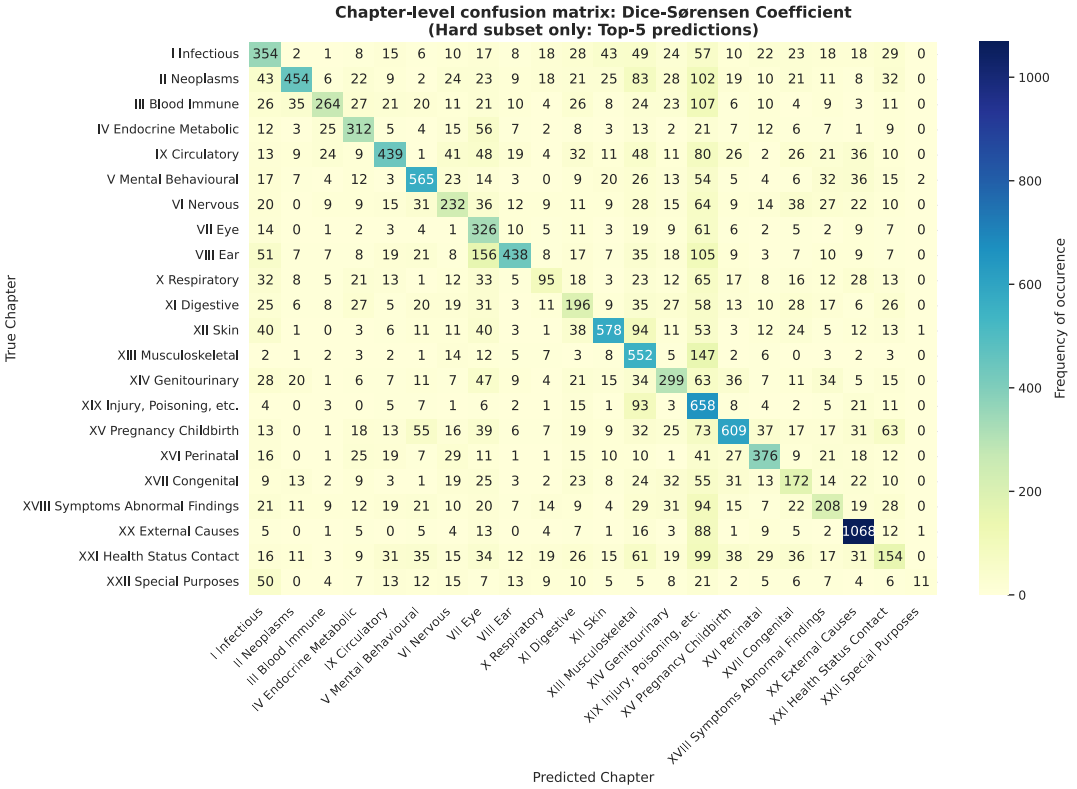
6.1.3 Chapter-level error analysis

Figures 6.3a and 6.3b display confusion matrices over the 22 ICD-10-CM chapters for original SapBERT and Dice-Sørensen coefficient respectively. They count all chapters appearing in the Top-5, relative to the true chapter encoded in the hard example. These figures complement the near-miss results with a qualitative view of where both models fail and why. We limit the discussion to the two top-performing models, in order to avoid unnecessarily cluttering the section. The confusion matrices for the other models can be found in Appendix C.

For the SapBERT model (6.3a), the diagonal is strongly dominant across nearly all chapters. Off-diagonal clusters are sparse, consistent with the high chapter precision reported in the near-miss analysis. The confusions that do occur are often clinically interpretable. For instance, Chapter XVII (Congenital) examples are sometimes erroneously classified into Chapter XIV (Genitourinary, 61 counts) or Chapter XIII (Musculoskeletal, 45 counts), reflecting real ICD-10 subcategories: congenital malformations of the genitourinary system and the musculoskeletal system, leading to shared terminology. Chapter XXI (Health Status Contact) similarly disperses into Chapter V (Mental Behavioural, 50 counts), Chapter XIX (Injury, Poisoning, and Certain Other Consequences of External Causes, 120 counts), and Chapter XVII (Congenital, 79 counts). This follows from Health Status Contact being more of an administrative chapter, sitting at the intersection of multiple chapters: it handles counselling and history of psychological trauma, along with aftercare following an injury, and genetic screening. A subtle asymmetry also appears between Chapter XII (Skin) and Chapter XIII (Musculoskeletal): Skin is predicted as Musculoskeletal 109 times, while the reverse occurs only once. Skin conditions over joints or extremities share anatomical terminology with musculoskeletal codes, but the reverse is rarely true. Finally, we note that Chapter XXII (Special Purposes) has a very diffuse confusion pattern, but this only reflects the very small number of evaluation examples rather than a systematic model failure.



(a) Original SapBERT



(b) Dice coefficient

Figure 6.3: Chapter-level confusion matrix for original SapBERT (top) and Dice coefficient (bottom) on the hard subset. Rows are true chapters, columns are predicted chapters. SapBERT's diagonal is strongly dominant, with clinically interpretable errors. Dice's diagonal is significantly weaker, with systematic over-prediction of Chapter XIX and less clinically coherent errors.

For the Dice coefficient (6.3b), the diagonal is present but considerably weaker, and the off-diagonal clusters are more dispersed across rows. We find again some of the clinically interpretable confusion pairs highlighted above, along with some additional masses (e.g. examples from Chapter VIII (Ear) misclassified in Chapter VII (Eye, 156 counts): despite referring to distinct organs, descriptions for ear and eye conditions share many short anatomical proximity terms, making them susceptible to bigram overlap confusion).
An observed consistent pattern is **misclassification of examples from unrelated chapters into Chapter XIX (Injury, Poisoning, and Certain Other Consequences of External Causes)**. The most concerned chapters are II Neoplasms, III Blood Immune, VII Ear, and XIII Musculoskeletal, with more than 100 error counts each. Although present in a smaller measure with SapBERT, it is very prominent in the Dice confusion matrix. This is likely driven by common non-anatomical keywords such as *injury*, *traumatic*, and *toxicity*, which appear across many chapter descriptions and create parasitic bigram overlap with codes from Chapter XIX.

Taken together, the two heatmaps add a qualitative dimension to the quantitative results. SapBERT's errors are sparse and concentrated in clinically coherent chapter boundaries, while Dice's errors are denser and more diffuse, driven more by lexical attractor effects with weaker clinical foundation. For an ICD coder, the nature of errors matters: a model whose mistakes are clinically interpretable is easier to audit and correct.

6.2 Full discharge note to ICD codes results

This section presents the results of our experiments on the full discharge-note-to-ICD-10 code assignment task. We first describe how we train and compare our best model against published baselines. We then analyse performance across granularity levels and code-frequency bins to assess how the hierarchical training objective impacts rare-code prediction. Finally, we examine computational cost and interpretability, providing a concrete view of where the model succeeds and

where it fails at the token and chunk levels.

6.2.1 External validation and Refined training protocol

To compare `modelD_hierarchy` against published models, we additionally train it on Setup B (described in Section 5.1.2). This requires a dedicated training protocol, distinct from the fixed learning rate procedure used for all models on our own MIMIC-IV split throughout this work.
Very low learning rates of the order of 10^{-6} have been shown to be critical for stable fine-tuning of BERT-based models on ICD coding tasks [75]. Following the schedule adopted by PLM-ICD [4], we therefore introduce a linear scheduler with warm-up. This leads to a more controlled and stable training trajectory, aligned with best practices in the literature.
Table 6.2 traces the evolution of our model across the three training stages on Setup B. Two observations stand out. First, micro-F1 stabilises early (around 0.41-0.42), suggesting the model has reached a plateau on frequent codes, which dominate this metric. Second, macro-F1 and recall metrics continue to improve steadily throughout all ten epochs. This divergence indicates that the model progressively learns to recover more rare codes without degrading performance on frequent ones. This is precisely the effect targeted by the hierarchical conditioning mechanism. Figure 6.4 confirms that the training loss decreases monotonically across all stages. This indicates that the model had not converged and that additional epochs would likely yield further gains on rare codes.

Training stage	Micro-F1	Macro-F1	P@5	P@8	R@10	R@50
Fixed LR (ep. 3)	0.412	0.068	0.576	0.505	0.354	0.667
+ Linear scheduler (ep. 7)	**0.416**	0.114	0.620	0.543	0.380	0.703
+ Linear scheduler (ep. 10)	0.415	**0.123**	**0.627**	**0.550**	**0.385**	**0.708**

Table 6.2: Evolution of `modelD_hierarchy` performance on the Setup B test set across the three training stages. All metrics are computed at threshold $\tau = 0.85$, optimised on the validation set. Micro-F1 stabilises after the first scheduler phase, while macro-F1 and recall metrics continue to improve, suggesting that further epochs would primarily benefit rare code prediction.

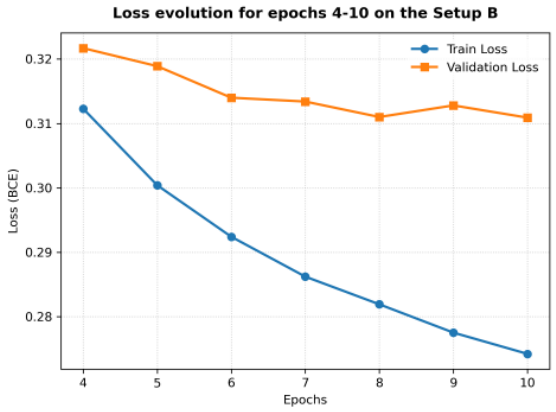


Figure 6.4: Loss evolution for epochs 4-10 on the Setup B. The training loss decreases monotonically across all stages (0.3123 → 0.2924 → 0.2742), confirming that the model has not converged.

6.2.2 Comparison with State of the art models

As detailed in Section 5.1.2, comparing ICD coding models across studies is inherently difficult due to differences in label space, split strategy, and code filtering. Having established the training dynamics on Setup B (Section 6.2.1), we now compare `modelD_hierarchy` against published baselines. Table 6.3 recalls the three experimental setups, and Table 6.4 presents the full comparison.
Micro-F1 (0.415 on Setup B, 0.428 on our setup). Micro-F1 is dominated by frequent codes, so it primarily reflects how well a model handles the most common diagnoses. Our scores (0.415–0.428) remain below all baselines on both setups. This gap is expected and directly tied to our architectural choice: label attention operates over 12 chunk-level `[CLS]` vectors rather than the full token sequence. By contrast, LAAT and PLM-ICD attend at the token level. CAML uses

local convolutional windows and RoSimTail-ICD uses cross-attention between label embeddings and the full note. All four approaches have access to finer-grained signals than our chunk-level representations.

As shown in Section 6.2.1, micro-F1 plateaus early during training and does not improve with additional epochs, confirming that this is a structural ceiling rather than a training issue. The architectural trade-off is intentional: our model runs 3.5× faster than PLM-ICD at inference (Section 6.2.6), which matters in a deployment setting.

	Setup A [59]	**Setup B** [5, 60]	**Our Setup** 5.1.2
Dataset	MIMIC-IV	MIMIC-IV	MIMIC-IV
Train / Val / Test	110,442 / 4,017 / 7,851	88,969 / 13,310 / 19,999	75,133 / 23,608 / 23,534
Unique ICD-10 codes	25,230	7,942	9,683
Split strategy	Random	Patient-level	Patient-level
Code guarantee in all splits	No	Partial (∼0.5% missing in val)	Near-complete (<0.16%)
Code type	Diagnosis only	Diagnosis + Procedure	Diagnosis only
Code filtering	—	≥ 10 occurrences	≥ 3 patients

Table 6.3: Comparison of experimental setups. All three use MIMIC-IV ICD-10 on the full code task but differ in label space size, split strategy, and code coverage. Setup A risks data leakage through admission-level random splitting. Setup B and our setup both use patient-level splitting; our setup additionally retains a 22% larger label space and achieves near-complete code coverage across all splits.

Model	**Micro-F1**	**Macro-F1**	**P@5**	**P@8**	**P@10**	**R@10**	**R@20**	**R@50**
Setup A — *MIMIC-IV-ICD benchmark (25,230 codes, 110k/4k/8k split)*								
CAML	0.527	0.041	–	0.644	–	–	–	–
LAAT	0.554	0.045	–	0.670	–	–	–	–
PLM-ICD	0.570	0.049	–	0.695	–	–	–	–
Setup B — *MIMIC-IV ICD-10 (7,942 codes, 122k docs, patient-level split)*								
CAML	0.554	0.160	–	0.668	–	–	–	–
LAAT	0.579	0.203	–	0.689	–	–	–	–
PLM-ICD	0.585	0.211	–	0.699	–	–	–	–
RoSimTail-ICD	**0.593**	**0.293**	–	**0.713**	–	–	–	–
modelD_hierarchy†	**0.415**	**0.123**	**0.627**	**0.550**	**0.508**	**0.385**	**0.535**	**0.708**
Our setup — *MIMIC-IV ICD-10 (9,685 codes, 122k docs, patient-level split)*								
modelD_hierarchy	**0.428**	**0.068**	**0.615**	**0.534**	**0.492**	**0.394**	**0.539**	**0.700**

† Trained on Setup B label space following the dedicated training protocol (Section 6.2.1).

Table 6.4: Performance comparison across three experimental setups (see Table 6.3) using various metrics 4.8. Our model operates on chunk-level `[CLS]` representations (12 chunks of 512 tokens) rather than word-level attention. This limits the accuracy but reduces inference cost. And it is the only model reporting recall@k: R@50 = 0.708 on Setup B and 0.700 on our setup, recovering 70% of true codes within the top-50 predictions.

Macro-F1 (0.123 on Setup B, 0.068 on our setup). Macro-F1 weights all codes equally regardless of frequency, making it our most informative metric for rare-code performance. On Setup B, our macro-F1 of 0.123 approaches CAML (0.160) and continues to improve throughout training as shown in Table 6.2. This confirms that the hierarchical conditioning mechanism contributes directly to rare-code recovery. Full codes benefit from the supervision signal of their more frequent parent codes at the chapter and 3-character levels. The gap with RoSimTail-ICD (0.293) reflects a difference in design philosophy. RoSimTail-ICD targets the long tail through dedicated modules (SSDC and MATR), whereas our model addresses imbalance only through the ICD-10 hierarchy, without auxiliary losses or multi-stage training.

Top-k precision (P@5, P@8, P@10). Our P@8 of 0.550 on Setup B remains below baselines (0.668-0.713), again consistent with the chunk-level granularity trade-off described above.

Recall (R@10, R@20, R@50). This is the metric most directly relevant to our deployment goal. Recovering 70% of the true code set within the top 50 predictions makes our model well suited to a human-in-the-loop workflow, where completeness matters more than precision alone. Our training objective explicitly prioritised recall, since missing a relevant code is costlier than suggesting an extra one for a human reviewer to discard. This is reflected on both setups. R@50 reaches 0.700 on our setup and 0.708 on Setup B. No published baseline in this comparison reports recall@k, which suggests this dimension has been largely overlooked in the literature.

Inference time and architectural trade-offs

Table 6.5 benchmarks inference time across all models on the same NVIDIA RTX 6000 Ada Generation GPU. Among transformer-based models, **our model (modelD_hierarchy) is the fastest**, running 3.5× faster than PLM-ICD and 2.1× faster than RoSim-Tail despite sharing the same PubMedBERT backbone. This follows directly from two architectural choices summarised in Table 6.6. First, a reduced chunk count (12 vs. 32). Second, label attention operates over chunk-level `[CLS]` representations rather than the full token sequence, substantially reducing the size of the attention matrix. RoSim-Tail adds further latency through its two auxiliary modules (SSDC and MATR).
CAML and LAAT are faster still due to their lightweight CNN/LSTM architectures (15M and 37M parameters vs. 125M+ for transformers), but this speed advantage comes at a cost in rare-code performance.
The key point is not that our model is the simplest model, but that its single structural addition — the ICD-10 hierarchy — requires neither auxiliary losses, nor multi-stage training, nor external alignment module. In a production healthcare setting, where models must be retrained on updated ICD versions, audited, and explained to non-technical stakeholders, this simplicity has real operational value beyond benchmark numbers.

Model	Mean (ms/doc)	P95 (ms/doc)	Throughput (doc/s)
CAML	3.1	3.1	317.1
LAAT	20.7	31.6	46.3
ModelD_hierarchy *	**53.3**	**55.3**	**16.7**
RoSim-Tail *	114.7	—	8.7
PLM-ICD *	184.4	185.6	5.4

Table 6.5: Inference time per batch (RTX 6000 Ada). Among transformer-based models (*), our modelD_hierarchy is the fastest. The speed is driven by a reduced chunk count and a compact label attention over `[CLS]` representations rather than the full token sequence.

6.2.3 Overview of model configurations

Figure 6.5 presents the variants of our model. Each row corresponds to a different model variant. The first eight columns capture the key architectural and training choices: chunking strategy, pre-trained encoder, aggregation mechanism and multi-task hierarchy. The last three columns of Table 6.5 report micro-F1, macro-F1, and recall@20 on the full code task, our primary evaluation target. All metrics use the optimal decision threshold selected for each model on the validation set by maximising micro-F1 (see Table 6.7). You can find the results for all our trained models across all metrics, beyond just micro-F1, macro-F1, and recall@20 in Appendix D

These metrics are not equally informative in our setting. Micro-F1 is dominated by frequent codes and remains compressed across all valid models (0.39–0.46), making it a poor discriminator. Macro-F1 and recall@20 are more interesting. The former reflects on how well a model handles rare codes. The latter shows whether the correct codes appear in a short candidate list. They are both directly relevant to the coding-assistance use case.
These columns already tell a clear story. Macro-F1 ranges from 0.024 to 0.068 and recall@20 from 0.49 to 0.54, with our modelD_hierarchy leading on both. And model marked 0† produced no predictions due to the instability of the Optuna optimisation.

Feature	CAML	LAAT	PLM-ICD	RoSimTail -ICD	modelD hierarchy
Encoder					
Pre-trained language model	No	No	Yes	Yes	Yes
Biomedical domain	No	No	Yes	Yes	Yes
Word embeddings (Word2Vec / fastText)	Yes	Yes	No	No	No
Recurrent encoder (BiLSTM)	No	Yes	No	No	No
Convolutional encoder (CNN)	Yes	No	No	No	No
Long document handling					
Chunking strategy	No	No	Yes	Yes	Yes
Chunk size (tokens)	—	—	128	128	**512**
Max chunks	—	—	24	32	**12**
Max sequence length (tokens)	—	4000	3072	4096	**6144**
Attention input: token-level	Yes	Yes	Yes	Yes	No
Attention input: chunk `[CLS]` only	No	No	No	No	Yes
Classification head					
ICD hierarchical modelling	No	No	No	No	Yes
Long-tail specific mechanism	No	No	No	Yes	Part.
Semantic alignment correction (SSDC)	No	No	No	Yes	No
Multi-stage tail refinement (MATR)	No	No	No	Yes	No

Table 6.6: Architectural comparison across models. Our `modelD_hierarchy` relies on a single structural prior (ICD-10 hierarchy) with label attention over 12 `[CLS]` representations, versus full token-sequence attention over 32 chunks for PLM-ICD and RoSim-Tail, the latter further adding SSDC and MATR modules.

Model Architecture (Full Code)	Chunking	PubMedBERT	ModernBERT	Agg: Mean	Agg: Bi-GRU	Agg: Attn All	Agg: Attn CLS	Multi-task	Micro F1 (Full)	Macro F1 (Full)	Recall@20 (Full)
TF-IDF	✗	✗	✗	✗	✗	✗	✗	✗	**0.456**	0.039	0.524
modelB mean	✓	✓	✗	✓	✗	✗	✗	✗	0.429	0.024	0.489
model_gru	✓	✓	✗	✗	✓	✗	✗	✗	—	—	—
modelC_cls allword	✓	✓	✗	✗	✗	✓	✗	✗	0†	0†	0.144
modelC_cls attention	✓	✓	✗	✗	✗	✗	✓	✗	0.433	0.031	0.519
modelC ModernBERT	✓	✗	✓	✗	✗	✗	✓	✗	0.445	0.042	0.527
modelD masking	✓	✓	✗	✗	✗	✗	✓	✗	0.446	0.033	0.533
modelD hierarchy ★	✓	✓	✗	✗	✗	✗	✓	✓	0.429	**0.068**	**0.539**

Figure 6.5: Architecture of the trained models with micro-F1, macro-F1, and recall@20 on the full-code task (optimal threshold per model, selected on validation set). Macro-F1 and recall@20 are the primary metrics; micro-F1 is dominated by frequent codes and is uninformative. **ModelD_hierarchy leads on both and is selected for deployment.** Models marked $0^{\dagger}$ produce no predictions due to Optuna instability.

Model	Threshold (Full)
TF-IDF	0.19
ModelB_mean	0.22
ModelC_cls attention	0.23
ModelC_cls allword	0.50
ModelC_cls_langchain	0.18
ModelC_ModernBERT	0.30
ModelD_masking	0.22
ModelD_hierarchy	0.88

Table 6.7: Optimal decision thresholds per model, optimised on the validation set

Progressive improvements. TF-IDF serves as our baseline (recall@20 = 0.524, macro-F1 = 0.039). We introduce PubMedBERT with mean pooling in modelB_mean but it fails to outperform the baseline (recall@20 = 0.490, -6.5%), showing that naive aggregation is insufficient. In model_gru, we replace mean pooling with a Bi-GRU over chunk representations, but it was not trained on the full-code task due to computational constraints and is therefore only evaluated at the chapter and 3-character levels. Replacing mean pooling with attention-based aggregation (modelC_cls_attention) recovers performance (recall@20 = 0.519, -1% vs. baseline). We also tested a LangChain based chunking strategy (`modelC_cls_langchain`), which uses a recursive character text splitter instead of our fixed size chunking. This did not improve over `modelC_cls attention` and was not retained for further development

Swapping the encoder for ClinicalModernBERT (modelC_ModernBERT) yields our best single-encoder result (recall@20 = 0.527, macro-F1 = 0.042), though the gain likely reflects hyperparameter tuning; subsequent models retain PubMedBERT. A second setting exploiting ClinicalModernBERT's extended context window with larger chunks was also evaluated, see Section 5.3.5. The model produced near zero predictions because Optuna hyperparameter search failed to converge. The model was therefore excluded from further analysis.

Cascaded masking (ModelD_masking). This is our first attempt to exploit the ICD hierarchy. The idea is simple: predictions are used from a coarser level to eliminate implausible codes at the next finer level. Concretely, chapter predictions block impossible 3-character codes, which in turn restrict the set of full-code. This is applied at inference time on top of our already trained modelC_cls_attention. It improves recall@20 to 0.533 (+1.7% vs. baseline) but macro-F1 remains weak (0.033, -15% vs. baseline), and three encoder passes make it costly.

Multi-task hierarchy (ModelD_hierarchy). Our modelD_hierarchy resolves this limitation by consolidating the three models into a single multi-task architecture: one encoder pass feeds three classification heads that jointly predict chapters, 3-character codes, and full codes. Higher-level predictions regularise the full-code head, particularly for rare codes. This yields macro-F1 = 0.068 (+106% vs. modelD_masking, +74% vs. baseline) and recall@20 = 0.539 (+1.1% vs. modelD_masking, +2.9% vs. baseline), at the cost of a small drop in micro-F1 (0.429 vs. 0.446). It is our strongest model and the system selected for deployment.

6.2.4 Performance across granularity levels

Figure 6.6 reports macro-F1, recall@20, and micro-F1 across the three granularity levels for selected model variants. **All metrics decrease as granularity increases:** macro-F1 falls from 0.55-0.74 at the chapter level to 0.13-0.28 at 3-character, and collapses to 0.02-0.07 at full code. This reflects the long-tail distribution of ICD-10 full codes: most codes rarely appear in the training set and are therefore difficult to predict individually.

Higher granularity levels are more reliable. Chapter and 3-character predictions are more accurate than full code predictions. This makes hierarchical modelling valuable: if coarser levels are easier to predict, their outputs can guide full code prediction. This is precisely the approach taken in our modelD_hierarchy : it jointly trains three classification heads (chapter, 3-character, full code) within a single model, so that higher level predictions regularise the full code head. The result is the highest macro-F1 at both the chapter (0.74) and full-code levels (0.068, +74% vs. TF-IDF), confirming that joint training directly helps with rare codes.

Strong intermediate-level performance does not guarantee full-code performance. Our modelC_cls_allword achieves the best 3-character recall@20 (0.78) and macro-F1 (0.28), yet collapses at the full-code level due to training instability. This serves as a reminder that each granularity level must be validated independently.

6.2.5 Impact of training imbalance

Figure 6.7 shows performance by ICD-10 code frequency in the training set, across four bins : very rare (<10 occurrences), rare (10–50), medium (50–500), and frequent (>500). The four panels report micro-F1, macro-F1, recall@10, and the gap between recall@10 and micro-F1.

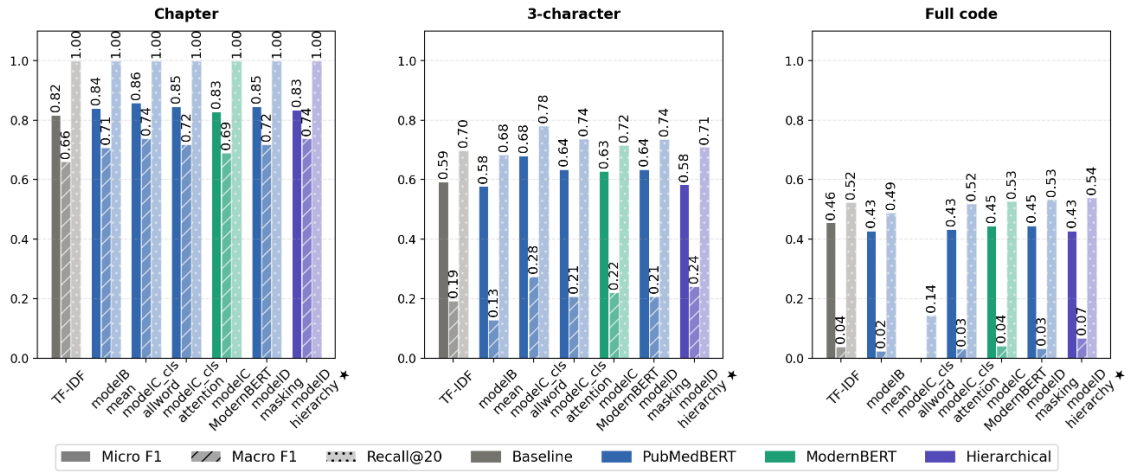


Figure 6.6: Performance across granularity levels for selected model variants. All metrics drop with granularity, reflecting the long-tail distribution of ICD-10 full codes.

All our models collapse on rare codes. Micro-F1 exceeds 0.48 on frequent codes but falls below 0.02 on very rare ones. Macro-F1 is near zero on both rare bins for most of our models, meaning they simply fail to predict those codes. This is a direct consequence of the long-tail distribution: 5,244 very rare codes contribute almost nothing to the training loss, while 366 frequent codes dominate it.

ModelD_hierarchy is our only model that actually learns rare codes. While all other models effectively fail : macro-F1 between 0.0001 and 0.0002 on very rare codes, our modelD_hierarchy reaches 0.0062, a substantial improvement. On the rare codes bin, the gap is equally striking: 0.0768 versus 0.0116-0.0235 elsewhere. Hierarchical training channels useful signal from coarser heads down to the full-code head, giving the model a fighting chance even on codes seen only a handful of times.
The **gap panel** (recall@10 - micro-F1) reveals a further distinction between representation quality and prediction confidence. A large gap means the model ranks the correct code in its top 10 but does not commit to predicting it. The representation is correct, but the confidence score falls below the decision threshold. This gap peaks in the rare bin (10-50) for all models, reaching 0.32 for TF-IDF and around 0.27 for neural models, then collapses to near zero on frequent codes where confidence and coverage converge. **This suggests that the bottleneck is not in how the model ranks codes, but in how confidently it scores them.** A consequence of seeing rare codes so infrequently during training is that the model never learns to predict them with high probability, even when it identifies them as plausible candidates.
Finally, TF-IDF shows a surprisingly competitive recall@10 on very rare codes (0.146), above several neural models. Sparse representations naturally preserve the lexical specificity of rare codes : unusual terms that appear in few notes tend to be strong identifiers. Neural models learn smoother, more generalised representations that benefit frequent codes but can dilute these distinctive signals. This is a limitation that modelD hierarchy partially compensates for through hierarchical regularisation.

6.2.6 Computational cost

Training time

Training times were measured on an NVIDIA RTX 6000 Ada Generation GPU. Model_C_cls_attention required 20h13 for 5 epochs on the full task, serving as the reference encoder. ModernBERT required approximately 2 days for 5 epochs on the chapter and 3-char tasks, making it significantly more expensive than Model_C_cls_attention (1 day and 15h for the same task).
Model D (hierarchical) trained for 10 epochs in 26h25, which is remarkably efficient given that it jointly learns all three classification tasks simultaneously : chapter, 3-char, and full code in a single training run. In practice, Model D's single 26h25 run replaces what would otherwise be three independent training jobs, making it the most cost-effective architecture when the full multi granularity pipeline is considered.

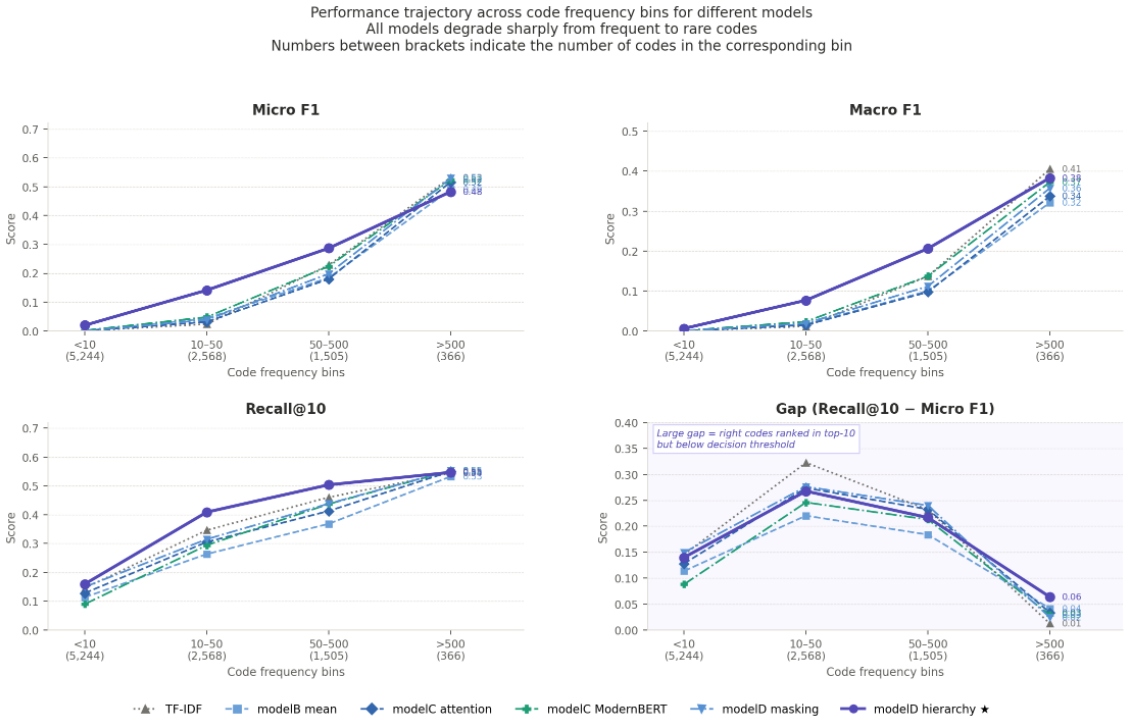


Figure 6.7: Performance trajectory across code frequency bins for different models. All models collapse on rare codes; ModelD_hierarchy is the exception, outperforming others by a large margin on macro-F1. The gap panel (recall@10 - micro-F1) reveals that rare codes are ranked correctly but not predicted, pointing to a confidence calibration issue.

Inference time

Table 6.8 reports inference times over 200 documents. All neural models run in 50-67 ms/doc. The exception is ModelD_masking (183.6 ms/doc), which requires three sequential encoder passes : one per granularity level. ModelD_hierarchy avoids this by consolidating everything into a single pass, keeping latency low (53.3 ms/doc). TF-IDF is the slowest at 403.5 ms/doc, though it runs on CPU and is not directly comparable.

Model	ms/doc	doc/s
TF-IDF (CPU)	403.5	2.5
ModelB_mean	49.9	20.0
ModelC_cls_attention	51.3	19.5
ModelC_ModernBERT	67.2	14.9
ModelD_masking	183.6	5.4
ModelD_hierarchy	**53.3**	**16.7**

Table 6.8: Inference time per document measured on an NVIDIA RTX 6000 Ada Generation GPU over 200 documents. TF-IDF runs on CPU (scikit-learn) and is not directly comparable. ModelD_hierarchy offers the best performance-speed trade-off among the proposed architectures.

6.2.7 Attention heatmap

Figure 6.8 shows the attention heatmap for a note from the test set. Each row corresponds to a true ICD-10 code assigned to the note. The y-axis label indicates whether our modelD_hierarchy correctly predicted the code (✓) or missed it (×). Each column is a text chunk of the note. Each cell shows the normalised attention weight the model assigns to that chunk when predicting that code : **the brighter the cell, the more the model relies on that chunk.** This visualisation allows us to inspect whether the model focuses on the relevant parts of the note when making a prediction.

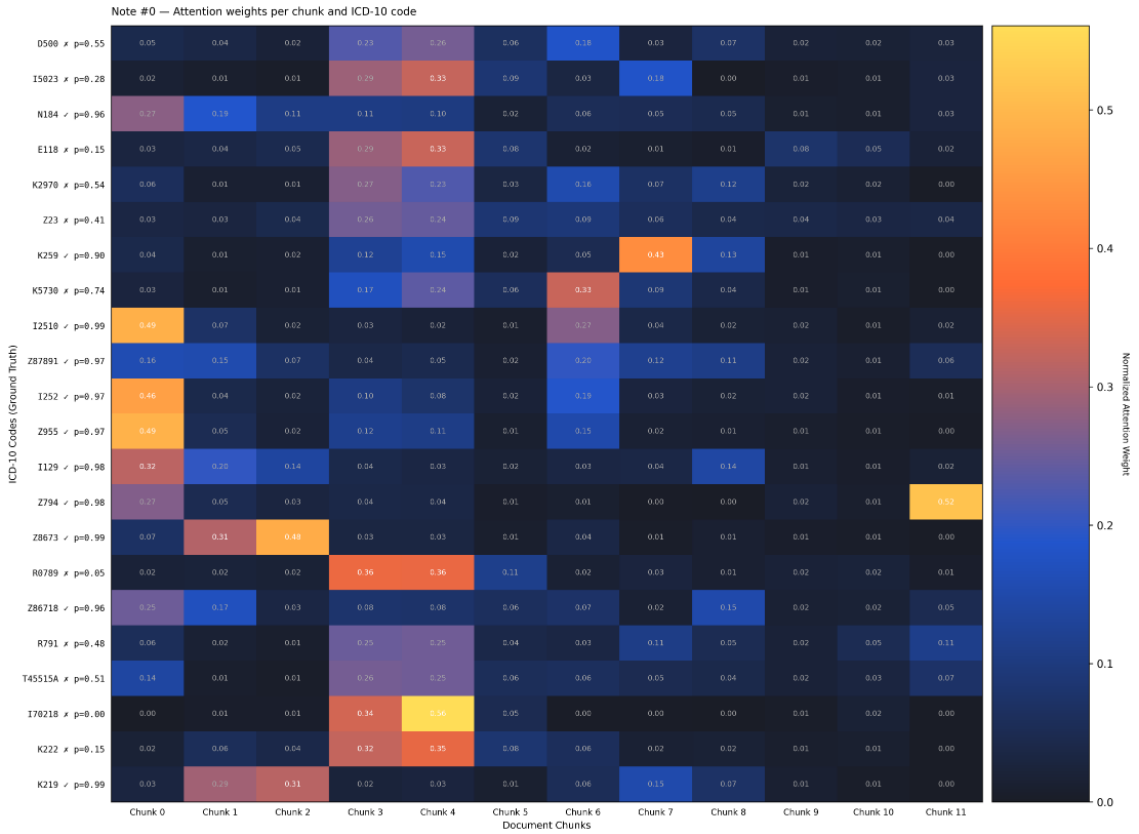


Figure 6.8: Attention heatmap for a note from the test set. Each row is a true ICD-10 code (✓ correct, × missed); each column is a text chunk. Cell brightness reflects how much the model relies on that chunk. Sharp, localised attention correlates with correct predictions; diffuse or misallocated attention leads to failures.

When attention is sharp, predictions are correct. For lexically explicit codes, the model focuses on the right chunk. I2510 (CAD/coronary artery disease, p=0.99) spikes on chunk C0 (weight: 0.49), which contains "cad s/p distant mi", where p denotes the confidence score assigned by the model to that code for this note.
K259 (gastric ulcer, p=0.98) concentrates on chunk C7 (weight: 0.43), containing "gastric ulceration: continued on home pantoprazole bid." In both cases, the evidence is explicit and spatially concentrated in a single chunk.

When attention is diffuse or misallocated, predictions fail. For I5023 (heart failure, p=0.28), evidence is spread across multiple chunks : "flash pulmonary edema," "bilateral pleural effusions" but attention remains diffuse (C2 attention weight: 0.006, C3 attention weight: 0.291, C4 attention weight: 0.326) and the model misses the code. The problem here is not the location of the evidence but its distribution: no single chunk concentrates enough signal to trigger a confident prediction.
For E118 (Type 2 Diabetes Mellitus with complications, p=0.15), the failure is different. Attention correctly focuses on chunks where "diabetes" appears, but predicting this specific code requires linking that mention to "70/30 insulin". This evidence appears in a different chunk that specifies the complication. The model misses this cross-chunk reasoning step, likely compounded by the code's rarity (0.33% in training).

6.2.8 Leave-One-Out token analysis

The Leave-One-Out (LOO) analysis measures how much each token contributes to the prediction score within a given chunk. All analyses in this section are performed on `modelD_hierarchy`, our best model selected for deployment. Concretely, each token is removed one at a time, and we record the change in prediction probability: $\Delta p = p_{\text{base}} - p_{\text{without token}}$. A large positive Δp means the token is useful : removing it actually hurts the prediction (shown in red). A negative Δp means the token is noise : removing it actually improves the prediction (shown in green). Each figure displays the 30 tokens with the largest absolute impact. Note that the final prediction score

aggregates all chunks weighted by attention; the LOO analysis therefore reveals what drives each chunk's contribution, rather than the final score directly.
We analyse three representative cases (Figures 6.9 to 6.13) illustrating both the strengths and failure of our model.

Single dominant token. The heatmap in the section above showed that attention is sharp and well-localised on the right chunk for I2510 and K259. The LOO explains why: a single token carries the entire signal. For I2510, "cad" alone drives the prediction ($\Delta p \approx +0.023$, Figure 6.9); for K259, "ulceration" dominates at $\Delta p \approx +0.170$ (Figure 6.10). One explicit keyword is sufficient for the model to be confident.

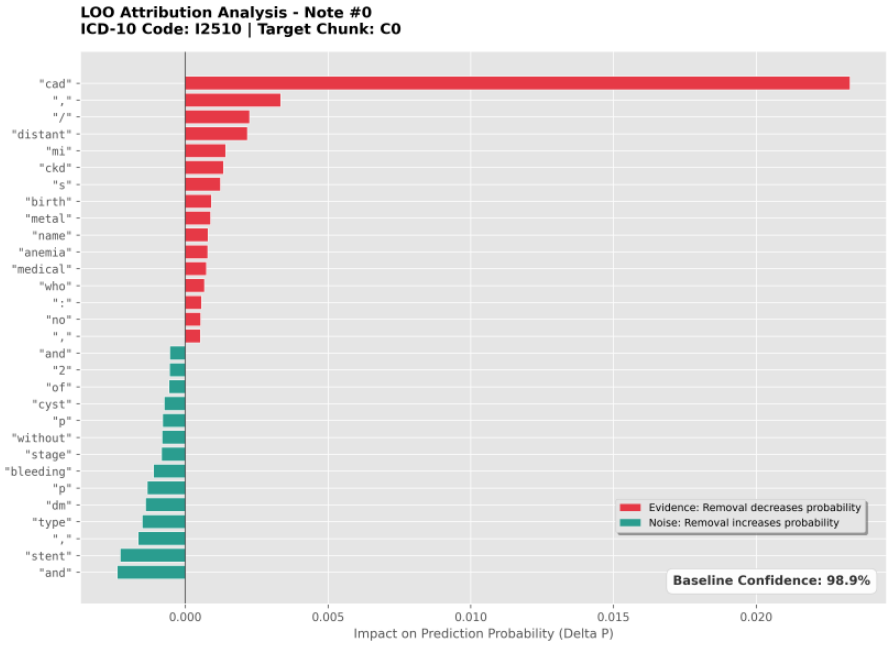


Figure 6.9: LOO analysis for code I2510 (CAD) in chunk C0, see 6.2.8 for more details. The token "cad" alone drives the prediction ($\Delta p \approx +0.023$), confirming that a single explicit keyword is sufficient for a confident prediction.

Vocabulary is not the bottleneck. For I5023 (heart failure), the model succeeds on Note 2 (p=0.99) where evidence is concentrated in one chunk, but fails on the note from the heatmap (p=0.28) where the same tokens are spread across chunks (Figure 6.11). The model knows the relevant vocabulary however the failure in Note 1 is due to an attention aggregation problem.

Failed prediction. For E118 (Type 2 Diabetes Mellitus with complications, p=0.15), the most attended chunk C4 is dominated by irrelevant lab tokens : "pitting," "wbc," lab markers and several act as strong noise (Figure 6.12). In chunk C10, "neuropathic" and "nifedipine" provide weak useful signals, but since C10 receives little attention, their contribution to the final score remains negligible (Figure 6.13). The failure combines two problems: the wrong tokens dominate the most attended chunk, and the right signals are buried in low-attention chunks.

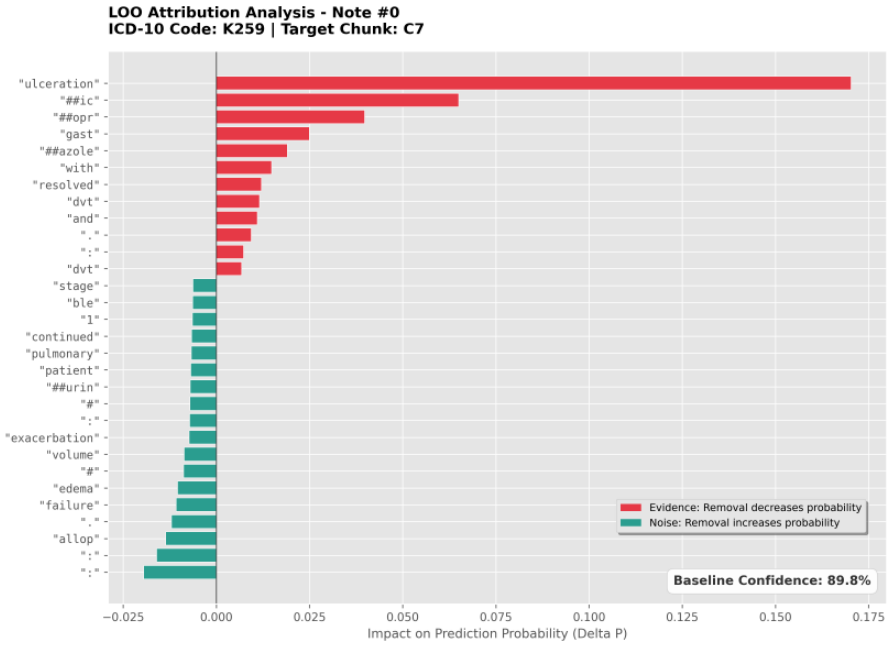


Figure 6.10: LOO analysis for code K259 (gastric ulcer) in chunk C7. "ulceration" dominates the prediction ($\Delta p \approx +0.170$). Tokens such as "failure" and "edema" appear as noise : unrelated diagnoses co-occurring in the same chunk create interference.

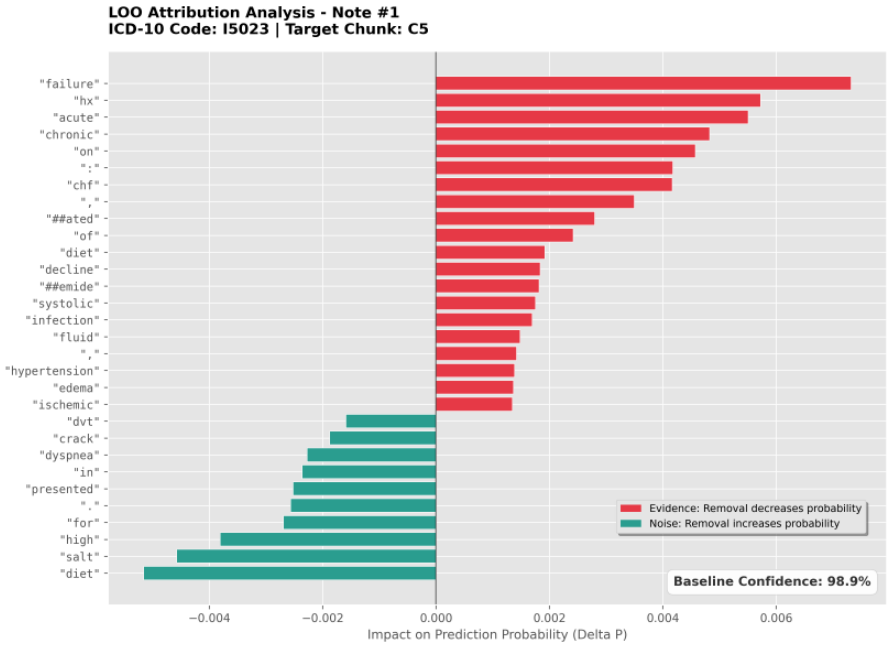


Figure 6.11: LOO analysis for code I5023 (heart failure) in chunk C5 of Note 2 (p=0.99). Evidence is dense and concentrated in one chunk. Contrast with the note from the heatmap (p=0.28), where identical tokens spread across chunks cause failure : the bottleneck is attention aggregation, not vocabulary.

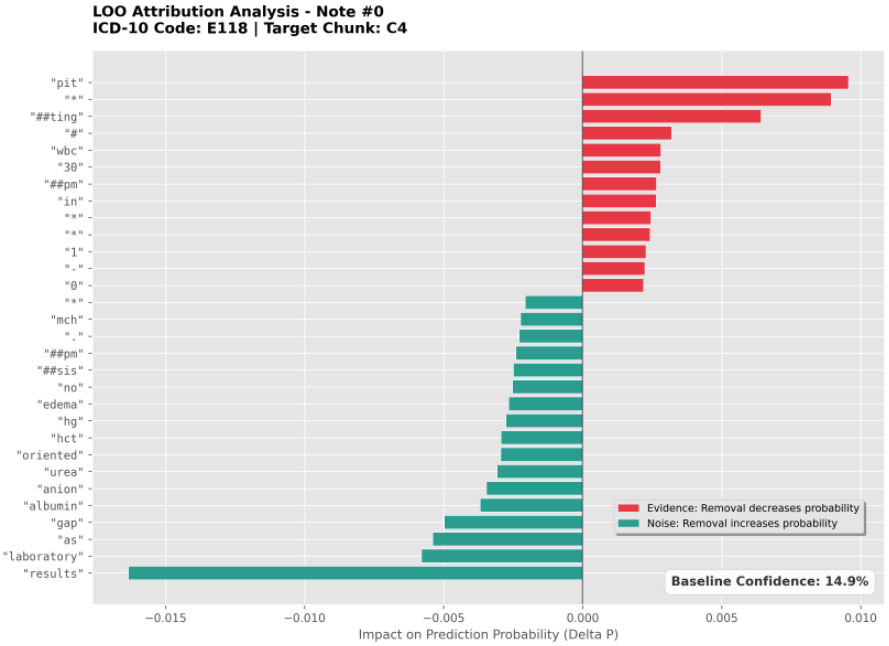


Figure 6.12: LOO analysis for code E118 (Type 2 Diabetes Mellitus with complications) in chunk C4 (most attended chunk, weight: 0.33). The chunk is dominated by irrelevant lab tokens; "laboratory," "albumin," and "urea" act as strong noise, actively degrading the prediction

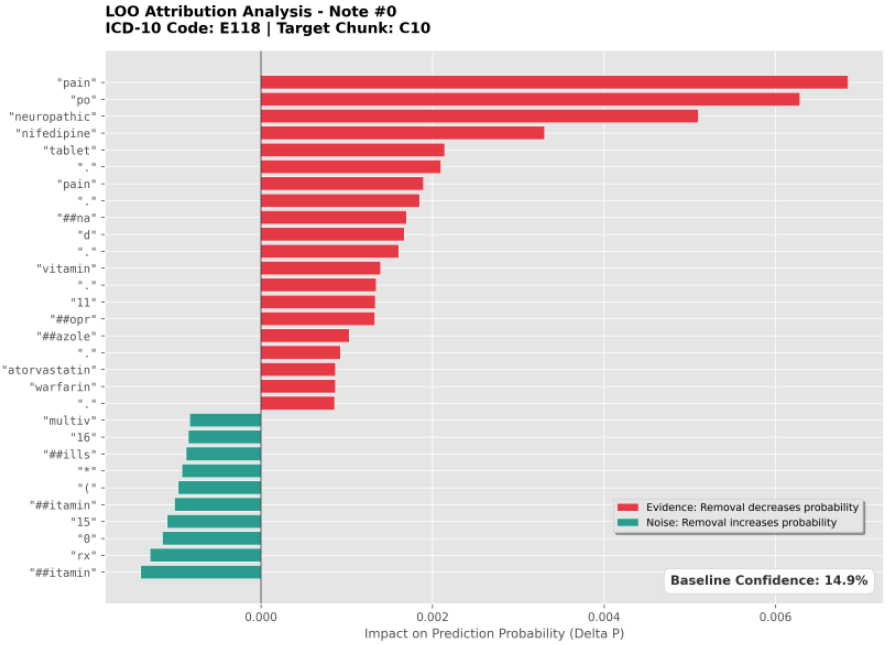


Figure 6.13: LOO analysis for code E118 (Type 2 Diabetes Mellitus with complications) in chunk C10. "neuropathic" and "nifedipine" emerge as weak useful signals, but the overall contribution remains too small to cross the decision threshold

Key takeaways from the interpretability analysis. The attention heatmap and LOO analysis together reveal three failure modes:

- **Distributed evidence** (I5023): relevant tokens exist but are spread across chunks : no single chunk accumulates enough signal to trigger a confident prediction.
- **Attention misallocation** (E118, chunk C4): the most attended chunk contains irrelevant tokens that actively degrade the prediction, while the right signals are buried in low attention chunks.

- **Chunk-level interference** (K259): unrelated diagnoses co-occurring in the same chunk introduce noise that competes with the target signal.

Conversely, the model is highly reliable when a single explicit token dominates a high attention chunk. These findings suggest that future improvements should focus on aggregation rather than token-level representation.

Chapter 7

Integration into OpenMRS

The natural next step of the thesis is the translation of the models into a tool that can be effectively used by clinicians. It requires embedding the models within a real-world clinical information system, OpenMRS in this case as motivated in Section 7.1. Furthermore, the inference pipeline must be adapted to the constraints and characteristics of the deployment environment, and an interaction model must be designed to align with existing clinical workflows. The result is the ICDHelper module, openly available at `https://github.com/tinyhuman-png/openmrs-module-icdhelper` under the Mozilla Public License v2.0 supplemented by an additional Healthcare disclaimer, consistent with the standard licensing approach for OpenMRS modules. The decision to develop an open-source model was deliberate and important to us. It reflects the underlying principle that health informatics systems should be transparent, collaborative, and universally accessible. Moreover, open sourcing the project enabled highly valuable engagement with a passionate and diverse international community.

This chapter begins by introducing the relevant architectural properties of OpenMRS that motivated our design decisions. Section 7.1 also provides a concise overview of the OpenMRS data model, with the aim of clarifying the structural components the module integrates itself with. Section 7.2 then details the architecture and implementation of the proposed module, together with the tool integration into the clinical workflow. Subsequently, Section 7.3 addresses the testing and validation procedures. The chapter concludes in Section 7.4 with a discussion of the community feedback received.

7.1 OpenMRS as the target platform

As stated in Section 2.3, OpenMRS is an open-source electronic medical platform. It is designed for deployment in resource-limited settings and already adopted in over 8,100 facilities worldwide. Beyond its widespread adoption and active community, two key architectural properties make it a particularly suitable target for deploying an ICD coding tool: its concept dictionary and its module system.

7.1.1 Concept dictionary and CIEL

The concept dictionary is at the core of OpenMRS. It defines all the clinical data elements used throughout the system. Each concept represents a clinical entity such as a diagnosis, a symptom, a laboratory test, a medication, etc.
A very interesting feature of concept objects is their ability to be mapped to other concepts or to external standardised terminologies, such as SNOMED CT, ICD-10 and ICD-11, LOINC. Each concept mapping must specify the nature of the relationship between the concept and the other code or term. For instance, a concept can be exactly equivalent to the external term (`SAME-AS`), or broader than it (`BROADER-THAN`), or narrower (`NARROWER-THAN`), among other possible relationships.

The mapping mechanism is central for our ICDHelper module, as it provides the bridge between the ICD-10-CM codes and the internal OpenMRS data representation. When the AI models return a code suggestion, the module searches the local concept dictionary for concepts mapped to this code via the `ICD-10-CM` or `ICD-10-WHO` sources. If the resolution step succeeds, the module can link the code to a known OpenMRS concept, enabling structured persistence. The quality and

completeness of this resolution is directly dependent on the state of the local dictionary.

In practice, many OpenMRS deployments rely on the CIEL (Columbia International eHealth Laboratory) concept dictionary as primary concept source [76]. CIEL is maintained and regularly updated by the Department of Biomedical Informatics at Columbia University. It maps thousands of clinical concepts to their counterparts in ICD-10, ICD-11, SNOMED CT, and other international standards. It is distributed through the OpenConceptLab (OCL) platform, and the OCL module allows OpenMRS instances to subscribe to the dictionary and synchronise updates.[1]
We warmly recommend OpenMRS implementers to subscribe to CIEL before using ICDHelper module, as this dictionary provides many concepts that map to `ICD-10-WHO`. It is worth noting that the source for `ICD-10-CM` does not officially exist in CIEL, but administrators can define custom concept sources and mappings locally, so we included matches to this exact source for the deployments that have done so.

Three specific CIEL concepts are required for the correct operation of the module: concept `1284`, called "Problem list" or "Coded Diagnosis", concept `161602`, called "Diagnosis or problem, non-coded" or "Non-Coded Diagnosis", and concept `162169` called "Text of encounter note". These are verified at module startup and their absence is reported as an error in the server logs.

7.1.2 Modular architecture

The set of functionalities offered by OpenMRS is extended through a module system. Independent packages called modules can register new services, UI pages, REST endpoints, and dashboard components without requiring changes to the platform core. Developers distribute the modules as OMOD files, which are JAR archives with a specific internal structure that the OpenMRS runtime unpacks and loads dynamically at startup. This allowed us to deliver the coding assistant as a self-contained installable unit: the ICDHelper module.

At the root of this architecture lies the `config.xml` manifest file. This file strictly defines the module's metadata, its required dependencies, and the global properties required for its operation. The OpenMRS platform can gracefully prevent the module from starting if the host environment lacks the necessary dependencies.

In practice, a module integrates with the platform through several mechanisms. The module activator is a class that implements `BaseModuleActivator` and defines lifecycle hooks invoked when the module is started or stopped. Within the ICDHelper module, the `started()` method is responsible for reading the configured model directory from the global properties, initializing the AI models, and verifying that the required CIEL concepts are present in the local concept dictionary. This design ensures that all heavyweight initialisation happens once at startup, keeping subsequent prediction requests efficient.

Services exposed by a module are registered in a Spring application context configuration file, named
`moduleApplicationContext.xml`. This file wires the service implementation directly into OpenMRS service registry. Once registered, the service becomes accessible to any other module or controller via `Context.getService(ICDHelperService.class)`, following the same pattern used by all native OpenMRS services. Transaction management is handled transparently by Spring through a `TransactionProxyFactoryBean`, which wraps the service implementation and ensures that all database operations respect the platform's transaction boundaries.

For OpenMRS 2, the module user interface is built using the OpenMRS UI Framework, which follows a convention-over-configuration, component-based architecture. The UI is assembled using Pages, which define the overall layout and routing, and Fragments, which are reusable self-contained UI widgets. Both Pages and Fragments can be backed by Java controller classes that handle GET and POST requests and populate a data model, which is then passed to a Groovy Server Pages (GSP) template for rendering. While GSP is the standard engine for the UI Framework and the one used by our module, the platform also supports Open Web Apps (OWAs) which are client-side Single Page Applications typically built with React or Angular and interacting with the backend purely via the REST API, as well as legacy JSP pages for older administration interfaces.

[1] Rafal Korytkowski. Open Concept Lab. https://openmrs.atlassian.net/wiki/spaces/docs/pages/25474980/Open+Concept+Lab Accessed: 2026-05-08

To surface new functionalities within the platform's existing UI rather than only through the administration panel or a direct URI, modules must register extension points. These extension points are unfortunately not always thoroughly documented, so identifying the exact string identifier required to place a new button or widget often requires a tedious search through the source code.

7.1.3 OpenMRS data model

To understand how the ICDHelper module interacts with patient records, it is necessary to first briefly introduce the core entities of the OpenMRS data model. The system adopts a hierarchical data organisation aimed at reflecting real-world healthcare workflows (Figure 7.1).

The `Patient` is the central entity of the system: all clinical data is linked to a patient profile, directly or indirectly. When a patient interacts with the healthcare facility for a continuous period of time, this interval is called a `Visit`. Visits may include multi-day hospital admission or single clinic appointments. An `Encounter` represents a specific episode of care, capturing both the time at which the interaction occurred and the healthcare provider involved. Encounters typically happen during a visit, with a visit often aggregating multiple encounters. Finally, the `Observation (Obs)` is the atomic unit of clinical information, generally recorded during an encounter. Observations are represented as "question and answer" pairs: the question is always defined by a concept form the local dictionary, and the answer can take various data types (text, numeric, coded, datetime) depending on the question concept.

The ICDHelper module first retrieves the `Observations` within the current active `Visit` whose question corresponds to the Visit Note concept `162169`. Predicted diagnoses are then persisted as `Observations` attached to the encounter of the selected visit note: coded diagnoses as a coded answer under concept `1284`, and unresolved codes as free text under concept `161602`.

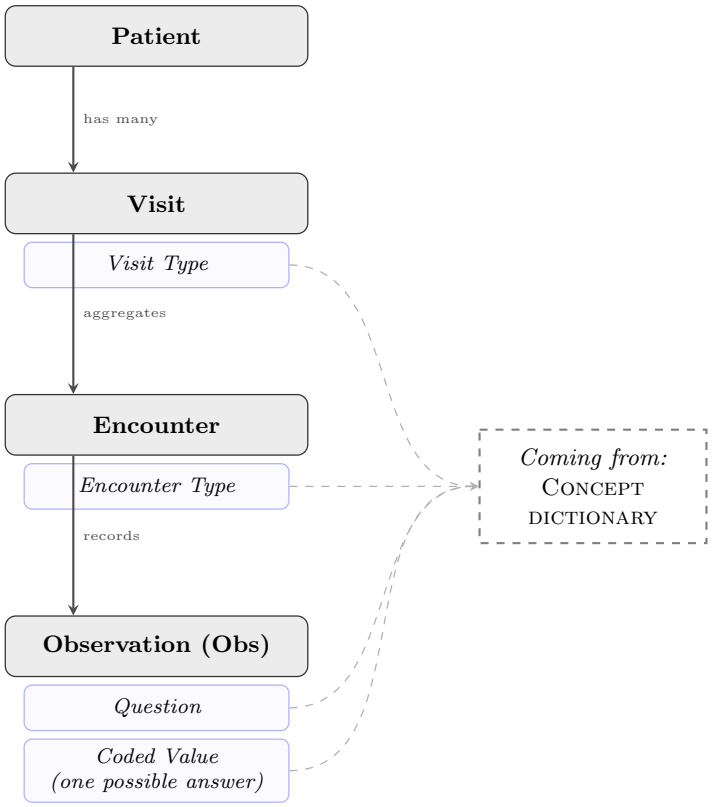


Figure 7.1: Simplified OpenMRS data model

7.2 Architecture and implementation of the proposed module

Our module follows the standard OpenMRS layered architecture for modules, with a separation between the prediction layer, the API layer, and the OMOD layer containing the user interface.

7.2.1 Prediction layer

For each analysis task, the best performing model identified during the evaluation presented in Chapter 6 was integrated into the module. Specifically, SapBERT for the short excerpts and the custom hierarchical BERT, modelD_hierarchy, for the complete note

A central design requirement of the module is that AI inference must run entirely offline, without any dependency on external services or network connectivity at prediction time. This constraint follows directly from the target deployment context. OpenMRS installations in low-resource clinical settings frequently operate on infrastructure with limited or unreliable internet access, and a coding assistant that degrades or fails when connectivity is unavailable would be of limited practical value.
For this purpose, we adopted **ONNX Runtime**[2]. ONNX (Open Neural Network Exchange) is an open standard format used for representing machine learning models. It was originally developed by Microsoft and Facebook, and is now maintained by the Linux Foundation AI & Data. It defines a common intermediate representation that allows models trained in major frameworks like PyTorch, TensorFlow, scikit-learn, etc. to be exported once and executed on any ONNX-compatible runtime, regardless of the original training environment. ONNX Runtime is the high-performance execution engine for this format. It is cross-platform, supports CPU and GPU execution, and provides native APIs for Java and Python, among others. It has become an industry standard for deployment of neural network models, particularly in contexts where Python dependencies are undesirable or unavailable.

In the ICD Helper module, the two prediction models were exported from PyTorch to ONNX format using the `torch.onnx.export()` method. Both have Opset version 17 and IR version 8. The Java API of ONNX Runtime then loads and executes these models directly within the JVM, eliminating external dependencies at inference time. Once the module is initialised, predictions are produced entirely within the OpenMRS server process, with no outbound communication. This satisfies the offline requirement completely: the tool functions in the same manner whether the server has internet access or not.
It is worth noting a practical build-time consequence of embedding the models within the module: the ONNX model files and vocabulary require a significant amount of memory during compilation. Users must therefore increase Maven's allocated heap before building. We recommend the configuration `MAVEN_OPTS="-Xmx3072m -Xms512m -XX:+UseG1GC"`.

`SapBertPredictor` and `HierarchicalPredictor` are implemented as standalone classes that encapsulate all interactions with ONNX Runtime. Both follow the same lifecycle: an `initialize()` method creates the ONNX session and loads the supporting data structures form disk at module startup, a `predict()` method performs inference on demand, and a `close()` method releases all resources at shutdown.
The predictors also share a common tokenizer: `BertWordPieceTokenizer`, a pure Java reimplementation of the HuggingFace `BertTokenizer` with `do_lower_case=True`. It was necessary to reimplement the tokenizer ourselves, as the official Java tokenizer binding provided by HuggingFace
(`ai.djl.huggingface:tokenizers`) causes a `LinkageError` when loaded within the OpenMRS module classloader. The reimplementation replicates the exact tokenisation pipeline: lowercasing, control character cleaning, punctuation splitting, and WordPiece subword segmentation. The code of `BertWordPieceTokenizer` was partially written with the help of generative AI, following EPL rules, and it was thoroughly tested, as described in Section 7.3.

The predictor implementations closely follow the architecture detailed in Section 5. The only important adaptation concerns the SapBERT predictor: a brute-force search replaced the FAISS lookup that did not integrate cleanly into OpenMRS. It was an acceptable substitution in terms of

[2]ONNX Runtime developers. ONNX Runtime. https://onnxruntime.ai/, 2021. Version: 1.14.0.

performance, given that the index only contains ~9,000 known ICD-10-CM code-description pairs. The embedding index is loaded once at startup, from a binary file to avoid unnecessarily large downloads for the user, and it is kept in memory for the lifetime of the module. Both predictors return the top-k closest codes along with their description and confidence score, with a default of $k = 20$ top-results.

The models in ONNX format, together with all supporting components such as the vocabulary and the embedding index, are available in the latest GitHub release. [3] We wrote a lightweight bash script `download_models` to automate their retrieval and placement within a designated directory at the project root. This approach simplifies deployment, requiring implementers only to transfer the directory to their OpenMRS instance.

7.2.2 API layer

The central component of the module is `ICDHelperService`, a service interface managed by Spring that exposes the core functionality of the module to other components of the platform. Its default implementation, `ICDHelperServiceImpl`, orchestrates the two AI models, formats the returned codes, handles concept resolution and manages persistence of selected diagnoses.

First, a small method `getVisitNotes()` retrieves all visit note observations associated with the current visit in the reverse chronological order. This functionality relies on the presence of concept `162169` ("Visit note"). The function leverages the Observation service from the OpenMRS API with the `getObservationsByPersonAndConcept`, thereby avoiding the need to query the database directly.

The primary method exposed by the service is `getPredictionFromNote()`. It accepts a clinical text and an analysis mode, delegates to the appropriate predictor, formats the returned codes and resolves each one against the concept dictionary. The analysis mode determines which predictor is called: SapBERT for highlighted excerpts, or the custom hierarchical BERT for complete notes. The dispatch is additionally safeguarded by a length check of the provided clinical text: an input of over 1,000 characters is automatically routed to the hierarchical BERT, even if the chosen analysis mode was 'selection' (the 50-token truncation of SapBERT would discard most of the clinical content).

Concept resolution is handled by `getConceptByICD10Code()`, which implements a two-pass lookup strategy. The first pass searches the `ICD-10-CM` source for a mapping to the complete code. Matches are prioritised by relationship type, in the order `SAME-AS` > `BROADER-THAN` > `NARROWER-THAN`, with ties broken deterministically by concept ID. This order reflects a deliberate data integrity choice: saving a concept that is broader than the predicted code is safer from an audit perspective than choosing one that is narrower.
If no match is found, a second pass searches the `ICD-10-WHO` source with a progressive truncation strategy. For example, a code such as `S82.001A` is tried first as-is, then as `S82.001`, then `S82.00`, and so on down to the three-character category code. When a `SAME-AS` match is found against a truncated code, the result is returned as `BROADER-THAN` to preserve clinical accuracy, as the matched concept is inherently broader than the original untruncated code. During the second pass, only `SAME-AS` matches are accepted, as allowing inexact matches to a truncated code would only add imprecision.

Each prediction is encapsulated in an `ICDResult` object, which aggregates the original ICD-10-CM code, its description, the confidence score given by the model, and if the resolution succeeded, the matched OpenMRS concept with the mapping type.

When a user selects one or more predictions to be saved, the `saveSelectionToEncounter()` method records each item either as a coded or as a non-coded observation, and persists it on the encounter parent to the visit note. This design ensures the predicted codes remain contextually bound to the specific note from which they were generated.
The predictions for which a matching OpenMRS concept was found, called coded diagnoses, are saved as observations under concept `1284` "Coded Diagnosis" with the matched concept as the coded answer. When the mapping is not exact (`BROADER-THAN` or `NARROWER-THAN`), the original

[3] https://github.com/tinyhuman-png/openmrs-module-icdhelper/releases/latest/

ICD-10-CM code is stored along in the observation's comment field to preserve clinical context and traceability.
Non-coded diagnoses, for which no concept mapping exists in the local dictionary, are saved as free-text observations under concept `161602` ("Non-Coded Diagnosis"), formatted as `<code>: <description>`.

Both save operations are idempotent: before creating a new observation, the method verifies whether an identical non-voided observation already exists under the same encounter. If one is found, it is returned directly without creating a duplicate, making the operation safe to repeat if a clinician saves the same suggestion twice.

7.2.3 OMOD layer

The OMOD layer contains all components specific to the web module: the page controller, the GSP template, the Spring web application context, and the app configuration registering the module's extension point. The separation between the API and OMOD Maven submodules ensures that the core service logic has no dependency on the web layer. It simplifies unit testing and would facilitate the future development of a different frontend framework.

The user interface is exposed as a single GSP page, accessible from the patient dashboard via the `patientDashboard.visitActions` extension point. The page controller, `IcdHelperPageController`, handles two POST actions: `predict`, which invokes the prediction service and populates the results table, and `save`, which persists the selected results.

As shown in Fig 7.2, the visit note picker lists all visit note observations for the current visit of the current patient in reverse chronological order. Once a note is selected, its full text is displayed in a scrollable panel below. It remains locked in place while the clinician interacts with the tool. The clinician can choose the analysis mode, noting that `selection` mode only works if some text is highlighted. Results are presented in a table showing the ICD-10-CM codes, their description, the matched concept names if any, along with the mapping types, and the confidence scores given by the model. Each row has a checkbox and is entirely clickable, allowing easy selection before saving.

Integration in the clinical workflow From the clinician's point of view, the ICDHelper module appears as an additional action under the current visit section of the patient dashboard, available whenever the patient has an active visit. This placement, as displayed in Figure 7.3, was deliberate: we embedded the coding assistant within the existing clinical workflow rather than as a separate administrative tool. We assumed that the clinician would likely already be on the patient dashboard when they want to assign diagnosis codes.
The interaction follows a linear flow. The clinician opens the tool, selects the visit note they want to analyse, chooses an analysis mode, reviews the AI suggestions, selects the relevant propositions, and saves them to the encounter. The entire interaction takes place within a single page, without navigating away from the patient context. Saved diagnoses appear immediately as standard OpenMRS observations attached to the same encounter as the source visit note. This makes them accessible to any module or report that reads observation data, therefore no special handling is needed downstream.

7.3 Testing and validation

Testing was divided between a unit test suite and an end-to-end test suite. The unit tests cover the service logic in isolation, without a running OpenMRS instance. The end-to-end tests, on the other hand, validate the complete user-facing workflow, including the user interface and the database persistence.

7.3.1 Unit tests

The unit test suite is built with JUnit and the Mockito framework. It ensures that the business logic is correct without requiring a fully booted OpenMRS environment, enabling fast and deterministic feedback during development. The tests cover all layers of the module.

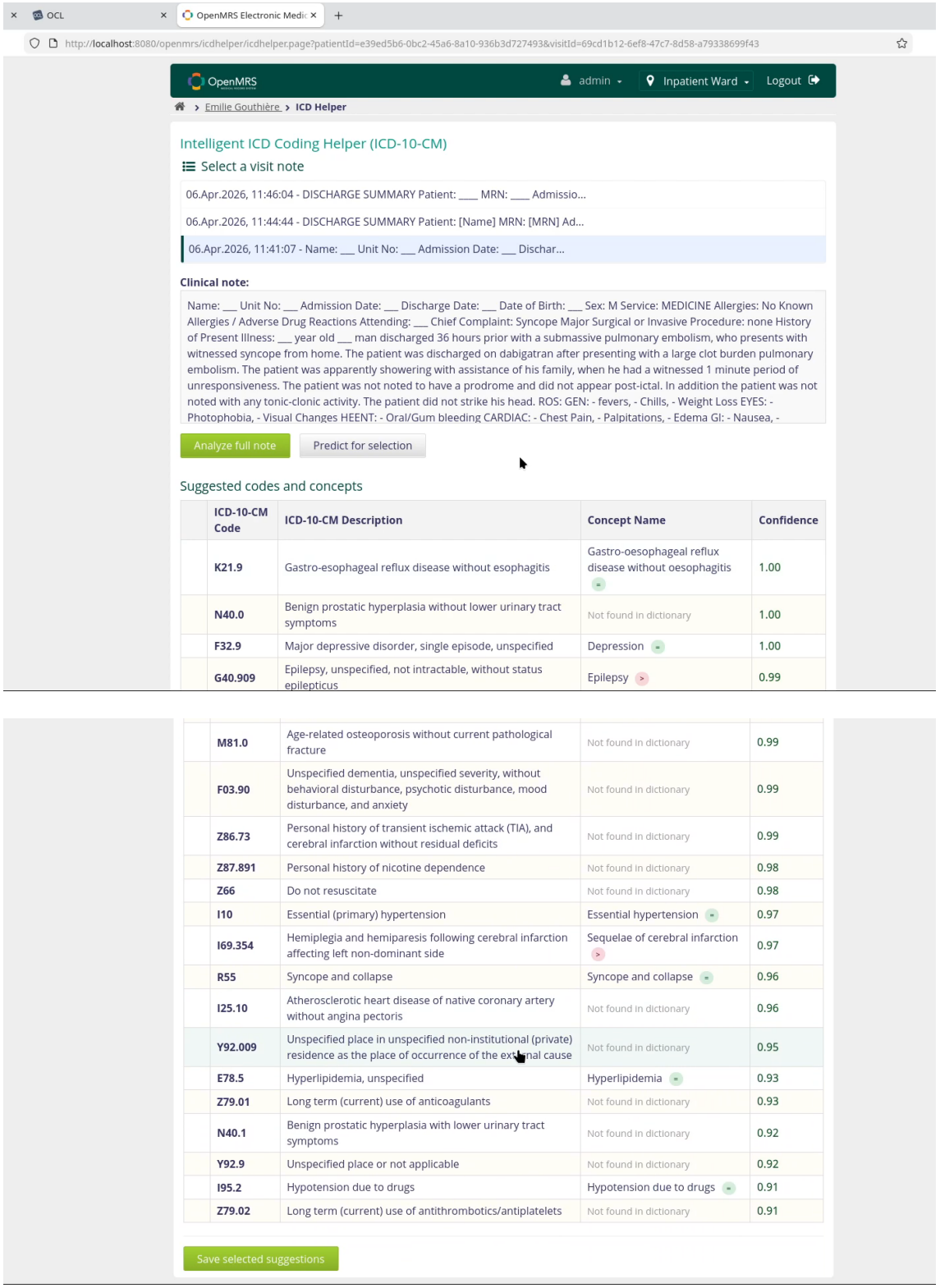


Figure 7.2: Example use of the tool page

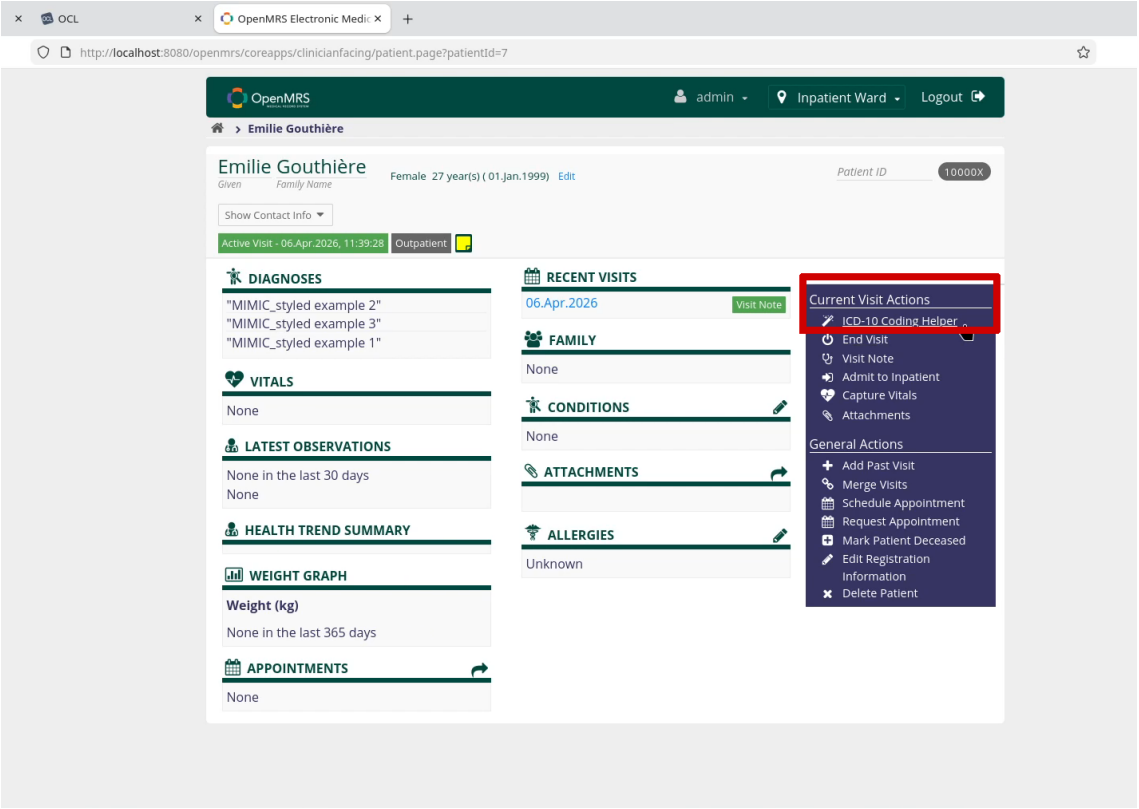


Figure 7.3: Tool placement in the patient dashboard

Prediction layer. Reimplementing the HuggingFace tokeniser in Java introduced a risk of tokenisation mismatch that would silently degrade the accuracy of the predictions. The `BertWordPieceTokenizerTests` class mitigates this risk by testing the tokeniser on a controlled vocabulary. Tests range from verifying the correct handling of edge-case characters, to asserting that the tokenisation of example clinical notes produces token ID sequences identical to those generated by the original Python HuggingFace `BertTokenizer`.
In addition, a separate set of Python tests –not included in the module– was developed to verify structural and behavioral consistency between the evaluated models and their exports in ONNX format. The tests check that example predictions are identical before and after the export. The Netron visualisation web tool was helpful during this process to visualise the exported ONNX models structure and verify input and output sizes names.[4]

API layer. The core service logic is tested in `ICDHelperServiceTest`. The tests cover all methods of `ICDHelperServiceImpl`: the routing of inputs to the correct predictor based on analysis mode and text length, the full concept resolution strategy including the two-pass lookup and the priority ordering, the correct persistence of coded and non-coded observations, the retrieval of visit notes in reverse chronological order, and all error handling. The OpenMRS concept dictionary is simulated using an XML dataset in memory, and the AI predictors are replaced with mock implementations, keeping the tests focused on the service logic alone.

OMOD layer. The unit tests for the web layer validate only the page controller; the GSP template and full UI behaviour are covered by the end-to-end suite. `ICDHelperPageControllerTest` mocks both the underlying `ICDHelperService` and the OpenMRS context objects such as `Patient` and `Visit`, and it simulates various GET and POST interactions. The tests verify that the controller correctly dispatches recognised actions, handles unrecognised actions gracefully, and populates the `PageModel` with the expected attributes and error messages before handing it off to the GSP rendering engine.

Table 7.1 summarises the coverage achieved by the unit test suite, as measured by JaCoCo. The results reflect the deliberate scope of the unit test suite. The two core components under ac-

[4]Lutz Roeder. Netron: Visualizer for neural network, deep learning and machine learning models. https://netron.app/

tive test (`ICDHelperServiceImpl` and `IcdhelperPageController`) achieve strong line and branch coverage of 84% and 91% respectively. The tokeniser reaches near-complete line coverage (96%), consistent with its role as a reimplementation that required precise behavioural validation against the Python reference.

The low figures for `HierarchicalPredictor` and `SapBertPredictor` (around 1% line coverage) are expected and do not represent a gap in the strategy. Their inference behaviour is validated through the Python export tests described above, and end-to-end through the Playwright suite running against a fully initialised module. `ICDHelperActivator` is similarly excluded from unit tests as it depends on a live OpenMRS context.

The overall 57% line coverage for the API module can be somewhat misleading as summary metric. This figure is artificially dragged down by the inclusion of classes that are intentionally excluded from unit testing. When considering only the classes under active testing, coverage consistently exceeds 84%.

Table 7.1: Unit test coverage by module layer, as reported by JaCoCo

Layer	**Class**	**Class, %**	**Method, %**	**Line, %**	**Branch, %**
API module					
Service impl	`ICDHelperServiceImpl`	75	76	84	84
Tokenizer	`BertWordPieceTokenizer`	100	100	96	86
Predictor	`HierarchicalPredictor`	33	9	1	0
Predictor	`SapBertPredictor`	50	8	1	0
DTO	`ICDSearchResult`	100	91	90	100
OMOD module					
Controller	`IcdhelperPageController`	100	100	91	85
Overall (API)		61	57	57	64

7.3.2 End-to-end tests

An automated end-to-end test suite was built with Playwright and Typescript, running against a live OpenMRS instance deployed via Docker Compose with the module pre-loaded. To ensure test reliability and prevent state collisions between test runs, the suite adopts a strongly isolated testing architecture.

Each test generates its own unique `Patient`, `Visit`, `Encounter`, and associated `Obs` records via the OpenMRS REST API at setup time, ensuring complete independence from any pre-existing database content. Once a test completes, the fixture automatically purges the generated data, leaving the environment in a clean state for the next run. Authentication is handled once by a global `setup.ts` script, which performs the OpenMRS login and stores the session cookies to a `storageState` file. All subsequent tests load this state directly, bypassing the login screen and improving execution time.

The tests cover typical workflows a clinician would follow when using the tool, such as submitting a full note for prediction, highlighting an excerpt and submitting for selection prediction, saving multiple suggested codes. They additionally verify that the selected note stays locked after a prediction, that error messages are correctly displayed, and that saving is idempotent.

7.3.3 CI pipeline

The continuous integration pipeline is implemented with GitHub Actions and is triggered on every pull request and every push to the main branch. It first builds the module and runs the unit test suite. If this stage passes, OpenMRS is started in a Docker container with the module loaded and the model files available. Once the server is running and ready, the end-to-end test suite is run against it, limited to the Chromium browser so the execution time does not explode. The two-stage structure ensures that the expensive environment setup is only attempted when the core logic is verified.

The module was submitted to the OpenMRS community in a state were the full CI pipeline passes. The repository is publicly available at https://github.com/tinyhuman-png/openmrs-module-icdhelper.

7.4 Community feedbacks

We published a module release announcement on the OpenMRS Talk forum. [5]. We received feedback from several community members, including developers and implementers with direct experience in deploying OpenMRS. We categorised the reactions into four key themes: the offline design, the choice of coding system, the persistence model for coded diagnoses, and the user interface.

Offline design The offline inference decision was consistently highlighted as the most significant strength of the module. Respondents, including Ellen Ball from Partners In Health (whose sites span Haiti, Sierra Leone, and other low-connectivity environments) confirmed that this property directly addresses a real deployment constraint. A user raised the question of how model updates would be distributed in an offline setting and whether a synchronisation mechanism was planned. In the current implementation, updates require manually replacing the model files in the server directory and restarting the module. A future release could introduce a startup check that notifies administrators when updated model assets are available, following a pattern similar to how the OpenConceptLab module handles concept dictionary updates.

ICD-10-CM versus ICD-10-WHO The most recurring technical feedback concerned the choice of ICD-10-CM as the target coding system. Several respondents noted that ICD-10-CM is specific to the U.S. and not widely used within the OpenMRS community, where ICD-10-WHO is standard. Moreover, CIEL does not include ICD-10-CM mappings directly. Dr. Andrew Kanter, director of the Columbia International eHealth Lab (CIEL), further added that, for OpenMRS deployments using OCL-managed dictionaries, the recommended approach is to map clinical observations to interface concepts that carry mappings to both ICD-10 and SNOMED CT, rather than mapping directly to ICD codes as is currently done in this implementation.

The choice of ICD-10-CM was driven by data availability: MIMIC-IV, used to train and evaluate the hierarchical model, provides clinical notes mapped to ICD-10-CM codes, and there are no comparable large-scale, openly accessible dataset mapped to ICD-10-WHO codes to our knowledge. Partial adaptation is feasible, though. For the selection mode, switching from ICD-10-CM to ICD-10-WHO would only require rebuilding the pre-computed SapBERT embedding index against WHO codes and descriptions, without retraining the model. For the full note mode on the other hand, a migration would require a new training dataset, which remains an open challenge. Andrew Kanter pointed to IMOHealth as a potential source of data or research partnership, and expressed willingness to help identify ICD-10-WHO labelled datasets. These exchanges are left as a starting point for future collaboration, and the question of dataset availability is discussed further in Section 8.3.

Observation persistence model A community member noted that coded ICD-10 diagnoses, particularly those carrying billing-specific modifiers present in ICD-10-CM, should be stored under the OpenMRS `Visit Diagnosis` construct rather than as generic coded observations. This is a valid point that was not addressed in the current implementation, where diagnoses are saved as observations under concept `1284`. Aligning with the Visit Diagnosis model is identified as a priority for a future release.

User interface A respondent suggested adding explicit checkboxes to the results table to make the row selection mechanism more visible. This feedback was incorporated: the final implementation includes checkboxes on each row while keeping the entire row clickable, so that users are not required to target the checkbox precisely to toggle their selection.

[5] https://talk.openmrs.org/t/new-icdhelper-module-released/49292 and https://talk.openmrs.org/t/new-icdhelper-module-release/49293

Usability survey To complement the community feedback, we designed a usability survey which targets potential end users of the developed coding tool. The survey combined the System Usability Scale (SUS) questionnaire [77] with questions specific to the ICD coding context, covering the perceived usefulness of predictions, trust in AI-generated suggestions, and overall integration into the clinical workflow. We wrote the questionnaire with LimeSurvey, a free and open-source tool used by many prominent organisations in the FOSS community such as Ubuntu, OpenOffice, and GNOME. It is accessible at https://egouthiere.limesurvey.net/796197?lang=en&newtest=Y.

Unfortunately, to this day, no complete responses have been collected. The absence of survey data does not invalidate the module, but it does mean that formal usability validation remains inconclusive. Collecting a representative sample of answers from clinicians and coders in real deployment contexts is an important direction for future work. It would both validate the current design and guide the development of functionalities most relevant to end users.

7.4.1 OpenMrs Virtual Showcase

We enrolled in the OpenMRS Virtual Community Showcase, which took place on April 29^{th} and 30^{th}. It is a recurring online gathering aimed at providing insights into the different ongoing projects across the OpenMRS community. We presented the ICDHelper module in a 5-minute lightning talk, placing the ICD coding context, concisely explaining the proposed solution and demonstrating the tool in use. It was a highly valuable exercise for us to present our work to a live audience and receive direct feedback. The community comments were largely similar to those gathered from the publication message on the OpenMRS Talk forum, which reinforced their relevance. Attending the broader showcase also provided useful perspective on how OpenMRS is being deployed and extended in practice today, and highlighted how frequently artificial intelligence is being integrated into global health informatics projects, a trend that contextualises and motivates the present work.

Chapter 8

Future work

8.1 Future works on group of words to ICD codes task

The results of this first task open several directions for future work. Some of these can be pursued directly from the current experimental setup, while others depend on the availability of a larger, real-world evaluation dataset.

Real-world evaluation on clinical note fragments The synthetic evaluation set used in this task was designed to approximate the lexical variability of real clinical expressions, but it remains derived from official ICD descriptions by language models. Whether the performance gaps observed between models, especially the large advantage of SapBERT over unaligned neural models and lexical baselines on hard examples, transfer to naturalistic clinical text remains an open question.

Such an evaluation set could be constructed from MIMIC-IV, but would require human annotation since the ICD codes are associated with discharge notes at the document level. The most rigorous method would select a set of clinical notes, then ask annotators to identify short diagnostic expressions (a sentence, a clause, or a noun phrase) and to pair each one with the ICD code it most directly refers to. This is closer to named entity recognition annotation than to full document coding, and is already less demanding than annotating which parts of a note support each of the 14 codes assigned on average per admission. It however requires non-negligible clinical coding expertise. Some identification time could be gained by exploiting the strict structure of MIMIC-IV discharge note. In particular, the *Discharge Diagnosis* section of the note contains short, semi-structured diagnosis phrases that could be automatically extracted. Alternatively, we could leverage existing named entity recognition (NER) tools, such as the open source GLiNER2 [78], which was tested and identified as a promising tool for this step.

Alignment training on ClinicalModernBERT The most directly motivated future direction is to apply SapBERT-style self-alignment pretraining to ClinicalModernBERT. The results of the first task of this thesis establish that the self-alignment objective is the decisive factor for retrieval performance, not biomedical pretraining per se. ClinicalModernBERT already performs better than PubMedBERT, suggesting that its architectural improvements provide a useful initialisation, but one that is insufficient on its own. Training with a multi-similarity loss on UMLS pairs, as done for the original SapBERT but starting from the ClinicalModernBERT encoder rather than PubMedBERT, could yield a model that simultaneously benefits from clinical domain pretraining, modern architecture (extended context window, updated attention mechanisms), and synonym-aware metric learning.
Our reimplementation of SapBERT already provides the training infrastructure for this experiment, the only modification needed is to replace the base encoder. This direction requires no additional data beyond UMLS and can be evaluated on the existing synthetic dataset.

Addressing the Injury Poisoning attractor in lexical models The Dice-Sørensen coefficient's systematic over-prediction of Chapter XIX was identified in the results. A simple TF-IDF-style reweighting applied to the bigram overlap could potentially correct this bias without abandoning the lexical approach entirely. Specifically, bigrams that appear frequently across many

chapters would receive lower weights than more specialised terms, reducing the overlap that drives the attractor effect.

Exploring other promising models We limited ourselves to the evaluation of four lexical baseline methods and three distinct neural models. There exists however some more recent biomedical entity linking models, which were not explore in this work but may be well suited for this task. The most promising next baselines are KRISSBERT [79] and BioSyn [80], which aim to improve robustness to paraphrases or unseen clinical wording. Another realistic extension is to replace the pure neighbour matching with a retrieve-then-rerank design [81].

Synthetic dataset as a fine-tuning signal A more targeted use of the synthetic dataset generated in this thesis is as a fine-tuning corpus for alignment training, given an additional held-out test set. Each generated description variant, paired with its corresponding official ICD description, constitutes a weak positive pair: both refer to the same code, and should therefore be close to each other in the embedding space. Fine-tuning SapBERT on these pairs, followed by evaluation on a real-world test set of clinical expressions, would assess whether training on LLM-generated paraphrases improves the retrieval performance compared to alignment on UMLS pairs alone. This approach, however, requires the realistic dataset mentioned above as a prerequisite. If both the training and test sets were generated by the same LLMs, any observed performance gain might reflect adaptation to the generation style rather than genuine improvement on clinical language.

8.2 Future works on our best model : modelD_hierarchy

`modelD_hierarchy` was designed with deployment constraints in mind, favouring architectural simplicity over maximising raw performance. Nevertheless, several directions for improvement remain, which could extend its capabilities by addressing the specific failure modes identified in the results and interpretability analyses.

Retrieve and Re-rank pipeline One promising direction to improve recall@20 would be to adopt a retrieve-and-re-rank strategy [43], with a two-stage pipeline for automatic medical coding. In their approach, a retrieval stage first reduces the full code space to a small candidate set using auxiliary EHR knowledge (DRG codes, CPT codes, prescribed medications) combined with BM25 lexical matching. A re-ranking stage then scores these candidates using Clinical Longformer for document encoding, a Graphormer to model inter-code co-occurrence relationships, and contrastive learning to align document representations with their true ICD codes in the embedding space.
We explored a lightweight variant of this idea. Our model's top-50 or top-100 predictions were used to define the candidate set. GLiNER [82] was then applied. It is a general-purpose named entity recognition model. It identifies text spans corresponding to arbitrary entity types from short natural language prompts. The objective was to extract clinical entities from the discharge note using entity types such as disease, symptom, medication, and procedure. Subsequently, a BGE cross-encoder reranker was employed to score each candidate code by pairwise comparison between the extracted entities and the official ICD code descriptions. The final score combined both signals according to the following formula:: score $= (1 - \alpha) \times$ (BERT score) $+ \alpha \times$ (Reranker score).
Despite being conceptually well founded, this approach did not yield consistent improvements over our modelD_hierarchy recall@20. Two primary limitations were identified. First, the entity extraction performed by GLINER introduced noise, missing or mislabelling clinical concepts that were critical for rare codes. Second, the BGE reranker was not fine tuned on ICD code descriptions, limiting its ability to discriminate effectively between codes with similar clinical meaning. A fully trained retrieve and re-rank pipeline, with domain adapted components and end-to-end optimisation [43], remains a strong avenue for future work.

Focal loss for class imbalance Our current model uses a weighted binary cross-entropy loss with a single scalar `pos_weight` applied uniformly to all positive classes at each granularity level (5.0 for chapters, 15.0 for 3-char codes, and 30.0 for full codes). While this global upweighting helps to emphasise positive examples, it does not differentiate between easy and hard predictions. This limitation has been identified in the literature: easily classified negatives dominate the total loss and overwhelm the gradient signal from rare, hard examples [83]. In our setting, around 14 codes are present per note out of 9 685, meaning thousands of confidently predicted absent codes

collectively suppress learning on true positives.
Focal Loss addresses this by introducing a modulating factor $(1 - p_t)^\gamma$ to the standard cross-entropy: $\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$ When p_t is large (easy example), the factor goes to 0 and the loss is down-weighted. When p_t is small (hard example), the factor stays near 1 and the loss is preserved. With $\gamma = 2$, an example classified with p_t = 0.9 incurs a loss 100× lower than standard BCE, while a hard example at p_t = 0.1 retains a scaling factor of 0.81. Unlike `pos_weight`, this modulation is dynamic and naturally redirects learning capacity toward rare, misclassified codes without requiring any architectural changes.

Extension to French clinical notes As part of this thesis, the proposed model was integrated as a module within OpenMRS, an open-source electronic medical record system widely deployed in low- and middle income countries. Several of these countries, including Senegal, Côte d'Ivoire, and the Democratic Republic of Congo use French as their official language in clinical practice [1]. The current model is therefore unable to serve these healthcare systems, as it was trained exclusively on English clinical notes. Extending it to French, using biomedical language models such as CamemBERT-bio [84] or DrBERT [85], would be a natural and high-impact next step. The main challenge remains the lack of an annotated French clinical dataset equivalent to MIMIC, which would need to be constructed.

Multimodal integration Combining free-text discharge notes with structured EHR data including prescriptions, laboratory results, and vital signs has been shown to significantly improve ICD coding performance [86]. This direction is particularly relevant to our model's failure cases. As shown in our interpretability analysis, critical evidence such as an insulin regimen is often present in the medication list but ignored by the model, as the relevant chunk receives insufficient attention weight. Encoding structured data such as prescriptions as additional binary features alongside the textual chunks would directly address this limitation at low architectural cost.

8.3 ICDHelper module perspectives

While the current implementation of the ICDHelper module successfully demonstrates the feasibility of embedding offline neural models within OpenMRS, technical evaluation and feedbacks highlighted several constraints and opportunities for improvement. This section outlines the structural limitations of the current module and establishes a roadmap for future development.

Prediction of ICD-10-WHO codes and extension to ICD-11 As discussed in Section 7.4, ICD-10-CM is not widely used within the OpenMRS community: ICD-10-WHO is the standard, and CIEL does not include ICD-10-CM mappings. A more OpenMRS-native approach would be to predict against CIEL interface concepts directly, then derive ICD-10 codes from their curated mappings, as recommended by the OCL team. This approach would make the resolution step more reliable and less dependent on truncation heuristics, and could even permit an extension of the prediction tool to other standardised terminologies, as long as the mapping exists in CIEL. Especially, following the feedback from a community member, this could help the transition to ICD-11 that is currently ongoing without having to wait for a MIMIC dataset featuring the new revision.
For the selection mode, the adaptation is achievable without any retraining: the pre-computed embedding index could be rebuilt against ICD-10-WHO code descriptions or CIEL concept names and synonyms instead of ICD-10-CM. For the full note mode, however, a migration would require a new training dataset providing clinical notes mapped to ICD-10-WHO codes or CIEL concepts. No openly accessible dataset comparable to MIMIC-IV currently exists for this purpose, and identifying such a dataset remains the primary obstacle.

Dependency on clinical note structure The full note model was trained on MIMIC-IV discharge summaries, which follow a structured format characteristic of academic medical centres in the United States. In practice, clinical notes vary considerably in structure, length, and style across countries and institutions. Notes produced within the deployment context of OpenMRS may be less structured, or written in a problem-oriented way rather than in a narrative format, which degrades the performance of the hierarchical model.
Addressing this limitation would require either fine-tuning the model on a more diverse corpus of

[1] OpenMRS. Around the world. https://openmrs.org/around-the-world/, 2026. Accessed: 2026-05-15.

clinical notes, or collecting and annotating a dataset from actual OpenMRS implementations. The latter would be particularly valuable, as it grounds the evaluation in the real deployment context. The usability survey, which was initiated to collect clinician feedback on prediction quality in a live deployment, would have been a first step toward identifying where the model falls short in practice, if it had reached its audience better.

Visit Diagnosis persistence model In the current implementation, selected diagnoses are saved as standard OpenMRS observations under concept `1284` ("Coded Diagnosis") and concept `161602` ("Non-Coded Diagnosis"). As raised during community feedback, coded ICD-10 diagnoses should more properly be stored using the `Visit Diagnosis` construct provided by the `emrapi` module. This is the standardised persistence model for diagnoses in OpenMRS, which is recognised by downstream modules such as the billing module. The `emrapi` module is already a dependency of our module through `coreapps`, so this migration would not introduce new dependencies. It would nonetheless require refactoring the persistence logic in `ICDHelperServiceImpl` and updating the UI to reflect the distinction between primary and secondary diagnoses, which the Visit Diagnosis model supports explicitly.

Integration into the OpenMRS billing ecosystem A natural downstream application of structured ICD-10 diagnosis persistence is integration with the OpenMRS billing module. If the ICDHelper module stored diagnoses using the Visit Diagnosis model described above, they could be fed directly into the billing pipeline without manual re-entry. This integration would significantly increase the practical utility of the module in settings where ICD coding is driven by billing requirements rather than purely clinical documentation needs.

Explainability in the user interface The current interface presents prediction results as a ranked list of codes with confidence scores. However, it provides no information about why a particular code was suggested. For a clinical coding assistant, explainability is essential both for building trust in the tool and for enabling clinicians to make informed decisions when selecting among the suggested codes. The attention weights of the hierarchical model computed at prediction time, already explored in Section 6.2.7, can be used for this purpose. We could surface these weights directly in the user interface, for instance by highlighting with different degrees of opacity the text chunks that influenced the most each prediction.

Automated model update mechanism In the current design, updating the AI models is a manual step that could easily be missed in production deployments. A future release could introduce a startup check that compares the locally installed models version against the latest available release, and notifies the administrator when an update is available, following a pattern similar to how the OpenConceptLab module handles concept dictionary updates. This would be particularly interesting for the SapBERT embedding index, which can be updated whenever the ICD-10-CM code list changes.

Platform evolution The target version for the module implementation is the OpenMRS Reference Application 2.12.0, which bundles the core platform with a set of standard modules providing patient registration, visit management, encounter recording, and a clinical dashboard. Although OpenMRS 3 (O3) represents the current development direction of the platform, it was not selected as the target version for several reasons. First, version 2 is still deployed in production environments. Second, a stable Docker image of v2.12.0 was readily available for development and testing. Third, the GSP/UI Framework allowed us to develop the full module in a single Java-based stack without introducing a separate frontend build pipeline. Migration to O3, which would primarily involve rewriting the frontend layer as a React-based microfrontend, is left as a natural direction for future work. Given that O3 is increasingly adopted, this transition represents a priority fro ensuring the long-term relevance of the module within the OpenMRS ecosystem.

Chapter 9

Conclusion

Clinical coding translates the raw complexity of patient encounters into standardised diagnostic codes. It constitutes a critical step for billing, epidemiological surveillance, and interoperability, yet it remains a predominantly manual process in many healthcare systems worldwide. This process is time-consuming, costly, and susceptible to inconsistency. This thesis aims to bridge the gap between the research literature on automated ICD coding and to develop a tool capable of supporting clinicians in real-world practice. The goal is not to replace the medical coder, but rather to assist them by surfacing relevant code candidates and reducing the burden of manual search.

This objective was addressed along two complementary axes. The first concerns the machine learning models themselves. We built and evaluated a complete pipeline for the short excerpt coding task, comparing four lexical baselines against four neural models on a synthetic evaluation dataset of 2,567 examples which was both generated and validated as part of this work. The results established clearly that self-alignment pretraining constitutes the decisive factor, with original SapBERT achieving a MRR of 0.477 compared to 0.373 for the best lexical baseline and 0.181 for unaligned PubMedBERT.
For the full note coding task, we developed a progressive sequence of models on MIMIC-IV, culminating in modelD_hierarchy. It is a single multi-task PubMedBERT encoder that jointly predicts ICD-10-CM codes at three granularity levels simultaneously (chapter, 3-character, and full code). It uses a residual bridge mechanism that lets frequent parent codes regularise the prediction of rare children. Our hierarchical model produced the largest gains precisely where they matter most. `modelD_hierarchy` achieves macro F1 = 0.068 (+74% vs. TF-IDF baseline) and recall@50 = 0.700, meaning the model recovers 70% of all true codes within its top 50 predictions. The interpretability analysis confirmed that the model understands the right clinical vocabulary. The observed failures are not primarily attributable to the recognition of relevant terms, but instead from where the model looks in the note and the confidence with which it scores its findings.

The second axis was deployment. Both models were integrated into ICDHelper, an OpenMRS module that runs offline within the JVM using ONNX Runtime, with no external API call at inference time. The module resolves AI-predicted codes against the local CIEL dictionary, saves selected diagnoses as structured observations on the patient's encounter, and integrates seamlessly into the existing clinical dashboard without disrupting clinicians' workflow. ICDHelper has been released under the Mozilla Public License, submitted to the OpenMRS community, and already reviewed by implementers, including Partners in Health and the director of the CIEL dictionary.

Three open questions define the most important directions for future work. The first is the coding system gap. The models were trained on ICD-10-CM because MIMIC-IV is the only large-scale openly accessible dataset with discharge notes mapped to diagnostic codes, but ICD-10-WHO is the standard within the OpenMRS community. For the selection mode, adapting to ICD-10-WHO requires only rebuilding the embedding index; for the full note mode however, it requires a new training dataset that does not yet exist openly.
The second direction is clinical validation. While the module has been built and released, its predictive performance and integration within the OpenMRS platform remain to be evaluated in a real deployment context. The usability survey we designed did not reach its audience in time. These two gaps represent clear and transparent boundaries of the present work, delineating where our contributions conclude and where subsequent efforts must begin.
A third direction concerns model performance itself. Despite the gains brought by hierarchical

training, the full note coding task remains far from solved. The model recovers 70% of true codes within its top 50 predictions. While this is promising for a coding assistance workflow, reviewing 50 candidates per note is a substantial cognitive burden on clinicians. The interpretability analysis identifies two concrete directions for improvement. First, better attention allocation, so the model focuses on clinically relevant chunks rather than irrelevant ones. Second, better confidence calibration for rare codes, so that correctly ranked candidates are actually predicted. Addressing both would enable the model to produce fewer, more precise candidates, thereby reducing user burden and advancing practical deployment of the tool.

This thesis demonstrates that ICD coding assistance does not inherently require cloud infrastructure, external APIs, or a dedicated Python environment. A well-designed Java module, operating on the same server as the EHR, can deliver competitive predictions, resolve them against a structured clinical vocabulary, and persist them as part of the standard patient record. Whether this is sufficient to meet the practical needs of medical coders remains an open question. Nevertheless, the tool has been developed, made openly available, and the community has already started showing interest.

Bibliography

[1] Fei Teng, Yiming Liu, Tianrui Li, Yi Zhang, Shuangqing Li, and Yue Zhao. A review on deep neural networks for icd coding. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4357–4375, 2023. https://doi.org/10.1109/TKDE.2022.3148267.

[2] Beichen Kang, Xiaosu Wang, Yun Xiong, Yao Zhang, Chaofan Zhou, Yangyong Zhu, Zhang Jiawei, and Chunlei Tang. *Automatic ICD Coding Based on Segmented ClinicalBERT with Hierarchical Tree Structure Learning*, pages 250–265. 04 2023. https://doi.org/10.1007/978-3-031-30678-5_19.

[3] Hang Dong, Matúš Falis, William Whiteley, Beatrice Alex, Honghan Joshua, Ian Van Gessel, Shuhua Zhou, and Honghan Wu. Automated clinical coding: what, why, and where we are? *npj Digital Medicine*, 5(1):159, Oct 2022. https://doi.org/10.1038/s41746-022-00705-7.

[4] Chao-Wei Huang, Shang-Chi Tsai, and Yun-Nung Chen. Plm-icd: Automatic icd coding with pretrained language models. https://arxiv.org/abs/2207.05289, 2022.

[5] Joakim Edin, Alexander Junge, Jakob D. Havtorn, Lasse Borgholt, Maria Maistro, Tuukka Ruotsalo, and Lars Maaløe. Automated medical coding on MIMIC-III and MIMIC-IV: A critical review and replicability study. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23)*, SIGIR '23, page 2572–2582, New York, NY, USA, 2023. Association for Computing Machinery. https://doi.org/10.1145/3539618.3591918.

[6] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare*, 3(1):1–23, 2021. Article No.: 2. https://doi.org/10.1145/3458754.

[7] Emily Alsentzer, John Murphy, William Boag, Wei-Hung Weng, Di Jindi, Tristan Naumann, and Matthew McDermott. Publicly available clinical BERT embeddings. In Anna Rumshisky, Kirk Roberts, Steven Bethard, and Tristan Naumann, editors, *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 72–78, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-1909.

[8] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2019. http://dx.doi.org/10.1093/bioinformatics/btz682.

[9] Hude Quan, Bing Li, Lawrence D. Saunders, G. Adrien Parsons, Christian I. Nilsson, Arif Alibhai, William A. Ghali, and IMECCHI Investigators. Assessing validity of ICD-9-CM and ICD-10 administrative data in recording clinical conditions in a unique dually coded database. *Health Services Research*, 43(4):1424–1441, 2008. https://doi.org/10.1111/j.1475-6773.2007.00822.x.

[10] World Health Organization. ICD-10 version: 2019. https://icd.who.int/browse10/2019/en, 2019. Accessed: 2026-03-26.

[11] Centers for Medicare Medicaid Services (U.S.);National Center for Health Statistics (U.S.);American Hospital Association (AHA);American Health Information Management Association (AHIMA). ICD-10-CM Official Guidelines for Coding and Reporting. https://stacks.cdc.gov/view/cdc/250974, 2025. Accessed: 2026-04-20.

[12] Alistair Johnson, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. MIMIC-IV-Note: Deidentified free-text clinical notes. *PhysioNet*, January 2023. Version 2.2. https://doi.org/10.13026/1n74-ne17.

[13] Alexandre Blanquet and Pierre Zweigenbaum. A lexical method for assisted extraction and coding of icd-10 diagnoses from free text patient discharge summaries. In *American Medical Informatics Association Annual Symposium*, 1999. https://api.semanticscholar.org/CorpusID:44878368.

[14] Leah S. Larkey and W. Bruce Croft. Combining classifiers in text categorization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, page 289–297, New York, NY, USA, 1996. Association for Computing Machinery. https://doi.org/10.1145/243199.243276.

[15] Pengtao Xie and Eric Xing. A neural architecture for automated ICD coding. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1066–1076, Melbourne, Australia, July 2018. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1098.

[16] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1101–1111, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-1100.

[17] World Health Organization. ICD-11 coding tool: Mortality and morbidity statistics (MMS). https://icd.who.int/devct11/icd11_mms/en/current. Accessed: 2026-05-21.

[18] Gergely Héja and György Surján. Using n-gram method in the decomposition of compound medical diagnoses. *International Journal of Medical Informatics*, 70(2):229–236, 2003. MIE 2002 Special Issue. https://doi.org/10.1016/S1386-5056(03)00049-2.

[19] Yun-Zhi Chen, Hui-Juan Lu, and Lan-Juan Li. Automatic ICD-10 coding algorithm using an improved longest common subsequence based on semantic similarity. *PLOS ONE*, 12(3):e0173410, Mar 2017. https://doi.org/10.1371/journal.pone.0173410.

[20] Kevin P. Donnelly. SNOMED-CT: The advanced terminology and coding system for eHealth. *Studies in health technology and informatics*, 121:279–90, 2006. https://api.semanticscholar.org/CorpusID:22491040.

[21] Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32 Database issue:D267–70, 2004. https://api.semanticscholar.org/CorpusID:205228801.

[22] Anthony N. Nguyen, David Truran, Mary Kemp, Bevan Koopman, Daniel Conlan, Judith O'Dwyer, Michael Zhang, Sarvnaz Karimi, Hamed Hassanzadeh, Michael J. Lawley, and David Green. Computer-assisted diagnostic coding: Effectiveness of an NLP-based approach using SNOMED CT to ICD-10 mappings. *AMIA Annual Symposium Proceedings*, 2018:807–816, Dec 2018. PMID: 30815123; PMCID: PMC6371260. https://pmc.ncbi.nlm.nih.gov/articles/PMC6371260/.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. Version 3. https://arxiv.org/abs/1301.3781.

[24] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162.

[25] Yanshan Zhang, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong Lu. BioWordVec, improving biomedical word embeddings with subword information and MeSH. *Scientific Data*, 6(1):52, May 2019. https://doi.org/10.1038/s41597-019-0055-0.

[26] Qingyu Chen, Yifan Peng, and Zhiyong Lu. Biosentvec: creating sentence embeddings for biomedical texts. In *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, page 1–5. IEEE, 2019. http://dx.doi.org/10.1109/ICHI.2019.8904728.

[27] Fangyu Liu, Ehsan Shareghi, Zaiqiao Meng, Marco Basaldella, and Nigel Collier. Self-alignment pretraining for biomedical entity representations. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4228–4238, Online, June 2021. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.naacl-main.334.

[28] Akhila Abdulnazar, Roland Roller, Stefan Schulz, and Markus Kreuzthaler. Unsupervised SAPBERT-based bi-encoders for medical concept annotation of clinical narratives with SNOMED CT. *Digital Health*, 10:1–12, 2024. https://doi.org/10.1177/20552076241288681.

[29] Adler Perotte, Rimma Pivovarov, Karthik Natarajan, Nicole Weiskopf, Frank Wood, and Noémie Elhadad. Diagnosis code assignment: models and evaluation metrics. *Journal of the American Medical Informatics Association*, 21(2):231–237, 03 2014. https://doi.org/10.1136/amiajnl-2013-002159.

[30] Yuwen Chen and Jiangtao Ren. Automatic icd code assignment utilizing textual descriptions and hierarchical structure of icd code. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 348–353, 2019. https://doi.org/10.1109/BIBM47256.2019.8983078.

[31] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing Management*, 39(1):45–65, 2003. https://doi.org/10.1016/S0306-4573(02)00021-3.

[32] Stefan Berndorfer and Aron Henriksson. Automated diagnosis coding with combined text representations. In R. Randell, R. Cornet, and C. McCowan, editors, *Informatics for Health: Connected Citizen-Led Wellness and Population Health*, volume 235 of *Studies in Health Technology and Informatics*, pages 201–205. IOS Press, 2017. Open Access under CC BY-NC 4.0. https://doi.org/10.3233/978-1-61499-753-5-201.

[33] Priyanka Nigam. Applying deep learning to icd-9 multi-label classification from medical records. Technical report, Technical report, Stanford University, 2016. https://cs224d.stanford.edu/reports/priyanka.pdf.

[34] Chung-Chian Hsu, Pei-Chi Chang, and Arthur Chang. Multi-label classification of ICD coding using deep learning. In *2020 International Symposium on Community-centric Systems (CcS)*, pages 1–6, 2020. https://doi.org/10.1109/CcS49175.2020.9231498.

[35] Thanh Vu, Dat Quoc Nguyen, and Anthony Nguyen. A label attention model for ICD coding from clinical text. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3335–3341. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track. https://doi.org/10.24963/ijcai.2020/461.

[36] Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. https://arxiv.org/abs/1811.01727, 2019.

[37] Biplob Biswas, Thai-Hoang Pham, and Ping Zhang. Transicd: Transformer based code-wise attention model for explainable icd coding. https://arxiv.org/abs/2104.10652, 2021.

[38] Zachariah Zhang, Jingshu Liu, and Narges Razavian. Bert-xml: Large scale automated icd coding using bert pretraining. https://arxiv.org/abs/2006.03685, 2020.

[39] Xu Zhang, Kun Zhang, Wenxin Ma, Rongsheng Wang, Chenxu Wu, Yingtai Li, and S Kevin Zhou. A general knowledge injection framework for ICD coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7180–7189, Vienna, Austria, July 2025. Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.findings-acl.374.

[40] Guillermo López-García, José M. Jerez, Nuria Ribelles, Emilio Alba, and Francisco J. Veredas. Explainable clinical coding with in-domain adapted transformers. *Journal of Biomedical Informatics*, 139:104323, 2023. https://doi.org/10.1016/j.jbi.2023.104323.

[41] Leonor Barreiros, Isabel Coutinho, Gonçalo M. Correia, and Bruno Martins. Explainable ICD coding via entity linking. In *Proceedings of the Workshop on CL4Health at NAACL*, 2025. https://doi.org/10.48550/arXiv.2503.20508.

[42] Sarah Wiegreffe, Edward Choi, Sherry Yan, Jimeng Sun, and Jacob Eisenstein. Clinical concept extraction for document-level coding. In Dina Demner-Fushman, Kevin Bretonnel Cohen, Sophia Ananiadou, and Junichi Tsujii, editors, *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 261–272, Florence, Italy, August 2019. Association for Computational Linguistics. https://doi.org/10.18653/v1/W19-5028.

[43] Xindi Wang, Robert E. Mercer, and Frank Rudzicz. Multi-stage retrieve and re-rank model for automatic medical coding recommendation. https://arxiv.org/abs/2405.19093, 2024.

[44] Kunying Niu, Yifan Wu, Yaohang Li, and Min Li. Retrieve and rerank for automated icd coding via contrastive learning. *Journal of Biomedical Informatics*, 143:104396, 2023. https://doi.org/10.1016/j.jbi.2023.104396.

[45] OpenMRS Inc. Open Medical Record System (OpenMRS) product overview. https://openmrs.org/product/, 2026. Accessed: 2026-03-28.

[46] Ryan M Eshleman, Hui Yang, and Barry Levine. Structuring unstructured clinical narratives in openmrs with medical concept extraction. In *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 764–769, 2015. https://doi.org/10.1109/BIBM.2015.7359782.

[47] Michael Mugisha, Jean Baptiste Byiringiro, Melissa Uwase, Theophile Abizeyimana, Berchmas Ndikubwimana, Nadine Karema, Michele Kayiganwa, Candide Tran Ngoc, Nenad Kostanjsek, Can Celik, Donada Marc, Celestin Twizere, and David Tumusiime. Integration of International Classification of Diseases Version 11 Application Program Interface (API) in the Rwandan Electronic Medical Records (OpenMRS): Findings from Two District Hospitals in Rwanda. In *The Importance of Health Informatics in Public Health during a Pandemic*, volume 272 of *Studies in Health Technology and Informatics*, pages 280–283. IOS Press, 2020. https://doi.org/10.3233/SHTI200549.

[48] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. https://arxiv.org/abs/1310.4546, 2013.

[49] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. Presented at NIPS 2014 Deep Learning Workshop. https://arxiv.org/abs/1412.3555.

[50] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805, 2019.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017. https://arxiv.org/abs/1706.03762.

[52] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5797–5808. Association for Computational Linguistics, 2019. https://doi.org/10.18653/v1/P19-1580.

[53] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. https://api.semanticscholar.org/CorpusID:49313245.

[54] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 21(140):1–67, 2020. http://jmlr.org/papers/v21/20-074.html.

[55] Simon A. Lee, Anthony Wu, and Jeffrey N. Chiang. Clinical ModernBERT: An efficient and long context encoder for biomedical text. *arXiv preprint arXiv:2504.03964*, 2025. Version 1, [cs.CL]. https://arxiv.org/abs/2504.03964.

[56] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. https://doi.org/10.1016/j.neucom.2023.127063.

[57] National Library of Medicine. *Unified Medical Language System (UMLS)*. U.S. National Library of Medicine. https://www.nlm.nih.gov/research/umls/index.html. Accessed: 2026-03-21.

[58] Alistair E. W. Johnson, Lucas Bulgarelli, Lu Shen, Anne Gayraud, Alicia Erasmus, Steven Williams, Nicholas Ebert, Benjamin Moody, Braden Gow, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. MIMIC-IV, a freely accessible electronic health record dataset. *Scientific Data*, 10(1):1, Jan 2023. https://doi.org/10.1038/s41597-022-01899-x.

[59] Thanh-Tung Nguyen, Viktor Schlegel, Abhinav Kashyap, Stefan Winkler, Shao-Syuan Huang, Jie-Jyun Liu, and Chih-Jen Lin. Mimic-IV-ICD: A new benchmark for eXtreme MultiLabel Classification, 2023. https://arxiv.org/abs/2304.13998.

[60] Yuhao Wu, Yifan Wu, Wei Fan, and Min Li. Enhancing icd classification with semantic embedding rectification and long-tail refinement. *Knowledge-Based Systems*, 330:114530, 2025. https://doi.org/10.1016/j.knosys.2025.114530.

[61] Meta AI. Llama 3.1 technical report. *arXiv preprint arXiv:2407.21783*, 2024. https://arxiv.org/abs/2407.21783.

[62] Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. Biomistral: A collection of open-source pretrained large language models for medical domains. https://arxiv.org/abs/2402.10373, 2024.

[63] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. https://arxiv.org/abs/1908.10084.

[64] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. https://arxiv.org/abs/2002.10957.

[65] Dorian Brown. Rank-BM25: A Collection of BM25 Algorithms in Python, 2020. https://doi.org/10.5281/zenodo.4520057.

[66] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *IEEE Transactions on Big Data*, 12(2):346–361, 2026. https://doi.org/10.1109/TBDATA.2025.3618474.

[67] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3292500.3330701.

[68] Damian Pascual, Sandro Luck, and Roger Wattenhofer. Towards BERT-based automatic ICD coding: Limitations and opportunities. In Dina Demner-Fushman, Kevin Bretonnel Cohen, Sophia Ananiadou, and Junichi Tsujii, editors, *Proceedings of the 20th Workshop on Biomedical Language Processing*, pages 54–63, Online, June 2021. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.bionlp-1.6.

[69] Laila Rasmy, Yang Xiang, Ziqian Xie, Cui Tao, and Degui Zhi. Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction. *npj Digital Medicine*, 4(1):86, May 2021. https://doi.org/10.1038/s41746-021-00455-y.

[70] Xinhang Li, Xiangyu Zhao, Yong Zhang, and Chunxiao Xing. Towards automatic icd coding via knowledge enhanced multi-task learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, page 1238–1248, New York, NY, USA, 2023. Association for Computing Machinery. https://doi.org/10.1145/3583780.3615087.

[71] Xiaobo Li, Yijia Zhang, Xiaodi Hou, Shilong Wang, and Wen Qu. Multitask gated interactive network for automatic international classification of diseases coding with dual denoising mechanism. *Engineering Applications of Artificial Intelligence*, 159:111687, 2025. https://doi.org/10.1016/j.engappai.2025.111687.

[72] Jaejun Lee, Raphael Tang, and Jimmy Lin. What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning. https://arxiv.org/abs/1911.03090, 2019.

[73] B. J. Ferrell. Fine-tuning strategies for classifying community-engaged research studies using transformer-based models: Algorithm development and improvement study. *JMIR Formative Research*, 7:e41137, Feb 2023. https://doi.org/10.2196/41137.

[74] Consortium des Équipements de Calcul Intensif (CÉCI). High-performance computing resources and services. https://www.ceci-hpc.be/. Computational resources were provided by the CÉCI, funded by the F.R.S.-FNRS and the Walloon Region.

[75] Shaoxiong Ji, Matti Hölttä, and Pekka Marttinen. Does the magic of BERT apply to medical code assignment? a quantitative study. *Computers in Biology and Medicine*, 139:104998, 2021. https://doi.org/10.1016/j.compbiomed.2021.104998.

[76] Columbia International eHealth Laboratory. CIEL Concept Dictionary. Open Concept Lab, 2021. https://app.openconceptlab.org/#/orgs/CIEL/, Accessed: 2026-04-27.

[77] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995. https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale.

[78] Urchade Zaratiana, Gil Pasternak, Oliver Boyd, George Hurn-Maloney, and Ash Lewis. Gliner2: An efficient multi-task information extraction system with schema-driven interface. https://arxiv.org/abs/2507.18546, 2025.

[79] Sheng Zhang, Hao Cheng, Shikhar Vashishth, Cliff Wong, Jinfeng Xiao, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Knowledge-Rich Self-Supervision for Biomedical Entity Linking. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 868–880, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.findings-emnlp.61.

[80] Mujeen Sung, Hwisang Jeon, Jinhyuk Lee, and Jaewoo Kang. Biomedical entity representations with synonym marginalization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3641–3650, Online, July 2020. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.335.

[81] Dongfang Xu, Zeyu Zhang, and Steven Bethard. A generate-and-rank framework with semantic type regularization for biomedical concept normalization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8452–8464, Online, July 2020. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.748.

[82] Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. Gliner: Generalist model for named entity recognition using bidirectional transformer. https://arxiv.org/abs/2311.08526, 2023.

[83] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. https://arxiv.org/abs/1708.02002, 2018.

[84] Rian Touchent, Laurent Romary, and Eric de la Clergerie. Camembert-bio: Leveraging continual pre-training for cost-effective models on french biomedical data, 2024. https://arxiv.org/abs/2306.15550.

[85] Yanis Labrak, Adrien Bazoge, Richard Dufour, Mickael Rouvier, Emmanuel Morin, Béatrice Daille, and Pierre-Antoine Gourraud. DrBERT: Un modèle robuste pré-entraîné en français pour les domaines biomédical et clinique. In Christophe Servan and Anne Vilnat, editors, *Actes de CORIA-TALN 2023. Actes de la 30e Conférence sur le Traitement Automatique des Langues Naturelles (TALN), volume 4 : articles déjà soumis ou acceptés en conférence internationale*, pages 109–120, Paris, France, 6 2023. ATALA. https://aclanthology.org/2023.jeptalnrecital-international.13/.

[86] Keyang Xu, Mike Lam, Jingzhi Pang, Xin Gao, Charlotte Band, Piyush Mathur, Frank Papay, Ashish K. Khanna, Jacek B. Cywinski, Kamal Maheshwari, Pengtao Xie, and Eric P. Xing. Multimodal machine learning for automated icd coding. In Finale Doshi-Velez, Jim Fackler, Ken Jung, David Kale, Rajesh Ranganath, Byron Wallace, and Jenna Wiens, editors, *Proceedings of the 4th Machine Learning for Healthcare Conference*, volume 106 of *Proceedings of Machine Learning Research*, pages 197–215. PMLR, 09–10 Aug 2019. https://proceedings.mlr.press/v106/xu19a.html.

Appendix A

Lyra

This appendix provides a concise guide for people who wish to run experiments on the Lyra cluster. The full CÉCI documentation[1] should be consulted for any advanced usage.

A.1 Connecting to the cluster

Lyra is accessed via SSH to `lyra.ulb.be` on port 22. A CÉCI account is required and can be created at the CÉCI support portal. Authentification uses your CÉCI passphrase and an SSH key file, which is generated during account creation and must be configured on your local machine. The full connection setup procedure is described in the CÉCI documentation.
Once connected, you land on a login node. No computation should be run there. It is only used to prepare scripts, manage files, and submit jobs.

A.2 Managing storage

Storage is split into three distinct areas, each with a different purpose and quota.
First, he home directory has a quota of 100 GB. It should contain only code, configuration files, and small outputs.

For large files such as datasets or model checkpoints, the global scratch space should be used instead, with a quota of 5 TB per user. This high-performance disk is shared between all compute nodes. A personal directory can be created there:

```
mkdir -p $GLOBALSCRATCH/ndevlees_data
```

To access it conveniently from the home directory, a symbolic link can be created:

```
ln -s $GLOBALSCRATCH/ndevlees_data ~/data_scratch
```

Finally, the local scratch is a temporary directory local to the compute node, defined dynamically during a job. It is useful for very fast I/O operations during training, but its content is lost when the job ends.

To transfer files to and from the clusters, the recommended command is `scp`. For instance to load a file:

```
scp path/to/file/on/local/computer ceci_username@cluster_name:destination/path/on/
    cluster
```

A.3 Setting up a Python environment

It is strongly recommended to use a virtual environment to isolate dependencies from the system modules and ensure reproducibility across runs. On CÉCI clusters, Python is not available by default but is provided through the module system. The `module purge` command clears any previously loaded modules, and `module load` loads a specific version. This ensures a clean and

[1] CÉCI Consortium. Documentation and User Guide for High-Performance Computing resources. https://support.ceci-hpc.be/doc/, 2026. Accessed: 2026-03-28. The CÉCI is funded by the F.R.S.-FNRS and the Walloon Region.

reproducible software environment.
The virtual environment is created once on the login node:

```
module purge
module load Python/3.11.3-GCCcore-12.3.0
python -m venv ~/.venv
source ~/.venv/bin/activate
pip install torch transformers accelerate peft sentence-transformers
```

Once created, the virtual environment must be activated in every Slurm job script before running any Python code, as shown in the example below.

A.4 Submitting jobs with Slurm

Lyra uses Slurm as its job scheduler. Rather than running programs directly, users submit batch scripts. The scripts describe the requested resources and the commands to execute. Slurm queues the job and dispatches it to a compute node when the requested resources become available. Jobs are submitted with:

```
sbatch my_script.sh
```

Anatomy of a job script

A job script is a Bash script where lines prefixed with `#SBATCH` are interpreted by Slurm as resource directives. These directives must appear at the top of the file, before any executable command. The following script was used throughout this thesis:

```
#!/bin/bash
#SBATCH --job-name=ModernBERT_train
#SBATCH --output=logs_%j_%x.txt
#SBATCH --error=errors_%j_%x.txt
#SBATCH --partition=batch
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=80G
#SBATCH --time=2-00:00:00
#SBATCH --gpus=1
module purge
module load Python/3.11.3-GCCcore-12.3.0
source ~/.venv/bin/activate

export JOB_DIR=~/data_scratch/job_$SLURM_JOB_ID
mkdir -p $JOB_DIR

cd ~/modelC
srun python -u train_modelC_cls.py --output_dir $JOB_DIR

FINAL_DESTINATION=~/modelC/results_$SLURM_JOB_ID
mkdir -p $FINAL_DESTINATION
cp $JOB_DIR/* $FINAL_DESTINATION/
rm -rf $JOB_DIR
```

Each directive controls a specific aspect of the job:

- `--job-name` sets a human readable label for the job, visible in `squeue` output.
- `--output` and `--error` redirect standard output and standard error to separate log files. The `%j` placeholder is replaced by the Slurm job ID and `%x` by the job name, making it easy to trace logs back to a specific run.
- `--partition=batch` selects the partition on Lyra where GPU nodes are available. The default partition (`debug`) has a 30 minute time limit and is not suitable for training runs.
- `--ntasks=1` declares that the job consists of a single process. This is appropriate for single GPU Python training scripts.
- `--cpus-per-task=8` allocates 8 CPU cores to that process, which are used by PyTorch's data loaders (`num_workers`).
- `--mem=80G` sets the total RAM allocated to the job. The job is killed if it exceeds this limit.

- `--time=2-00:00:00` sets the maximum wall clock duration in `D-HH:MM:SS` format. The job is automatically terminated if it runs longer.
- `--gpus=1` requests one GPU. This is the syntax recommended for Lyra; the GPU is allocated exclusively to the job.

The `srun` command at the end launches the actual process inside the allocated resources. Although it can be omitted for single step jobs, using it is recommended as it enables more accurate resource reporting and proper signal handling.

As recommended by the CECI documentation, the files generated by our batch job are stored inside the globalscratch. The directory is cleaned once the job finishes, and all files are transferred back to the home file system. If our script needed large input files, we would also place them under globalscratch directory.

Monitoring and cancelling jobs

Once submitted, the job ID is returned immediately. Jobs can be monitored with:

```
squeue --me
```

A job in the `PD` (pending) state is waiting for resources; `R` means it is running. The reason column explains why a pending job has not started yet (e.g., `Resources` means insufficient resources are currently free, `Priority` means higher-priority jobs are ahead in the queue). To cancel a job:

```
scancel <job_id>
```

Checking available resources

Before submitting a job, the `sinfo` command gives an overview of available partitions and node states.

Appendix B

Hyperparameters of our models for the full note task

Model	LR	Stride	Max Chunks	Batch Size	Epochs	pos_weight
TF-IDF‡	—	—	—	—	—	—
ModelB_mean	4.87e-5	256	16	8	5	—
ModelC_cls	2e-5	256	12	6	5	—
ModelC_allword	1.14e-5	384	12	4	5	—
ModelC_cls_langchain	2e-5	256	12	6	5	—
ModelC_ModernBERT	1.61e-5	256	12	2	5	—
ModelD_masking	—	256	12	4	5	—
ModelD_hierarchy	BERT: 5e-6 / heads: 2e-4	256	12	4	**10**	chap:5, cat3:15, full:30†

Table B.1: Hyperparameters used for all trained models. †ModelD_hierarchy uses a weighted multi-task loss: $\mathcal{L} = 0.3\,\mathcal{L}_{\text{chap}} + 0.5\,\mathcal{L}_{\text{cat3}} + 1.0\,\mathcal{L}_{\text{full}}$. ‡TF-IDF hyperparameters: ngram_range=(1,3), max_features=20,000, C=9.97 (logistic regression, optimised via cross-validation).

Appendix C

Additional plots regarding the short excerpts task

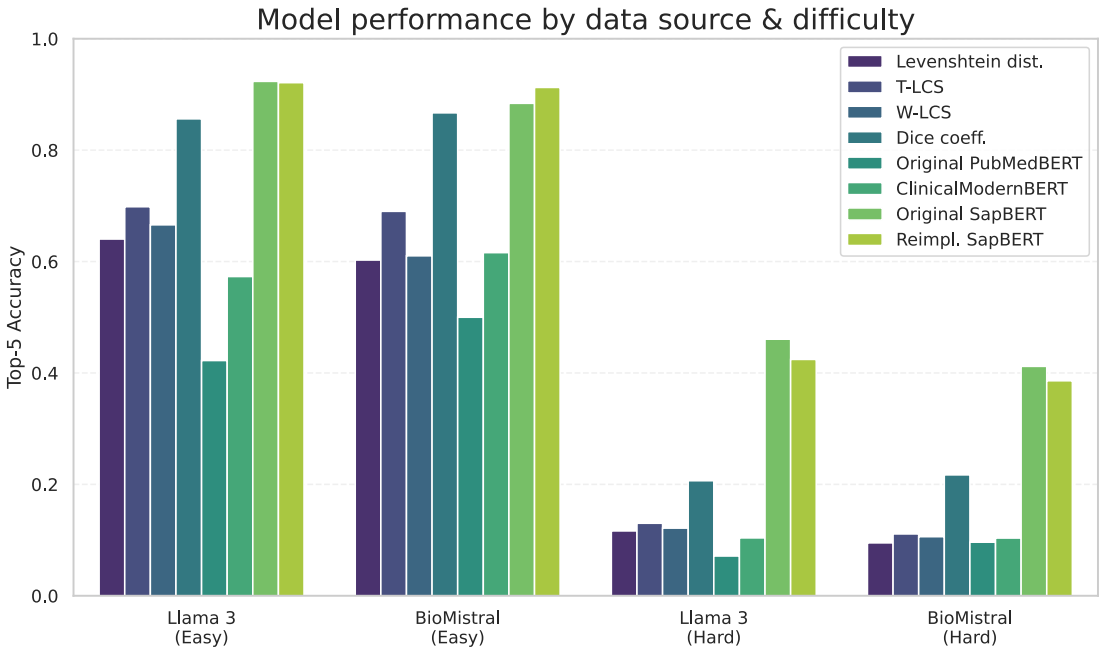


Figure C.1: Performance by source LLM and difficulty level. Results are broadly consistent across Llama 3 and BioMistral for a given level of difficulty, confirming that the choice of generation model does not systematically bias evaluation outcomes.

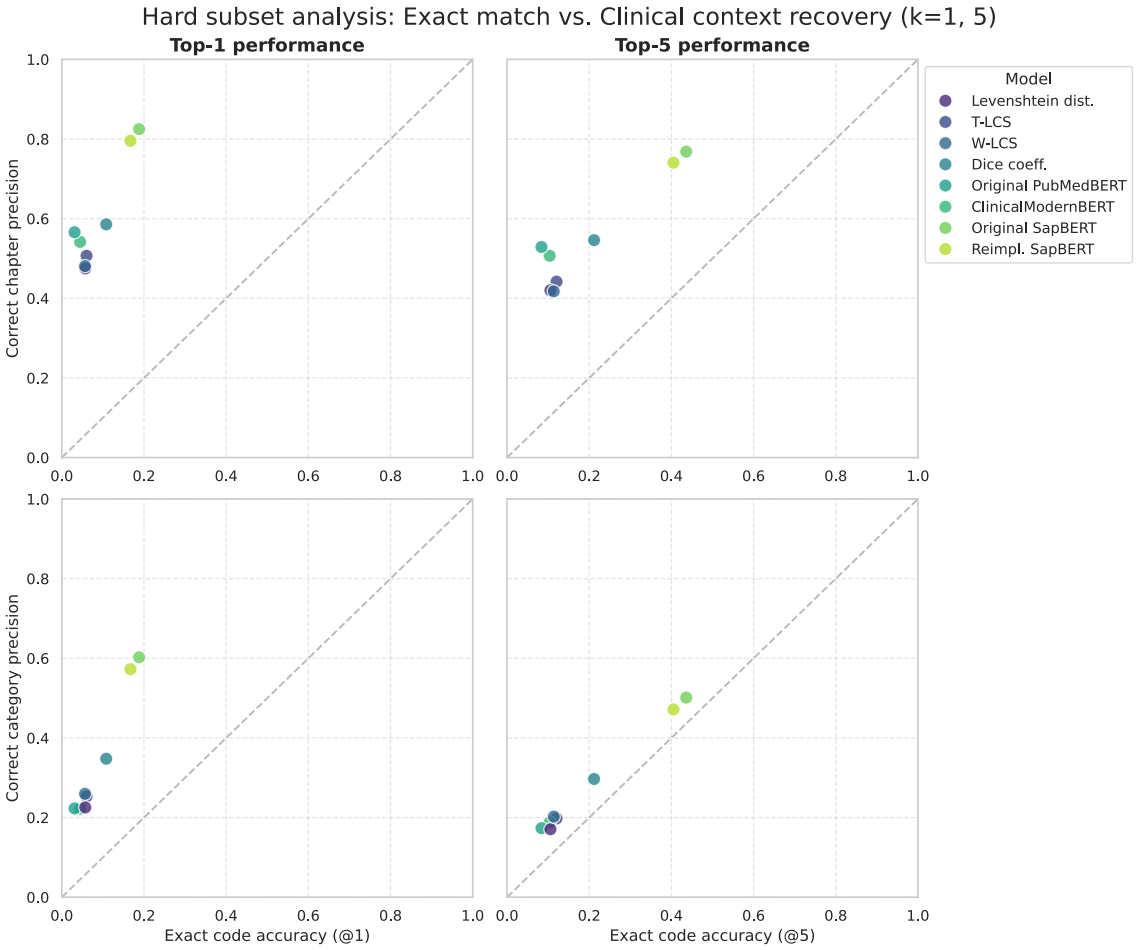


Figure C.2: Near-miss analysis on the hard subset at chapter and category levels for k=1 and k=5. Each point represents a model. Points above the diagonal indicate models that recover on average the correct broader concept even when the exact code is missed, revealing partial understanding. Precision@k takes into account the *proportion* of correct chapter of category among the k candidates. It penalises models return candidates from many different chapters or categories in order to recover the correct one.

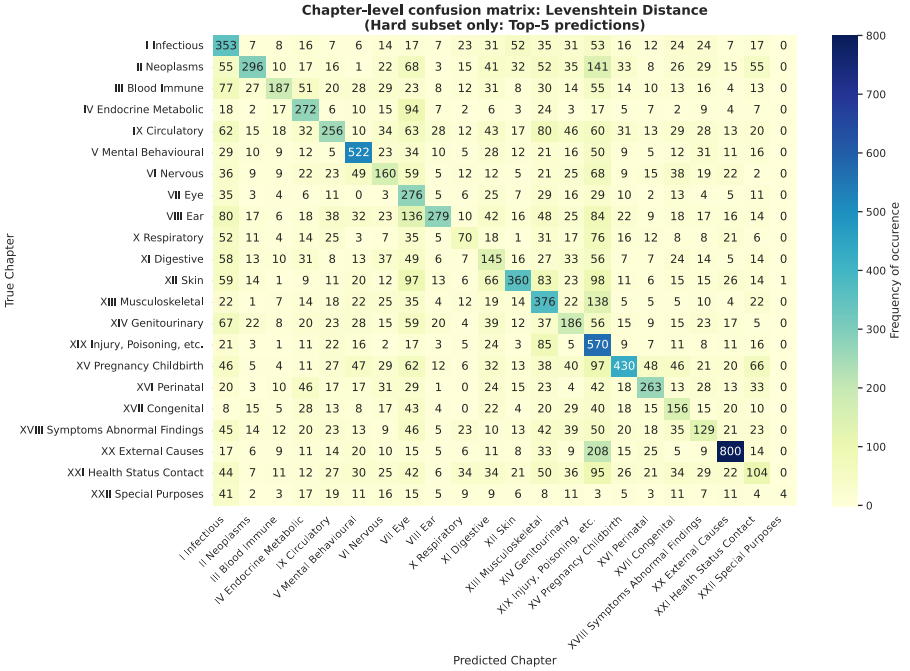


Figure C.3: Chapter-level confusion matrix for Levenshtein distance on the hard subset (Top-5 predictions). The matrix reveals a tendency to confuse clinically adjacent chapters, and chapters with overlapping terms descriptions. The misclassification into Chapter XIX (Injury, Poisoning, etc.) is most visible Chapter XX (External causes), Chapter II (Neoplasms), and Chapter XIII (Musculoskeletal). The method makes many errors in predictions for Chapter X (Respiratory).

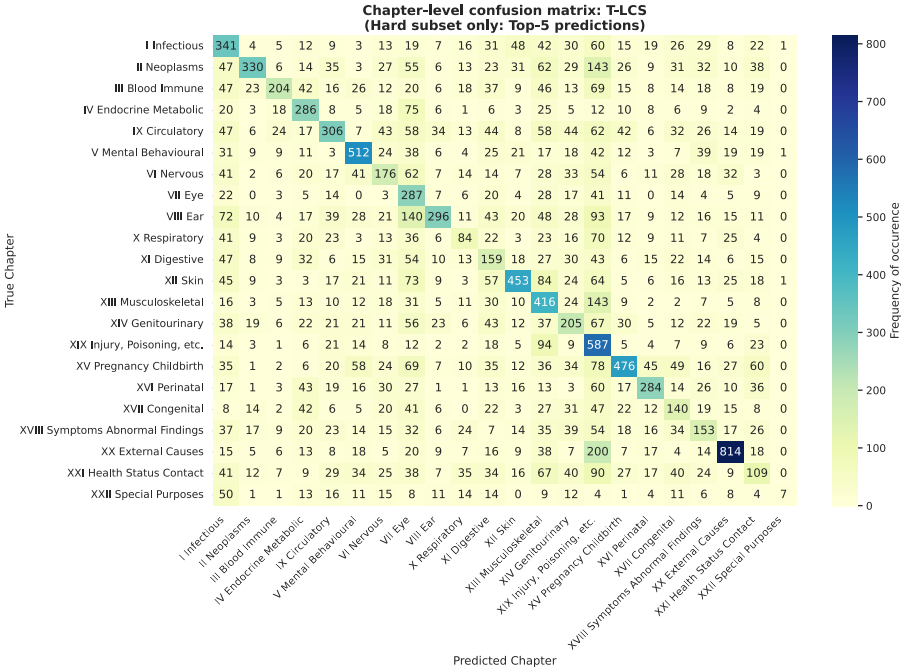


Figure C.4: Chapter-level confusion matrix for T-LCS on the hard subset (Top-5 predictions). Confusion patterns are broadly similar to those of Levenshtein distance, reflecting the shared lexical nature of both baselines.

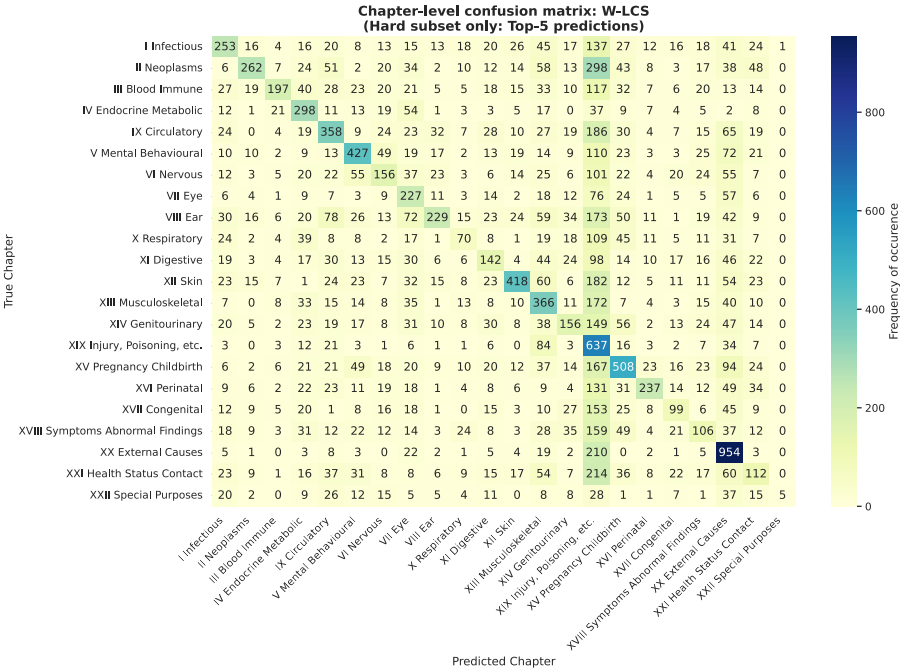


Figure C.5: Chapter-level confusion matrix for W-LCS on the hard subset (Top-5 predictions). This method has the strongest attraction to Chapter XIX (Injury, Poisoning, etc.): most chapters misclassifies more than 100 examples, sometimes 150-200, into Chapter XIX.

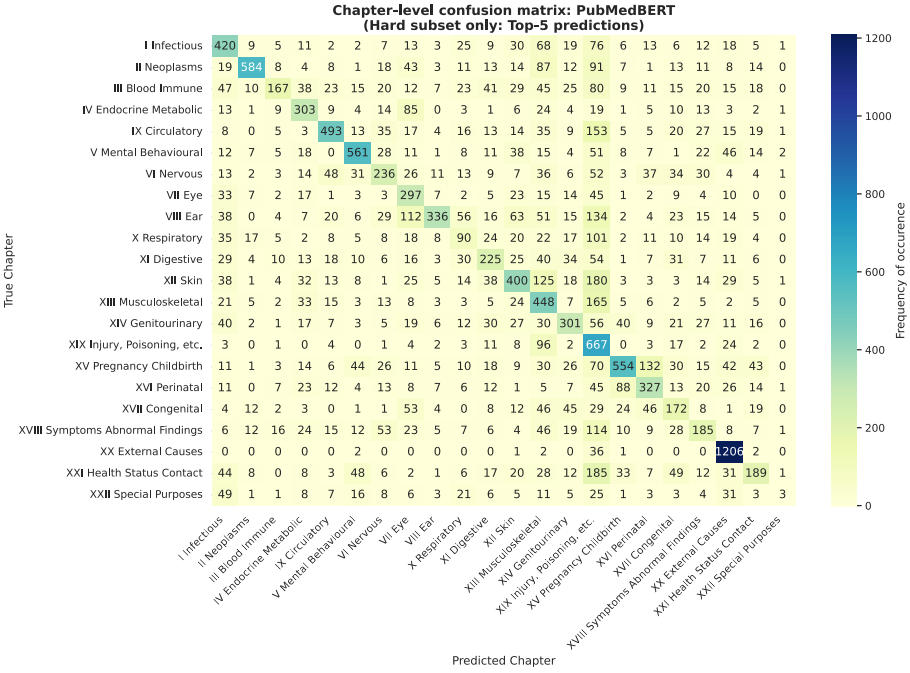


Figure C.6: Chapter-level confusion matrix for PubMedBERT on the hard subset (Top-5 predictions). Compared to lexical baselines, PubMedBERT has a stronger diagonal.

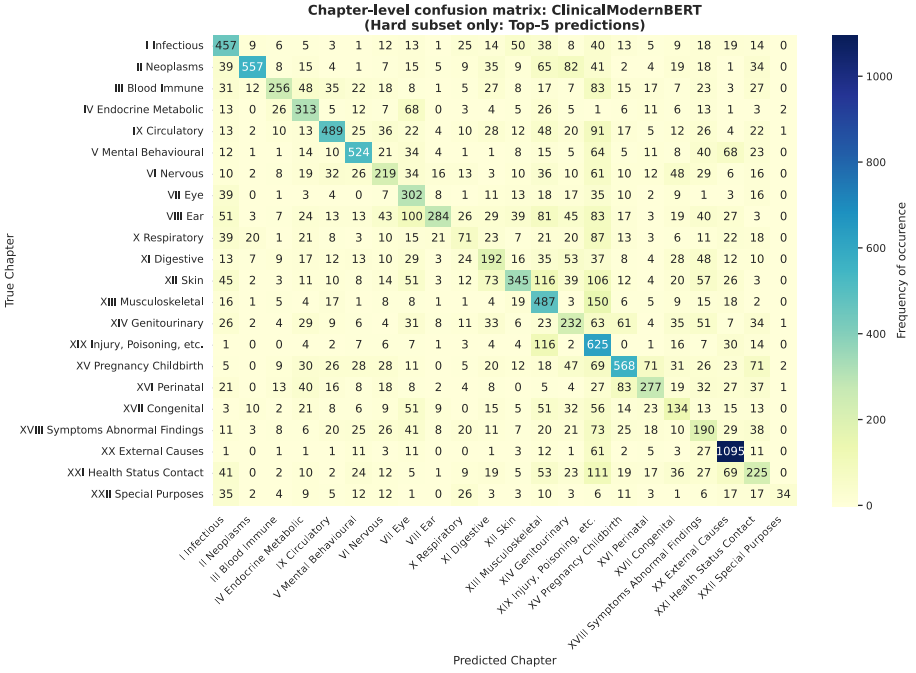


Figure C.7: Chapter-level confusion matrix for Clinical ModernBERT on the hard subset (Top-5 predictions). The results are very similar to those of PubMedBERT. However, Clinical ModernBERT displays a weakest attraction to Chapter XIX (Injury, Poisoning, etc.)

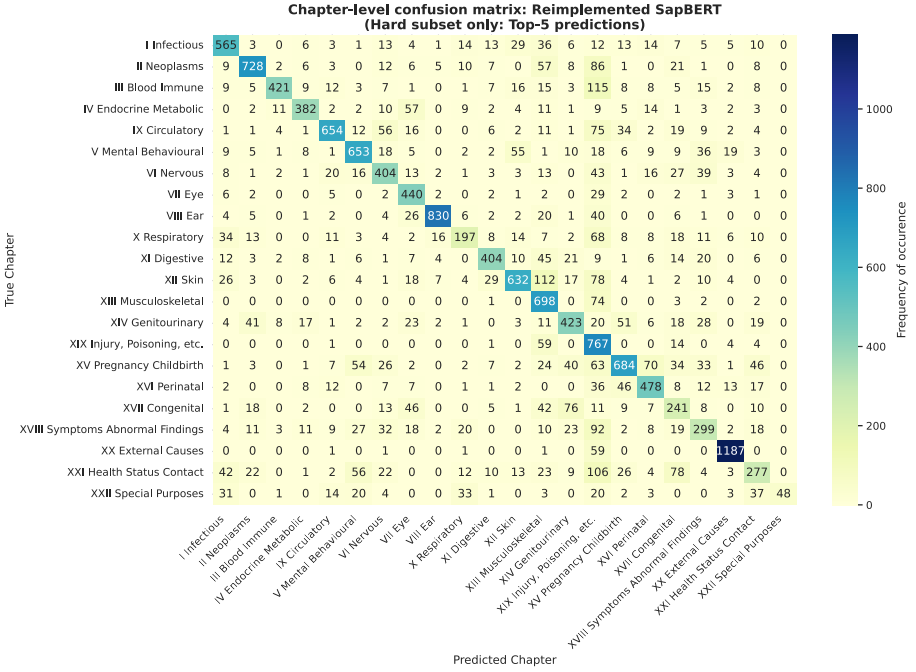


Figure C.8: Chapter-level confusion matrix for reimplemented SapBERT on the hard subset (Top-5 predictions). The diagonal is markedly stronger than all other models in this appendix, consistent with its leading MRR score and the previous analysis of original SapBERT. Residual confusion is concentrated in clinically ambiguous chapters. Examples from Chapters X (Respiratory) and Chapter XVII (Congenital) are still the most challenging.

Appendix D

Results of our models on our dataset for the full note task

Table D.1: Our models results — *full* task (Full ICD codes)

Model	Micro-F1	Macro-F1	R@5	R@10	R@15	R@20	R@50	P@5	P@10	P@20
TF-IDF	0.4564	0.0394	0.2634	0.3922	0.4711	0.5242	0.6748	0.6425	0.5069	0.3553
ModelB_mean_optuna	0.4290	0.0238	0.2514	0.3679	0.4402	0.4899	0.6400	0.6214	0.4791	0.3323
ModelGRU	—	—	—	—	—	—	—	—	—	—
ModelC_cls_allword_optuna	0.0000	0.0000	0.0641	0.0992	—	0.1435	0.2181	0.1774	0.1391	—
ModelC_cls_attention	0.4334	0.0313	0.2633	0.3904	—	0.5187	0.6665	0.6343	0.4968	—
ModelC_cls_langchain	0.3793	0.0137	0.2242	0.3288	—	0.4435	0.5941	0.5576	0.4276	—
ModelC_cls_modernbert_optuna	0.4452	0.0415	0.2663	0.3977	0.4752	0.5269	0.6683	0.6440	0.5085	0.3528
ModelD_masking	0.4462	0.0332	—	0.4009	—	0.5327	0.6682	0.6425	0.5073	—
ModelD_hierarchy	**0.4285**	**0.0677**	**0.2598**	**0.3942**	—	**0.5386**	**0.6999**	**0.6145**	**0.4923**	—

Table D.2: Our models results — *3char* task (First 3 characters of the ICD code)

Model	Micro-F1	Macro-F1	R@5	R@10	R@15	R@20	R@50	P@5	P@10	P@20
TF-IDF	0.5928	0.1933	0.3465	0.5253	0.6295	0.6980	0.8574	0.7740	0.6299	0.4439
ModelB_mean_optuna	0.5777	0.1286	0.3452	0.5211	0.6212	0.6836	0.8329	0.7779	0.6295	0.4365
ModelGRU	0.5440	0.0752	0.3469	0.5206	0.6170	0.6766	0.8228	0.7804	0.6278	0.4301
ModelC_cls_allword_optuna	0.6814	0.2759	0.3837	0.5952	—	0.7812	0.9080	0.8441	0.7095	—
ModelC_cls_attention	0.6349	0.2084	0.3680	0.5657	—	0.7371	0.8696	0.8108	0.6742	—
ModelC_cls_langchain	0.5918	0.1372	0.3522	0.5349	—	0.6949	0.8379	0.7865	0.6410	—
ModelC_cls_modernbert_optuna	0.6289	0.2209	0.3568	0.5458	0.6526	0.7155	0.8533	0.7883	0.6500	0.4535
ModelD_masking	0.6358	0.2080	—	0.5666	—	0.7364	0.8622	0.8115	0.6749	—
ModelD_hierarchy	**0.5838**	**0.2407**	**0.3420**	**0.5301**	—	**0.7091**	**0.8634**	**0.7507**	**0.6248**	—

Table D.3: Our models results — *chapters* task (ICD chapters)

Model	Micro-F1	Macro-F1	R@5	R@10	R@15	R@20	P@5	P@8	P@10	P@20
TF-IDF	0.8166	0.6606	0.6453	0.9086	0.9891	0.9998	0.8548	0.7274	0.6472	0.3657
ModelB_mean_optuna	0.8411	0.7077	0.6630	0.9260	0.9924	0.9998	0.8735	0.7455	0.6600	0.3657
ModelGRU	0.8472	0.7195	0.6651	0.9277	0.9927	0.9999	0.8756	0.7484	0.6614	0.3657
ModelC_cls_allword_optuna	0.8596	0.7386	0.6709	0.9347	—	0.9999	0.8830	—	0.6674	—
ModelC_cls_attention	0.8460	0.7185	0.6656	0.9267	—	0.9996	0.8762	—	0.6604	—
ModelC_cls_langchain	0.5606	0.4405	0.3882	0.6227	—	0.9280	0.5236	—	0.4372	—
ModelC_cls_modernbert_optuna	0.8285	0.6909	0.6514	0.9119	0.9895	0.9996	0.8605	0.7323	0.6491	0.3656
ModelD_masking	0.8459	0.7184	—	0.9267	—	0.9996	0.8762	—	0.6604	—
ModelD_hierarchy	**0.8349**	**0.7379**	**0.6580**	**0.9222**	—	**0.9999**	**0.8670**	—	**0.6565**	—

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**