ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

Scuola di Scienze

Corso di Laurea in Ingegneria e Scienze Informatiche

FROM WORDS TO CODES: LARGE LANGUAGE MODELS FOR ICD-9 EXTRACTION IN CLINICAL DOCUMENTS

Elaborato in
Programmazione di applicazioni data intensive

Relatore
Prof. Gianluca Moro

Co-relatori
Dott. Giacomo Frisoni

Presentata da
Salvatore Antonio Addimando

Seconda Sessione di Laurea
Anno Accademico 2022 – 2023

KEYWORDS

Natural Language Processing
Named Entity Recognition
Knowledge Distillation
Large Language Models
ICD-9-CM

Look up at the stars, not down your feet.
- Stephen Hawking

Saving someone means helping them when they are frightened;
but true heroes not only save lives but also people's hearts...
No matter how scared you are, always smile as if everything is alright.
In this world, those who smile are the strongest!
- Kōhei Horikoshi

Sommario

Spinti dalle recenti scoperte del deep learning, i modelli di generazione del linguaggio naturale (NLG) sono stati al centro di costanti progressi negli ultimi anni. Tuttavia, poiché la nostra capacità di generare testo artificiale indistinguibile dall'uomo è in ritardo rispetto alla nostra capacità di valutarlo, è fondamentale sviluppare e applicare metriche di valutazione automatica ancora migliori. Per facilitare ai ricercatori una valutazione generale dell'efficacia dei loro modelli, proponiamo NLG-Metricverse—una libreria open-source end-to-end per la valutazione NLG basata su Python. Questo framework fornisce una raccolta vivente di metriche NLG in un ambiente unificato e di facile utilizzo, fornendo strumenti per applicarli, analizzarli, confrontarli e visualizzarli in modo efficiente. Ciò include (i) l'ampio supporto a metriche automatiche eterogenee con gestione delle n-arità, (ii) la meta-valutazione sulle prestazioni individuali, le correlazioni metrica-metrica e metrica-umano, (iii) interpretazioni grafiche per aiutare gli esseri umani a ottenere meglio le intuizioni del punteggio, (iv) categorizzazione formale e documentazione conveniente per accelerare la comprensione delle metriche. NLG-Metricverse mira ad aumentare la comparabilità e la replicabilità della ricerca NLG, auspicabilmente stimolando nuovi contributi nell'area.

Abstract

In the realm of real-world named entity recognition and classification (NERC), the utilization of ICD-9 codes proves to be invaluable. Annotation scarcity and the need to generalize to unseen types present formidable obstacles in the field. In such situations, leveraging ICD-9 codes becomes a pivotal strategy to address these challenges effectively. ICD-9 codes offer a standardized system for categorizing medical diagnoses and procedures, providing a well-established framework to classify entities within the healthcare domain. By incorporating ICD-9 codes into NERC systems, practitioners can harness a wealth of prior knowledge and domain-specific information, thus enhancing the accuracy and efficiency of their NERC models. This integration empowers NERC systems to achieve remarkable outcomes, ensuring the precise identification and classification of medical entities, ultimately benefiting both healthcare providers and patients. Although large language models (LLMs) hold great potential, computational cost and inefficiency can limit their applicability, favoring smaller specialized networks. In this paper, we introduce ICD-Juicer, a novel LLM distillation framework tailored for improving NERC performance in resource-constrained environments, specifically focusing on the extraction of ICD-9 codes from clinical documents. Mechanically, ICD-Juicer transfers LLM clinical knowledge to BERT-based models. Accuracy is further enhanced by incorporating textual target class short descriptions in the prompt. We conducted extensive prompt engineering with different LLMs on an extremely large dataset, counting almost 2 million medical reports covering a wide variety of domains. It is important to note that the prompt provides document-level annotations from the MIMIC-III dataset to restrict GPT-3.5-turbo's output options. This way we managed to get consistent outputs throughout the creation of the augmented dataset. Our knowledge-distilled models consistently outperform state-of-the-art baselines, demonstrating the benefits of a wide range of observed classes. Furthermore, compared to other approaches, ICD-Juicer achieves superior results with significantly fewer parameters.

Introduzione

Il riconoscimento e la classificazione di entità nominative (NERC) consiste nell'identificare le menzioni di entità nominative e nel categorizzarle in tipi di entità specifici da un set predefinito. Nonostante il crescente utilizzo di modelli linguistici pre-addestrati in questo campo [1], le soluzioni esistenti in genere richiedono set di dati estesi e danno priorità a classi di entità comuni come persona, luogo e organizzazione [2]. Le applicazioni NERC del mondo reale spesso si occupano di domini specializzati e di una carenza di etichette per classi meno comuni, che sono in continua evoluzione. Di conseguenza, fare affidamento su o riaddestrare periodicamente modelli allo stato dell'arte (SOTA) diventa impraticabile. Inoltre, la creazione di dati di verità di base sostanziali è un'attività costosa e dispendiosa in termini di tempo, soggetta a errori e incoerenze umani che possono influenzare le prestazioni del modello.

Seguendo il paradigma dell'apprendimento in-context, i modelli di linguaggio di grandi dimensioni (LLM) hanno dimostrato risultati impressionanti nella generazione di testo di alta qualità condizionato su istruzioni specifiche [3]. Alimentati da un ampio pre-training su enormi volumi di testo grezzo, gli LLM possono essere preziose fonti di conoscenza per migliorare il riconoscimento e la classificazione di entità nominative (NERC) in contesti impegnativi. D'altra parte, le dimensioni stesse di questi modelli pongono notevoli ostacoli al loro dispiegamento. Ad esempio, per servire un LLM da 175B è necessaria una RAM GPU di almeno 350GB su un'infrastruttura specializzata [4]. Gli LLM allo stato dell'arte di oggi superano facilmente i 500B [5], aumentando drasticamente il fabbisogno di calcolo. Risorse paragonabili non sono semplicemente accessibili alla maggior parte dei team di prodotto, soprattutto quelli che cercano inferenze a bassa latenza. I modelli di linguaggio di piccole dimensioni (SLM) offrono un'alternativa più accessibile, trasparente ed efficiente a causa della loro capacità limitata. I ricercatori hanno esplorato nuove strategie di addestramento, con la distillazione della conoscenza (KD) che occupa un posto centrale.

Proponiamo che i modelli linguistici (LLM) contengano una conoscenza delle

attività, riducendo la necessità di ampi dati etichettati manualmente nell'addestramento dei modelli da sequenza a etichetta (SLM). A supportare il nostro approccio c'è l'opinione dell'Executive Director dell'Institute for Experimental AI della Northeastern University Usama Fayyad in un post recente su LinkedIn `https://www.linkedin.com/posts/eai-nu_experts-discuss-problems-potential-of-large-activity-7110635718988742656--GUe?utm_source=share&utm_medium=member_desktop`. Per migliorare il riconoscimento e la classificazione di entità nominate (NERC) a colpo zero per gli SLM, introduciamo ICD-Juicer, un framework unico che utilizza la distillazione della conoscenza (KD) dagli LLM (Figura **??**). Inizialmente, impieghiamo un LLM pre-addestrato per creare un ampio set di dati di distillazione del dominio attraverso l'aumento dei dati guidato da esempi su testo grezzo, affrontando i limiti dell'annotazione manuale. Per affrontare problemi come l'eccessiva confidenza e l'allucinazione degli LLM, introduciamo varie strategie di verifica per normalizzare le pseudo-etichette e filtrare i casi irrilevanti. Successivamente, il nostro approccio prevede un processo di KD a due fasi. Nella prima fase, un modello studente basato su BERT viene addestrato a imitare le annotazioni delle entità fornite dall'LLM sul set di dati sintetici, che include annotazioni legate al tipo. Nella seconda fase, il modello studente viene specializzato per l'attività NERC a valle. Nonostante non utilizzi un approccio ZSL, il set di dati MIMIC-III, con i suoi 1.159 codici ICD-9-CM unici rilevati nei 2.083.180 referti medici, ci ha permesso di far generalizzare il modello basato su BERT a diverse condizioni mediche e di riconoscere le diagnosi con maggiore accuratezza. Rendiamo disponibili il metodo KD proposto e i modelli distillati, e non vediamo l'ora di pubblicare il dataset MIMIC-III completamente annotato.

Introduction

Named entity recognition and classification (NERC) involves identifying named entity mentions and categorizing them into specific entity types from a predefined set. Despite the growing utilization of pre-trained language models in this field [1], existing solutions typically require extensive datasets and prioritize common entity classes such as person, location, and organization [2]. Real-world NERC applications often deal with specialized domains and a shortage of labels for less common classes, which constantly evolve. Consequently, relying on or periodically retraining state-of-the-art (SOTA) models becomes impractical. Additionally, creating substantial ground truth data is a costly and time-consuming endeavor prone to human errors and inconsistencies that can impact model performance.

Following the in-context learning paradigm, large language models (LLMs) have demonstrated impressive results in generating high-quality text conditioned on specific instructions [3]. Powered by extensive pre-training on huge volumes of raw text, LLMs can be valuable sources of knowledge for improving Named Entity Recognition and Classification (NERC) in challenging contexts. On the flip side, the sheer size of these models poses notable deployment obstacles. For instance, serving a 175B LLM requires at least 350GB of GPU RAM on a specialized infrastructure [4]. Today's state-of-the-art LLMs easily exceed 500B [5], drastically increasing the computation need. Comparable resources are simply not affordable for most product teams, especially those seeking low-latency inference. Small language models (SLMs) offer a more accessible, transparent, and efficient alternative due to their limited capacity. Researchers have explored novel training strategies, with knowledge distillation (KD) taking center stage.

We propose that Language Models (LLMs) contain task knowledge, reducing the need for extensive human-labeled data in training Sequence-to-Label Models (SLMs). To support our approach there is the opinion of the Executive Director of the Institute for Experimental AI at Northeastern University Usama Fayyad in a recent post on LinkedIn `https://www.linkedin.com/posts/eai`

-nu_experts-discuss-problems-potential-of-large-activity-7110635718988742656--GUe?utm_source=share&utm_medium=member_desktop. To improve Named Entity Recognition and Classification (NERC) for SLMs, we introduce ICD-Juicer, a unique framework utilizing Knowledge Distillation (KD) from LLMs (Figure **??**). Initially, we employ a pre-trained LLM to create a comprehensive domain distillation dataset through example-guided data augmentation on raw text, addressing the limitations of manual annotation. To address issues like LLM overconfidence and hallucination, we introduce various verification strategies to normalize pseudo-labels and filter out irrelevant cases. Subsequently, our approach involves a two-stage KD process. In the first stage, a BERT-based student model is trained to mimic the entity annotations provided by the LLM on the synthetic dataset, which includes type-bounded annotations. In the second stage, the student model is specialized for the downstream NERC task. Despite not using a ZSL approach, the MIMIC-III dataset, with its 1,159 unique ICD-9-CM codes detected across the 2,083,180 medical reports, allowed us to make the BERT-based model generalize across different medical conditions and recognize diagnoses more accurately. We openly release the KD recipe, and the distilled models, and we're also looking forward to releasing the fully annotated MIMIC-III dataset.

Contents

List of Figures

List of Tables

Chapter 1

Theoretical Framework

This chapter will describe the worlds of natural language processing and natural language generation, and then move on to discuss evaluation metrics and resulting concepts.

1.1 ICD-9-CM

La classificazione ICD-9-CM descrive in codici numerici o alfa-numerici i termini medici in cui sono espressi le diagnosi di malattia o di traumatismo, gli altri problemi di salute, le cause di traumatismo e le procedure diagnostiche e terapeutiche. I caratteri fondamentali della ICD-9-CM sono i seguenti:

- l'esaustivita' : tutte le entita' trovano una loro collocazione, piu' o meno specifica, entro i raggruppamenti finali della classificazione;
- la mutua esclusivita' : ciascuna entita' e' classificabile soltanto in uno dei raggruppamenti finali della classificazione;
- il numero limitato di raggruppamenti: circa 16.000 codici consentono la classificazione delle diagnosi, dei problemi di salute e delle principali procedure diagnostiche e terapeutiche;
- la specificita' dei raggruppamenti in ragione della rilevanza delle entita' nosologiche dal punto di vista della sanita' pubblica: le entita' nosologiche di particolare importanza per la sanita' pubblica o che si verificano con maggiore frequenza sono individuate da una specifica categoria; tutte le altre entita' nosologiche sono raggruppate in categorie non strettamente specifiche, che comprendono condizioni differenti, benche¤ tra loro correlate.

La struttura della classificazione e' determinata da due assi principali, l'eziologia e la sede anatomica.

1.2 Generative AI and Large Language Models

Neural networks underlie Language Models (LMs), providing them with a structure capable of estimating the probability distribution of token sequences and serving as predictors for subsequent tokens in a sequence. The likelihood of the sentence *"I am going to the park"* surpasses that of *"am going I the park to"*.
Any AI model tasked with generating fresh data, be it images, explanations of intricate concepts, or narratives, falls under the category of Generative AI models. During the training phase, it acquires knowledge from an existing dataset and subsequently crafts new data that aligns with the given context, bearing similar characteristics.

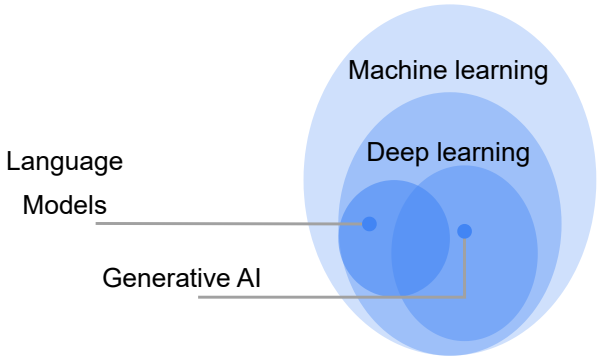


Figure 1.1: Relationships diagram between machine learning, deep learning, language models, and generative AI.

Generative AI language models accomplish this by iteratively seeking the most probable subsequent word, taking into account all the words that came before. This approach to natural language generation is referred to as autoregressive, as it relies on previous tokens to generate text that retains its coherence and flow.
These models harness sophisticated deep-learning neural network structures such as recurrent neural networks (RNNs) or contemporary transformer architectures.

At the heart of the process lies self-supervised training, a method of learning that harnesses the inherent structure of data to create training labels. This approach empowers models to discover and grasp statistical patterns within vast, non-annotated text corpora, thereby reducing the reliance on manually annotated data, which is not only scarce but also comes with a hefty price tag.

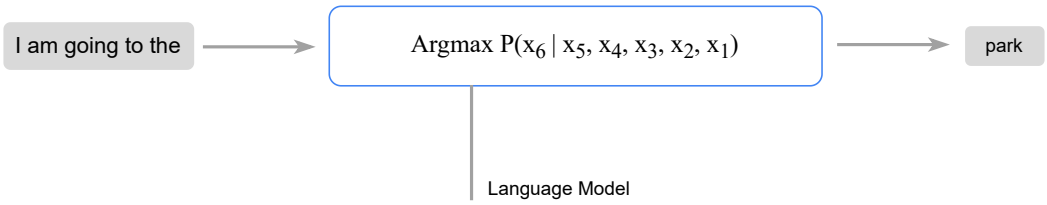


Figure 1.2: Representation of autoregression, iteratively predicting the most probable next word.

A multitude of text datasets, encompassing billions or even trillions of tokens, can be found, originating from sources like books, web pages, code repositories, scientific literature, and more. Within the context of a Language Model (LM), the input text is referred to as a "prompt," and the parameter governing the number of tokens allowed for this input, known as the "context window," typically accommodates several thousand words. However, the specific size of this window can vary from one model to another.

Significant research has demonstrated the profound impact of parameter scaling on the performance of language models (LMs), revealing the existence of scaling laws such as the KM scaling law [**?**] and the Chinchilla scaling law [**?**]. LLMs earn their "Large" designation due to their immense scale, encompassing both model parameters and training data.

As language models expand in dimensions, they unveil new, unanticipated capabilities often referred to as *emergent abilities* [**?**]. These abilities encompass mathematical prowess, intricate reasoning skills, and notably, In-Context Learning (ICL)—the capacity to glean insights from analogies and task examples. Nonetheless, recent research underscores that the emergence of these abilities is often attributed to the selection of inappropriate scales [**?**] and evaluation metrics [**?**]. These remarkable capabilities are not exclusive to larger models, as their performance exhibits a steady enhancement with increasing parameters, becoming apparent primarily when assessed using metrics that introduce non-linear or discontinuous adjustments to the error rate per token of any model. Consequently, it may be erroneous to consider *emergent abilities* as an inherent property of scaling model parameters.

LLMs find application across a multitude of tasks, including but not limited to text generation, summarization, classification, translation, question answering, and named entity recognition. Nevertheless, despite their impressive capabilities, they come with inherent risks and constraints. The primary source of training data is web pages, which contain an abundance of questions followed by answers. While many of these are accurate, they are accompanied by a significant number of inaccuracies and incorrect responses to common-sense queries. Given the vast quantity of this data, researchers have limited control

over the content to which LLMs are exposed during training. Consequently, there may be instances where text generation exhibits undesirable behaviors, such as propagating harmful or biased content or generating hallucinatory outputs that consist of nonexistent or false information.

1.2.1 Transformers

In pursuit of the ambitious goal set by NLP, researchers once turned to RNNs. Nevertheless, these neural networks came with numerous drawbacks, such as their inadequate performance with lengthy text inputs and the sequential nature that impeded their effectiveness. Then, in 2017, Google unveiled Transformers [6], an innovative architectural framework designed to overcome these constraints.

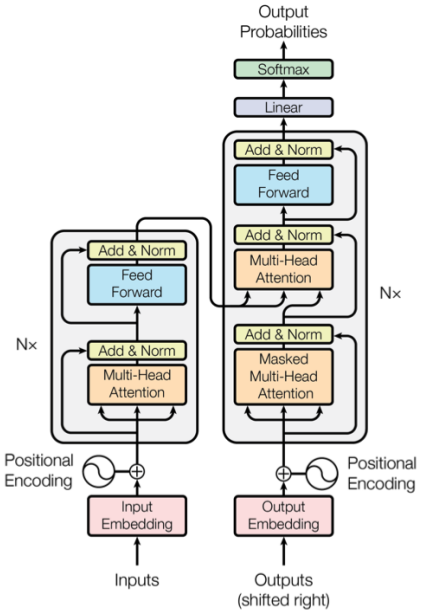


Figure 1.3: Transformers architecture

The central components of the system are the encoder and decoder, which comprise N stacked layers and collaborate closely (as illustrated in figure 1.3). Initially, the input is tokenized and embedded, a parallel process that also necessitates the inclusion of positional embeddings to maintain crucial word order information for effective natural language comprehension. Subsequently, this processed input is fed into the encoder, which transforms the input symbol representations $(x_1, ..., x_n)$ into a sequence of continuous representations denoted as $z = (z_1, ..., z_n)$. These representations then guide the decoder in generating an output sequence $(y_1, ..., y_m)$ in an autoregressive manner.

Each encoding layer encompasses two sub-layers: one dedicated to multi-head self-attention and another for a feed-forward network. Simultaneously, the decoder features an additional multi-head attention layer for conducting cross-attention with the output from the encoder stack.

The title of the paper, "Attention is all you need," succinctly conveys the core idea behind the architecture, which revolves around the concept of attention. This attention mechanism can be mathematically expressed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)$$

In this equation, Q corresponds to the input token, serving as a descriptor of the information being sought. Meanwhile, K refers to the keys, which are features used for comparison against other elements. Lastly, V represents the values, with all three of these elements—Q, K, and V—being represented as matrices. The resulting output is a weighted sum of the values, determined by the attention scores. Each attention score reflects the degree of relevance or similarity between a given query and key pair.

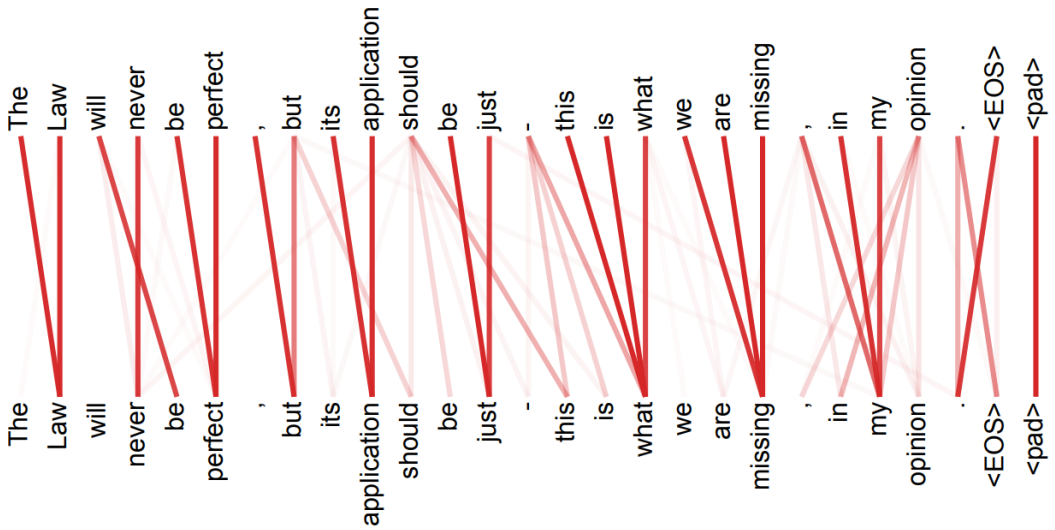


Figure 1.4: Attention map, an illustration of weights between each word and every other word. The color intensity indicates the relationship weight.

Utilizing multi-head attention, the model gains the ability to grasp a wide array of patterns and relationships within the input data. Each head within this architecture focuses on distinct aspects, resulting in three key mechanisms:

1. *Cross-attention*: In this mode, queries originate from prior decoder layers, while memory keys and values are drawn from the encoder's output. This

enables the model to extract relevant information from the encoder's output while generating the target sequence.

2. Encoder *self-attention*: At each layer, all keys, values, and queries are derived from the preceding output, allowing every position to attend to all positions in the previous encoder layer.

3. Decoder *self-attention*: This mechanism empowers each position in the decoder to attend to all positions, encompassing its own position as well as those that precede it.

The utilization of self-attention within the transformer framework enables it to grasp the significance and nuances of every word within a sentence. This capacity to discern intricate connections throughout the entire input greatly enhances language comprehension, propelling transformers to bridge a substantial divide with their predecessors.
The attention weights, initially set at random, undergo refinement through the course of model training.

1.2.2 Pre-training and Fine-tuning

The model's architecture begins with initializing parameters as random weights, necessitating a training phase for acquiring a grasp of natural language. During inference, the initial token serves as input, progressing through both the encoder and decoder, which sequentially produces a sentence. Consequently, the keys and values within the decoder's cross-attention layer, inherited from the encoder, remain constant. However, when it comes to training, the decoder's input varies and is contingent on the specific task at hand.

To minimize the necessity of initiating the process anew every time to grasp the language's semantics, GPT[**?**] and Bert[**?**] introduce a two-stage development approach aimed at reducing the resource and cost burdens of training.

Pre-training This initial phase of training is conducted with an open-ended approach, demanding significant computational resources and incurring high costs. Its primary purpose is to imbue the model with a deep understanding of language semantics, grammar, and word meanings, equipping it with the capability to manipulate language effectively. Pre-training encompasses a spectrum of self-supervised tasks, including:

1. Concealing a token within a sentence during decoding, necessitating the model's ability to discern the correct word to insert based on context.

2. Permuting sentence tokens and challenging the model to reconstruct the original sequence accurately.

3. Evaluating whether one sentence can logically follow another when presented with two sentences.

Crucially, these tasks are executed under the constraint that the decoding attention can solely consider previously generated tokens, with any subsequent tokens being concealed to maintain the autoregressive nature of the process.

Pre-training relies on an extensive dataset, typically comprising trillions of unlabeled text tokens, and the quality and diversity of this data significantly impact the ultimate performance of the model.

Fine-tuning Fine-tuning involves the process of training a pre-existing model for specific tasks, and it allows for multiple rounds of fine-tuning, resulting in diverse models. The training data typically includes input-output pairs for tasks like summarization and QA, requiring a smaller dataset than the initial pre-training phase, which makes it a cost-effective form of supervised training. Furthermore, fine-tuning can be employed to enable a pre-trained model to gain expertise in specialized domains such as biomedicine or law.

Incorporating Curriculum Learning (ICL) can be particularly valuable when instructing a Language Model with limited performance improvements without altering the model's parameters. This approach proves essential when direct access to the model's architecture is unavailable, and only inference can be performed.

The feature-based approach, on the other hand, centers on embedding generated by the pre-trained model and utilizes them as input features for training a classification model.

In contrast to the aforementioned methods, genuine fine-tuning entails updates to certain aspects of the model, including:

1. Augmenting output layers (Fine-tuning i): During this phase, the parameters of the pre-trained model remain static, and only newly introduced output layers undergo training.

2. Updating all layers (Fine-tuning ii): Although fine-tuning demands fewer resources compared to pre-training, the larger the model, the more resource-intensive it becomes to update all its layers.

3. Parameter-efficient fine-tuning (PEFT): This approach employs various parameter-efficient techniques that train only a limited number of parameters while achieving performance levels comparable to full fine-tuning.

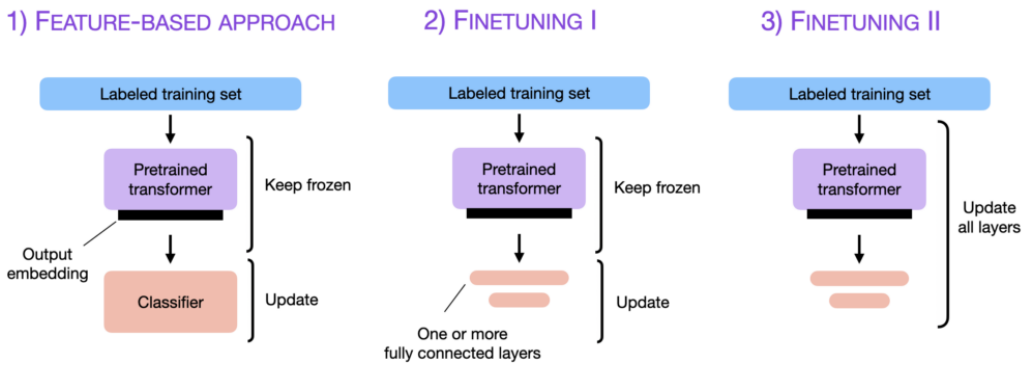


Figure 1.5: Illustration of different trained layer in feature-based approach, finetuning i and finetuning ii.

1.2.3 Encoder-Decoder, Encoder-only, Decoder-only

The emergence of self-attention transformer models has led to the development of three unique architectural setups that hold significant importance in a wide array of language comprehension and generation endeavors. These setups encompass encoder-exclusive, encoder-decoder, and decoder-centric models, all stemming from the fundamental transformer framework.

Encoder-only This particular model processes input text sequences and produces a fixed-length vector as an output, which serves as a representation of the text, also known as an embedding. This vectorization can take the form of a single vector for the entire sequence or distinct vectors for individual words. These vectors capture the semantic essence of the sequence, making them valuable for various downstream applications that revolve around contextual understanding and tasks like sentiment analysis or named entity recognition. Often, fine-tuning is applied to tailor the model to specific tasks. Unlike a text generation model, an encoder-only model specializes in comprehending text through bidirectional attention, enabling each token to attentively consider all other tokens within the input sentence.

Encoder-decoder The model in question, often referred to as the sequence-to-sequence (seq2seq) model, represents the foundational transformer architecture. It consists of two vital components: the encoder, responsible for processing an input sequence and producing an embedding, and the decoder, which generates an output sequence token by token. This architecture primarily excels at tasks such as text translation and summarization. Notable models that adhere to this paradigm include T5 and BART. A key distinguishing feature of encoder-decoder models, in contrast to encoder-only or decoder-only variants, is their

utilization of cross-attention mechanisms to blend information from both the source and target sequences.

Decoder-only Numerous contemporary language models (LLMs) employ a decoder-exclusive framework when it comes to text generation. This approach, akin to the encoder-decoder paradigm albeit lacking the initial component, is characterized by its simplicity and reduced parameter count. These models follow a straightforward procedure, accepting an input sequence and iteratively predicting the subsequent most likely token in an autoregressive fashion until an ending token is generated.

1.2.4 Prompt Engineering

Interacting with an LLM involves instructing it using text prefixes, such as "*Please translate the following text into Greek:.*" The model comprehends the task solely through natural language instructions and proceeds to generate the corresponding Greek sentence. Various methods exist for instructing an LLM:

1. *Zero-shot*: Structured prompts define both the instructions and the data.

2. *One-shot*: In addition to task descriptions and data, the prompt begins with a single task example and the correct answer.

3. *Few-shot*: Similar to one-shot, but with a limited number of task examples.

4. *Chain-of-thought (CoT)*: Elevates the model's complex reasoning capabilities by attempting to generate intermediate reasoning steps (e.g., "*Let's break it down step by step*"). Combining CoT with few-shot prompting can yield improved results.

Figure 1.6: Example of zero-shot, one-shot, and few-shot prompting to solve a sentiment analysis task.

Prompting is a widely employed technique for tackling numerous challenges (see figure x). However, maintaining control over the resultant output can be challenging due to the absence of a clear distinction between directives and input. To tackle this issue, many Language Models (LLMs) undergo fine-tuning using a prompt template, which delineates distinct sections as follows:

1. *System*: employed to convey a general directive or context to the LLM, aiding in the clarification of subsequent input.

2. *Directive / User Input*: utilized to specify a series of precise instructions that the LLM must adhere to, along with the input data.

3. *Assistant*: subsequent to this section, the LLM generates the textual output.

Figure 1.7: In LLaMA2-chat is possible to indicate sections among different tags, "[SYS] text [/SYS]" for system prompt and "[INST] text [/INST]" for instruction prompt.

Strategic manipulation of prompts taps into an LLM's inherent ICL capacity, while an alternative approach involves fine-tuning instructions, granting greater autonomy in tailoring text generation to specific objectives.

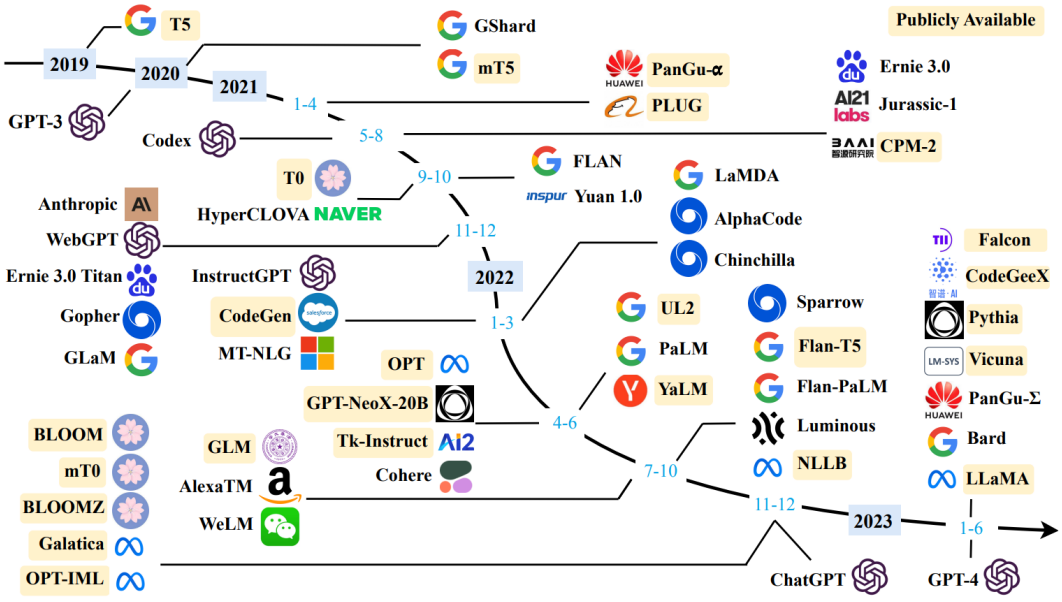


Figure 1.8: A timeline of existing large language models in recent years. LLMs yellow marked have publicly available model checkpoints and are reported only, which include publicly reported evaluation results.

From [7]

1.3 Knowledge Distillation

KD revolves around the transmission of knowledge from larger teacher networks to their smaller student counterparts [8, 9]. Focusing on the concept of output emulation, the student has the flexibility to adopt a distinct architecture compared to the teacher. Some researchers train students to mimic the teacher's predictions at the sequence level. This is achieved by assuming access

to token probability distributions and aligning with the top-scored outcome from the teacher's beam search [10, 11]. Due to the limitation of a small pool of training samples in fully expressing the teacher model's knowledge, data augmentation strategies become prominent [12]. In a manner similar to ours, recent publications do not directly mimic teacher probabilities. Instead, they embrace a "machine-to-corpus-to-machine" paradigm by training on substantial amounts of LLM-generated annotations [13, 14, 15, 16, 17, 18]. While the predominant focus of the literature is on question-answering tasks, the application of LLM-KD to Named Entity Recognition and Classification (NERC) has recently been explored only by [2] in traditional instructional-tuning scenarios. Additionally, it's worth highlighting that many of the distilled models from previous efforts still maintain a substantial billion-scale size and are computationally demanding. As far as we know, we are the first to investigate the effects of LLM distillation in NERC using limited computational resources.

Chapter 2

Related work

2.1 ICD-9-CM Datasets

ICD-9-CM (International Classification of Diseases, 9th Revision, Clinical Modification)[1] datasets play a crucial role in the healthcare industry by providing a standardized system for coding and classifying medical diagnoses and procedures. These datasets have been widely used for various purposes, including medical billing, research, and healthcare management.

Healthcare institutions and researchers often rely on ICD-9-CM datasets to analyze health trends, assess the effectiveness of treatments, and generate statistics on various medical conditions. However, it's important to note that ICD-9-CM has been largely replaced by ICD-10-CM (and subsequently ICD-11-CM), which offers more detailed and up-to-date coding systems to reflect advances in medical knowledge and technology.

In recent years, efforts have been made to digitize and make ICD-9-CM datasets more accessible through electronic health record systems and healthcare databases. This digital transformation has facilitated more efficient data management and analysis, benefiting both healthcare professionals and researchers in the field.

Despite these efforts, the field is still lacking great open-source datasets in BIO format ready to be trained on. This is where Knowledge Distillation (KD) comes in, because with only a dataset containing medical reports and document-level ICD-9 codes you can get to the final product of a Small Language Model perfectly capable of performing NERC.

[1] `https://www.nber.org/research/data/icd-9-cm-diagnosis-and-procedure-codes`

CODER [19] CODER (Cross-lingual knowledge-infused medical term embedding for term normalization) is a cross-lingual medical term embedding that is trained on a medical knowledge graph (KG) named the Unified Medical Language System (UMLS). CODER is designed for medical term normalization by providing close vector representations for different terms that represent the same or similar medical concepts, with support for multiple languages.

CODER is trained using contrastive learning, which is a type of machine learning that learns by comparing positive and negative examples. In the case of CODER, the positive examples are pairs of terms that represent the same or similar medical concepts, and the negative examples are pairs of terms that represent different medical concepts.

CODER embeddings have been shown to outperform various state-of-the-art biomedical word embeddings, concept embeddings, and contextual embeddings on a variety of tasks, including zero-shot term normalization, semantic similarity measurement, and concept relation classification.

SparkNLP [20] SparkNLP is a state-of-the-art open-source natural language processing (NLP) library for Apache Spark. It provides a wide range of NLP features, including text classification, named entity recognition, question answering, and machine translation. They trained health-related NER and classification models on different datasets published within the Social Media Mining for Health Applications (#SMM4H 2022) workshop.

These datasets are closed-source and they don't represent real-world scenarios at all because the text is perfectly clean, anonymous, brief, and also school quiz-like. Our intention is to build a NERC system that actually works on complex, diverse, and "dirty" reports provided by hospitals from all over the world.

MIMIC-III [21] MIMIC-III (Medical Information Mart for Intensive Care III) is a large, freely available database comprising de-identified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012.

The database includes information such as demographics, vital sign measurements made at the bedside (1 data point per hour), laboratory test results, procedures, medications, caregiver notes, imaging reports, and mortality (both in and out of the hospital).

MIMIC supports a diverse range of analytic studies spanning epidemiology, clinical decision-rule improvement, and electronic tool development. It is notable for four factors:

- it is freely available to researchers worldwide
- it encompasses a diverse and very large population of ICU patients
- it contains high temporal resolution data including lab results, electronic documentation, and bedside monitor trends and waveforms.
- it contains document-level ICD-9-CM classification codes

2.2 Llama 2

[2]

- Number of parameters: 7B, 13B, 70B
- Model Type: Encoder-Decoder
- Pre-training Data and Tasks:

    1. Overview:

       Llama 2 was pre-trained on 2 trillion tokens of data from publicly available sources. The fine-tuning data includes publicly available instruction datasets, as well as over one million new human-annotated examples. Neither the pre-training nor the fine-tuning datasets include Meta user data.

    2. Data Freshness:

       The pre-training data has a cutoff of September 2022, but some tuning data is more recent, up to July 2023.

- Huggingface Description:

  Meta developed and publicly released the Llama 2 family of large language models (LLMs), a collection of pre-trained and fine-tuned generative text models ranging in scale from 7 billion to 70 billion parameters. Our fine-tuned LLMs, called Llama-2-Chat, are optimized for dialogue use cases. Llama-2-Chat models outperform open-source chat models on most

[2] `https://huggingface.co/meta-llama/Llama-2-70b-chat-hf`

benchmarks we tested, and in our human evaluations for helpfulness and safety, are on par with some popular closed-source models like ChatGPT and PaLM.

2.3 GPT-3.5-turbo

[3]

- Number of parameters: 175B
- Model Type: Encoder-Decoder
- Pre-training Data and Tasks:
  1. Overview:

     GPT-3.5-turbo was trained on a massive dataset of text and code, including books, Wikipedia, code repositories, articles, chat logs, and social media posts. The specific datasets used to train GPT-3.5-turbo are not publicly disclosed, but it is known that the dataset is over 500 GB in size and contains over 100 billion words.
  2. Data Freshness:

     The data freshness of the pre-training data of GPT-3.5-turbo is September 2021.
- Huggingface Description:

  GPT-3.5 is a set of models that improve on GPT-3 and can understand as well as generate natural language or code. It is an autoregressive language model (LLM) from OpenAI that uses deep learning to produce human-like text. It is a fine-tuned version of GPT-3, the third-generation language prediction model in the GPT series created by OpenAI. GPT-3.5 has garnered significant attention and acclaim for its unparalleled ability to understand and generate human-like text. With an astounding 175 billion parameters at its disposal, GPT-3.5 stands as one of the most expansive and powerful language models ever constructed at the time of its release.

  GPT-3.5's exceptional performance is not only limited to its sheer size but also stems from its highly refined architecture. Harnessing the power

[3] `https://platform.openai.com/docs/models/gpt-3-5`

of deep learning, GPT-3.5 delivers consistently accurate and relevant results, elevating the standard for language models and establishing itself as a trailblazer in the field of artificial intelligence.

2.4 BioMedLM

[4]

- Number of parameters: 2.7B
- Model Type: Decoder-only
- Pre-training Data and Tasks:
  1. Overview:

     BioMedLM is a large-scale language model pre-trained on a vast corpus of biomedical texts. It has been fine-tuned to understand and generate text related to various biomedical and clinical tasks.
  2. Data Freshness:

     BioMedLM's data cutoff is after December 31, 2020, because it was trained on The Pile dataset, which was released on that date.
- Huggingface Description:

  BioMedLM 2.7B is a new language model trained exclusively on biomedical abstracts and papers from The Pile. This GPT-style model can achieve strong results on a variety of biomedical NLP tasks, including a new state-of-the-art performance of 50.3% accuracy on the MedQA biomedical question-answering task.

2.5 PMC-LLaMa

[5]

- Number of parameters: 7B, 13B
- Model Type: Encoder-Decoder

[4] https://huggingface.co/stanford-crfm/BioMedLM
[5] https://huggingface.co/chaoyi-wu/PMC_LLAMA_7B

- Pre-training Data and Tasks:

    1. Overview:

       PMC-LLaMA is trained on a massive dataset of medical text, including biomedical academic papers and medical textbooks. It is also fine-tuned on a dataset of medical question-answering pairs, rationale for reasoning, and conversational dialogues.

    2. Data Freshness:

       PMC-LLaMA was initially trained on a dataset of biomedical academic papers published up to 2022. It was then fine-tuned on a dataset of medical question-answering pairs, rationale for reasoning, and conversational dialogues collected in 2023.

- Huggingface Description:

  PMC-LLaMA is a large language model specifically designed for medical applications. It is a lightweight model with only 13 billion parameters, but it has been shown to outperform other LLMs on various public medical question-answering benchmarks, including ChatGPT.

2.6 MedLLaMA2

[6]

- Number of parameters: 7B
- Model Type: Encoder-Decoder
- Pre-training Data and Tasks:

    1. Overview:

       MedLLaMA2 was trained on the MEDQA dataset: a large-scale open domain question answering (OpenQA) dataset for solving medical problems, collected from the professional medical board exams. It covers three languages: English, simplified Chinese, and traditional Chinese, and contains 12,723, 34,251, and 14,123 questions for the three languages, respectively.

[6] `https://huggingface.co/llSourcell/medllama2_7b`

2. Data Freshness:

   MedLLaMA2's data cutoff is after August 4, 2020, because it was trained on The MEDQA dataset, which was released on that date.

- Huggingface Description:

  MedLLaMA2 is a large language model (LLM) trained on a massive dataset of medical text and code. It is a fine-tuned version of the Llama 2 model, which is a general-purpose LLM trained on a massive dataset of text and code from the real world.

2.7 GPT-NeoX

[7]

- Number of parameters: 20B
- Model Type: Decoder-only
- Pre-training Data and Tasks:

  1. Overview:

     GPT-NeoX was trained on a massive dataset of text and code called the Pile. The Pile is an 800GB dataset that contains a wide variety of text formats, including books, articles, code, and social media posts. It also includes text from different languages and dialects.

  2. Data Freshness:

     GPT-NeoX's data cutoff is after December 31, 2020, because it was trained on The Pile dataset, which was released on that date.

- Huggingface Description:

  GPT-NeoX is, to the best of our knowledge, the largest dense autoregressive model that has publicly available weights at the time of submission.

[7] `https://huggingface.co/llSourcell/medllama2_7b`

Chapter 3

Method

3.1 ICD-Juicer

In this chapter, we present our solution to NERC in specific domains with many different entity types using knowledge distillation.

3.2 Finding a Dataset

The first step to any LLM-related task is to find, or create, as we will see in a bit, the dataset on which to fine-tune the LLM. In our case, after months worth of searching for an ICD-9-annotated dataset, we got our hands on the only dataset that contained at least document-level ICD-9 codes: MIMIC-III.

3.3 Choosing an LLM

There is no absolutely correct answer to this question because while the obvious answer would be the biggest LLM available out there, you also need to take into consideration the existence of other smaller LLMs that despite the smaller size leverage a training dataset more useful to the task you are going to ask the LLM to perform.

In our situation, we needed an LLM capable of associating document-level ICD-9 codes to the specific parts of the document where the ICD-9 codes have been recognized. We didn't need the LLM to know the ICD-9 codes so well because we thought of providing the LLM with a short description (almost like a label) of every ICD-9 code present in the document. This way we avoided hallucinations or simply wrong associations due to the lack of knowledge of specific ICD-9 codes by the LLM. We're basically bridging the gap between the LLM and the NERC task by removing any possible distraction such as numbers.

After trying some medical domain LLMs without success, i.e. BioMedLM, PMC-LLaMa, MedLLaMA2, and GPT-NeoX, we got to try Llama2 first, and GPT-3.5-turbo as the final choice.

Llama2 The advantage of using Llama2 is that it is fully open source and you can easily run it from huggingface.co. The downside is that, since it isn't explicitly trained on medical text, you would need to run the 70B model to get results that are somewhat workable. Due to hardware limitations, we were constrained to use the 13B parameters version of Llama2, which unfortunately was not our final bet. It's worth noting that even if we didn't manage to create the dataset using Llama2, extensive prompt engineering has been done with the 13B model.

GPT-3.5-turbo It's inevitable that by perfecting the prompt structure and wording to extract as much clean output as possible from the 13B model the prompt was basically perfect when we switched to using GPT-3.5-turbo through the OpenAI APIs. The output was almost always compliant with our output format.

3.4 Prompt Engineering

The prompt engineering required for the creation of the dataset has been one of the key steps in the whole work. We started with Llama2 and switched to GPT-3.5-turbo almost at the end of the prompt engineering phase. What we wanted to get in response is a list of Python tuples like [('entity', 'entity type'), ...].

We first experimented with a simple prompt explaining the task, providing an output format to follow and the medical report on which to perform NERC. The output format was the most difficult to get right for the LLM, so the replies were not very useful because since they weren't consistent at all we couldn't even extract the expected output from the answer.

We then started improving the system prompt to emphasize the output format. Because if we didn't get consistent replies from the LLM we couldn't even think about starting with the creation of the dataset.

After studying a bit more of the MIMIC-III dataset we ended up finding the document-level ICD-9 codes in a handy table in the dataset. We then waited no time to try and give these document-level ICD-9 codes in the prompt.

This was an essential checkpoint in this step because, even if the output wasn't consistent yet, the answers were starting to look like actual NERC; where the text in the left side element of the tuple was actually taken from the text of the medical document and the right side element of the tuple was one of the given ICD-9 codes.

There was still something wrong because even if we took the best-looking answers, too few were outputs with correct associations. What helped the LLM make the associations more easily was not giving the ICD-9 codes in the prompt but, instead, the ICD-9 codes short descriptions provided in the SHORT_TITLE column in the MIMIC-III dataset.

We deduct that the lack of ICD-9 codes in the pre-training data of Llama2, but also numbers in general, were the cause of the wrong associations.

After this glow-up that the prompt had, we tried giving a few examples of the task in the prompt, but that increased repetition of the prompt tokens so the only changes that we made were cleaner reports and a clearer instruction.

3.4.1 Prompt Evolution

TODO: add refs

Table 3.1: Evolution of Prompts and System prompts 1

System Prompt	**Prompt**
You are an assistant. Always answer as concisely as possible. Don't answer with superfluous information. Always pay attention to the requested output format.	Given a medical report and the ICD-9 codes that were detected in it, extract the diagnoses and their corresponding ICD-9 codes. Medical Report: {report_text} Detected ICD-9 Codes: {detected_icd9_codes} Output Format: $[(\text{"}Diagnosis\text{"}, \text{"}ICD-9Code\text{"}), ...]$
You are a trained coder. Always output following the Output Format. Always specify the report HADM_ID at the beginning of the output.	Given a medical report and the ICD-9 codes that were detected in it, extract the diagnoses and their corresponding ICD-9 codes. Medical Report: {report_text} Detected ICD-9 Codes: {detected_icd9_codes} Output Format: $[(\text{"}Diagnosisinthereport\text{"}, \text{"}providedICD-9Code\text{"}), ...]$

Table 3.2: Evolution of Prompts and System prompts 2

System Prompt	**Prompt**
You are a trained coder. Always output following the Output Format. Always specify the report HADM_ID at the beginning of the output.	Given a medical report and the ICD-9 codes that were detected in it, answer with the diagnoses and their corresponding ICD-9 codes following the output format specified. Medical Report: {report_text} Detected ICD-9 Codes: {detected_icd9_codes} Output Format: $[(\text{"}Diagnosisinthereport\text{"}, \text{"}providedICD-9Code\text{"}), ...]$
You are a trained coder. Always output following the Output Format.	Given a medical report and the ICD-9 codes that were detected in it, answer with the diagnoses and their corresponding ICD-9 codes following the output format specified. Medical Report: {report_text} Detected ICD-9 Codes: {detected_icd9_codes} Output Format: $[(\text{"}Diagnosisinthereport\text{"}, \text{"}providedICD-9Code\text{"}), ...]$

Table 3.3: Evolution of Prompts and System prompts 3

System Prompt	**Prompt**
You are a trained coder. Always output following the Output Format. Don't output anything else that is not in the output format.	### Instruction: Given a medical report and the ICD-9 codes that were detected in it, answer with the diagnoses and their corresponding ICD-9 codes. Return the output in the following format: [("*Diagnosisinthereport*","*providedICD*−9*Code*"), ...] ### Medical Report: Report text ### Detected ICD-9 Codes: 4254, Other primary cardiomyopathies 2762, Acidosis ... ### Diagnosis and ICD-9 Codes: [("*Pulmonarytuberculosis*","01193"),("*Acidosis*","2762"), ...)] ### Medical Report: {report_text} ### Detected ICD-9 Codes: {detected_icd9_codes} ### Diagnosis and ICD-9 Codes:

Table 3.4: Evolution of Prompts and System prompts 4

System Prompt	**Prompt**
You are a trained coder. You must precisely follow every instruction. You only output what is requested and strictly following the output format. When you answer don't include any text that doesn't follow the output format. The following is the only output you can produce: [("*diagnosis*","*short description of ICD−9 code*"),...]	### Instruction: Given a medical report and the short descriptions of the ICD-9 codes that were detected in it, answer with the diagnoses and their corresponding ICD-9 codes' short descriptions. If one of the provided ICD-9 code's short descriptions doesn't match any part of the medical report don't include it in the output. Use the following output format: [("*diagnosis*","*shortdescriptionthatmatchesthediagnosis*"),...] ### Medical Report: Report text ### Short descriptions of the detected ICD-9 codes: Prim cardiomyopathy NEC Acid bas ... ### Diagnoses and short descriptions of the ICD-9 codes: [("*tuberculosis*","*PulmonTBNOS−microdx*"),("*pleuraleff*...)] ### Medical Report: {report_text} ### Short descriptions of the detected ICD-9 codes: {detected_icd9_codes} ### Diagnoses and short descriptions of the ICD-9 codes:

Table 3.5: Evolution of Prompts and System prompts 5

System Prompt	**Prompt**
You are a helpful, respectful and honest assistant expert in medicine which is supposed to help doctors in the extraction of named entities from medical reports. Always answer as helpfully as possible using the medical report provided. Your answers should only answer the question once and not have any text after the answer is done. The only expected output is an array of tuples as in the following example: $[(entity_1, Label_1),$ $(entity_2, Label_2), \dots]$ Please do not return any output that is different from the expected format.	### Instruction: Within the given medical reports identify spans of text which can be classified as one of the given candidate labels. ### Medical Report: Report text ### Candidate labels: Prim cardiomyopathy NEC Acidosis ... ### Named entities: $[(\text{"}tuberculosis\text{"}, \text{"}PulmonTBNOS-microdx\text{"}), (\text{"}pleuraleff...)$] ### Medical Report: Report text ### Candidate labels: CHF NOS Atrial fibrillation ... ### Named entities: $[(\text{"}prostatecancer\text{"}, \text{"}Malignneoplprostate\text{"}), (\text{"}gastroin...)$] ### Medical Report: {report_text} ### Candidate labels: {detected_icd9_codes} ### Named entities:

Table 3.6: Evolution of Prompts and System prompts 6

Final System Prompt	**Final Prompt**
You are an information extraction system. Your answers should only answer the question once and not have any text after the answer is done. You can only output a list of tuples of the following format: [('entity 1', 'type of entity 1'), ('entity 2', 'type of entity 2'), ...] Please don't output any text before and after the brackets []. If you don't know the answer, please output an empty list.	### Instruction: Based on the given context, your task is to extract the entities that can be classified as one of the given types. The response must be in a list of tuples of the following format: $[('entity1','typeofentity1'),$ $('entity2','typeofentity2'),...]$ ### types: {report['icd9_codes']} ### Context: "' {report['text']} "' ### Response:

3.5 Annotating the Dataset

At this point, we have a dataset, that includes medical reports together with document-level ICD-9 codes, and we have an LLM, GPT-3.5-turbo. To annotate the dataset we used the word level BIO-format, producing the dataset in multiple checkpoints. This way we could check how the model was performing after each fine-tuning session and also prevent some money from being wasted on OpenAI API calls in the unfortunate case of something in the annotation phase going wrong. Right before the output of GPT-3.5-turbo is used to annotate the current report, we check for each entity to be contained in the report text and for each entity type to be one of the given ones.

```
def process_output(output, report):
    report_tokens = word_tokenize(report['text'])
    tags = ['O'] * len(report_tokens)
    text = report['text'].lower()
    possible_codes = [code.lower() for code in report['icd9_codes']]
    output = [el for el in output if len(el) == 2]
    for entity, label in output:

        if entity.lower() in text and label.lower() in possible_codes:
            actual_label = report['icd9_codes'][possible_codes.index(
                label.lower())]
            entity_tokens = word_tokenize(entity)
            entity_length = len(entity_tokens)
            for i in range(0, len(report_tokens)):
                if report_tokens[i:i+entity_length] == entity_tokens:
                    begin = False
                    for c in range(i, i+entity_length):
                        if begin:
                            tags[c] = f"I-{actual_label}"
                        else:
                            tags[c] = f"B-{actual_label}"
                            begin = True

    count_tags = [1 if tag != 'O' else 0 for tag in tags]
    if sum(count_tags) > 0:
        with open('outputs/mimiciii-24000-33999.ner', 'a') as f:
            for i, (token, tag) in enumerate(zip(report_tokens, tags)):
                f.write(f"{token}\t{tag}\n")
            f.write("\n")
```

Figure 3.1: Function that processes each output and annotates the report

3.6 Teaching the Student

As SLMs (Small Language Models) we chose two BERT-based models, in particular: BERT-base-uncased and BERT-large-uncased. We fine-tuned both models on the annotated dataset in BIO format.

```python
from transformers import EarlyStoppingCallback

training_args = TrainingArguments(
    output_dir="bert_large_mimic_iii_token_classification_20ep",
    learning_rate=2e-5,

    per_device_train_batch_size=4,
    per_device_eval_batch_size=16,
    num_train_epochs=20,
    warmup_ratio=0.3,
    weight_decay=0.01,

    report_to = 'wandb',
    evaluation_strategy="epoch",
    save_strategy="epoch",
    metric_for_best_model="f1_macro",
    load_best_model_at_end=True

)

trainer = Trainer(
    model=model,
    args=training_args,

    train_dataset=tokenized_ds_train,
    eval_dataset=tokenized_ds_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics

)

trainer.train()

wandb.finish()
```

Figure 3.2: BERT-large training code snippet

```
from transformers import EarlyStoppingCallback

training_args = TrainingArguments(

    output_dir="bert_base_mimic_iii_token_classification_20ep",
    learning_rate=2e-4,
    per_device_train_batch_size=16,

    per_device_eval_batch_size=16,
    num_train_epochs=20,
    warmup_ratio=0.3,

    weight_decay=0.01,
    report_to = 'wandb',
    evaluation_strategy="epoch",

    save_strategy="epoch",
    metric_for_best_model="f1_macro",
    load_best_model_at_end=True

)

trainer = Trainer(

    model=model,
    args=training_args,
    train_dataset=tokenized_ds_train,

    eval_dataset=tokenized_ds_test,
    tokenizer=tokenizer,
    data_collator=data_collator,

    compute_metrics=compute_metrics
)


trainer.train()

wandb.finish()
```

Figure 3.3: BERT-base training code snippet

Chapter 4

Experiments

4.1 Experimental Setup

4.1.1 Hardware Configuration

The hardware configuration for the fine-tuning of the model was the one offered by the Google Colab Pro plan:

- GPUs: K80, P100, T4
- CPUs: 2 x vCPU
- RAM: 32GB

4.1.2 Dataset

```
# remove rows with reports over 1000 characters
df_reports = df_reports[df_reports['TEXT'].str.len() < 1000]
# take reports from index 24000 to 33999
df_reports = df_reports.iloc[24000:34000]
```

Figure 4.1: Last batch dataset filtering code snippet

The portion of the dataset used is the first 34,000 reports under 1000 characters long. This was to easily stay within the context length of GPT-3.5-turbo of 4K tokens and also to reduce the cost per report processed. Considering the scale of the dataset, over 2 million reports, it was a no-brainer to take 10,000 reports under 1000 characters long starting from the end of the dataset.

4.1.3 Metrics

In machine learning and statistics, metrics are essential for evaluating model performance. This section covers four key metrics - precision, recall, and F1-score - commonly used in classification tasks to assess model effectiveness.

- **Precision**: Precision is a measure of how many of the positive predictions made are correct (true positives).
- **Recall**: Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data.
- **F1-score**: F1-Score is a measure combining both precision and recall. It is generally described as the harmonic mean of the two. Harmonic mean is just another way to calculate an "average" of values, generally described as more suitable for ratios (such as precision and recall) than the traditional arithmetic mean.

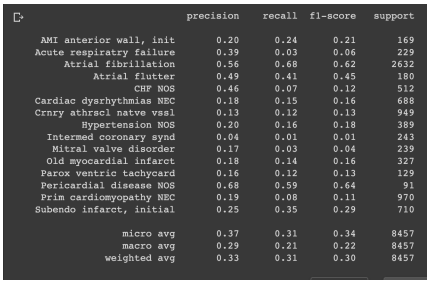
4.2 Results

Epoch	Training Loss	Validation Loss	Precision Macro	Recall Macro	F1 Macro
1	0.881400	0.760201	0.567536	0.094826	0.040408
2	0.708300	0.656689	0.555259	0.102404	0.062354
3	0.670700	0.636042	0.386062	0.124321	0.116729
4	0.656100	0.627453	0.342375	0.144892	0.142574
5	0.644300	0.625074	0.308239	0.140076	0.142876
6	0.626900	0.645700	0.323913	0.162862	0.156735
7	0.548900	0.668634	0.319862	0.138111	0.139886
8	0.499200	0.715115	0.201067	0.169865	0.159928
9	0.477800	0.737832	0.199532	0.185220	0.175718
10	0.442300	0.767590	0.195624	0.167610	0.167275
11	0.396800	0.817736	0.175486	0.179628	0.173256
12	0.350500	0.840572	0.182302	0.178217	0.175916
13	0.323900	0.874679	0.170803	0.182649	0.172558
14	0.314100	0.897720	0.180679	0.170094	0.170017

Figure 4.2: BERT-base training metrics log

	precision	recall	f1-score	support
AMI anterior wall, init	0.11	0.18	0.14	49
Acute respiratry failure	0.18	0.02	0.04	82
Atrial fibrillation	0.52	0.70	0.60	1136
Atrial flutter	0.46	0.43	0.44	84
CHF NOS	0.29	0.06	0.11	186
Cardiac dysrhythmias NEC	0.09	0.10	0.09	261
Crnry athrscl natve vssl	0.08	0.10	0.09	329
Hypertension NOS	0.15	0.14	0.14	143
Intermed coronary synd	0.00	0.00	0.00	91
Mitral valve disorder	0.11	0.02	0.04	83
Old myocardial infarct	0.09	0.09	0.09	100
Parox ventric tachycard	0.09	0.11	0.10	47
Pericardial disease NOS	0.61	0.53	0.57	47
Prim cardiomyopathy NEC	0.07	0.05	0.06	320
Subendo infarct, initial	0.14	0.25	0.18	237
micro avg	0.31	0.33	0.32	3195
macro avg	0.20	0.19	0.18	3195
weighted avg	0.28	0.33	0.29	3195

Figure 4.3: BERT-base top 15 most frequent labels metrics - span-level



	precision	recall	f1-score	support
AMI anterior wall, init	0.20	0.24	0.21	169
Acute respiratry failure	0.39	0.03	0.06	229
Atrial fibrillation	0.56	0.68	0.62	2632
Atrial flutter	0.49	0.41	0.45	180
CHF NOS	0.46	0.07	0.12	512
Cardiac dysrhythmias NEC	0.18	0.15	0.16	688
Crnry athrscl natve vssl	0.13	0.12	0.13	949
Hypertension NOS	0.20	0.16	0.18	389
Intermed coronary synd	0.04	0.01	0.01	243
Mitral valve disorder	0.17	0.03	0.04	239
Old myocardial infarct	0.18	0.14	0.16	327
Parox ventric tachycard	0.16	0.12	0.13	129
Pericardial disease NOS	0.68	0.59	0.64	91
Prim cardiomyopathy NEC	0.19	0.08	0.11	970
Subendo infarct, initial	0.25	0.35	0.29	710
micro avg	0.37	0.31	0.34	8457
macro avg	0.29	0.21	0.22	8457
weighted avg	0.33	0.31	0.30	8457

Figure 4.4: BERT-base top 15 most frequent labels metrics - token-level

	precision	recall	f1-score	support
Acute respiratry failure	0.12	0.07	0.09	80
Atrial fibrillation	0.57	0.65	0.61	1229
CHF NOS	0.08	0.05	0.06	187
Cardiac dysrhythmias NEC	0.12	0.14	0.13	278
Crnry athrscl natve vssl	0.11	0.10	0.11	299
Hypertension NOS	0.18	0.11	0.14	164
Intermed coronary synd	0.03	0.03	0.03	87
Old myocardial infarct	0.11	0.17	0.14	108
Prim cardiomyopathy NEC	0.08	0.08	0.08	243
Subendo infarct, initial	0.21	0.28	0.24	214
micro avg	0.33	0.35	0.34	2889
macro avg	0.16	0.17	0.16	2889
weighted avg	0.31	0.35	0.33	2889

Figure 4.5: BERT-base top 10 most frequent labels metrics - span-level

	precision	recall	f1-score	support
Acute respiratry failure	0.18	0.09	0.12	197
Atrial fibrillation	0.62	0.64	0.63	2846
CHF NOS	0.12	0.06	0.08	549
Cardiac dysrhythmias NEC	0.20	0.17	0.18	729
Crnry athrscl natve vssl	0.22	0.14	0.17	853
Hypertension NOS	0.25	0.11	0.16	455
Intermed coronary synd	0.07	0.05	0.06	241
Old myocardial infarct	0.27	0.28	0.27	352
Prim cardiomyopathy NEC	0.15	0.12	0.13	699
Subendo infarct, initial	0.32	0.34	0.33	661
micro avg	0.40	0.34	0.37	7582
macro avg	0.24	0.20	0.21	7582
weighted avg	0.36	0.34	0.34	7582

Figure 4.6: BERT-base top 10 most frequent labels metrics - token-level

	precision	recall	f1-score	support
Atrial fibrillation	0.66	0.57	0.61	1155
Cardiac dysrhythmias NEC	0.23	0.24	0.24	278
Crnry athrscl natve vssl	0.21	0.05	0.08	307
Prim cardiomyopathy NEC	0.09	0.06	0.07	240
Subendo infarct, initial	0.26	0.26	0.26	240
micro avg	0.47	0.37	0.41	2220
macro avg	0.29	0.24	0.25	2220
weighted avg	0.44	0.37	0.40	2220

Figure 4.7: BERT-base top 5 most frequent labels metrics - span-level

	precision	recall	f1-score	support
Atrial fibrillation	0.80	0.55	0.65	2692
Cardiac dysrhythmias NEC	0.34	0.27	0.30	740
Crnry athrscl natve vssl	0.41	0.07	0.12	902
Prim cardiomyopathy NEC	0.30	0.12	0.17	733
Subendo infarct, initial	0.47	0.38	0.42	717
micro avg	0.61	0.36	0.46	5784
macro avg	0.46	0.28	0.33	5784
weighted avg	0.58	0.36	0.43	5784

Figure 4.8: BERT-base top 5 most frequent labels metrics - token-level

Epoch	Training Loss	Validation Loss	Precision Macro	Recall Macro	F1 Macro
1	0.879500	0.713079	0.562268	0.105543	0.093065
2	0.796300	0.674240	0.206333	0.133802	0.123128
3	0.750700	0.661526	0.255288	0.108149	0.103083
4	0.741700	0.672869	0.297782	0.140362	0.141662
5	0.665000	0.668011	0.187835	0.167822	0.167102
6	0.622200	0.704936	0.194512	0.175737	0.176190
7	0.536800	0.763420	0.187545	0.178432	0.175392
8	0.506200	0.827349	0.187143	0.191214	0.184667

Figure 4.9: BERT-large training metrics log

```
from seqeval.metrics import classification_report
print(classification_report(y_true, y_pred, zero_division=1))
```

	precision	recall	f1-score	support
AMI anterior wall, init	0.12	0.10	0.11	49
Acute respiratry failure	0.12	0.05	0.07	82
Atrial fibrillation	0.52	0.70	0.60	1136
Atrial flutter	0.45	0.42	0.43	84
CHF NOS	0.09	0.04	0.06	186
Cardiac dysrhythmias NEC	0.15	0.15	0.15	261
Crnry athrscl natve vssl	0.11	0.11	0.11	329
Hypertension NOS	0.10	0.11	0.11	143
Intermed coronary synd	0.03	0.02	0.03	91
Mitral valve disorder	0.07	0.01	0.02	83
Old myocardial infarct	0.11	0.13	0.12	100
Parox ventric tachycard	0.14	0.17	0.16	47
Pericardial disease NOS	0.57	0.57	0.57	47
Prim cardiomyopathy NEC	0.05	0.03	0.04	320
Subendo infarct, initial	0.18	0.24	0.20	237
micro avg	0.32	0.33	0.32	3195
macro avg	0.19	0.19	0.18	3195
weighted avg	0.27	0.33	0.29	3195

Figure 4.10: BERT-large top 15 most frequent labels metrics - span-level

```
print(classification_report(y_true_tok, y_pred_tok, zero_division=1))
```

	precision	recall	f1-score	support
AMI anterior wall, init	0.25	0.15	0.19	169
Acute respiratry failure	0.17	0.04	0.06	229
Atrial fibrillation	0.55	0.69	0.61	2632
Atrial flutter	0.53	0.43	0.47	180
CHF NOS	0.23	0.07	0.11	512
Cardiac dysrhythmias NEC	0.22	0.18	0.20	688
Crnry athrscl natve vssl	0.19	0.13	0.15	949
Hypertension NOS	0.20	0.16	0.18	389
Intermed coronary synd	0.10	0.05	0.06	243
Mitral valve disorder	0.17	0.02	0.04	239
Old myocardial infarct	0.25	0.23	0.24	327
Parox ventric tachycard	0.24	0.19	0.21	129
Pericardial disease NOS	0.64	0.63	0.63	91
Prim cardiomyopathy NEC	0.16	0.06	0.09	970
Subendo infarct, initial	0.30	0.32	0.31	710
micro avg	0.38	0.32	0.35	8457
macro avg	0.28	0.22	0.24	8457
weighted avg	0.32	0.32	0.31	8457

Figure 4.11: BERT-large top 15 most frequent labels metrics - token-level

	precision	recall	f1-score	support
Acute respiratry failure	0.07	0.06	0.06	80
Atrial fibrillation	0.60	0.66	0.63	1229
CHF NOS	0.08	0.07	0.07	187
Cardiac dysrhythmias NEC	0.16	0.17	0.16	278
Crnry athrscl natve vssl	0.19	0.20	0.19	299
Hypertension NOS	0.14	0.14	0.14	164
Intermed coronary synd	0.03	0.02	0.03	87
Old myocardial infarct	0.16	0.19	0.18	108
Prim cardiomyopathy NEC	0.10	0.11	0.10	243
Subendo infarct, initial	0.23	0.28	0.25	214
micro avg	0.34	0.37	0.36	2889
macro avg	0.17	0.19	0.18	2889
weighted avg	0.34	0.37	0.35	2889

Figure 4.12: BERT-large top 10 most frequent labels metrics - span-level

	precision	recall	f1-score	support
Acute respiratry failure	0.10	0.08	0.09	197
Atrial fibrillation	0.63	0.64	0.63	2846
CHF NOS	0.15	0.10	0.12	549
Cardiac dysrhythmias NEC	0.21	0.18	0.19	729
Crnry athrscl natve vssl	0.28	0.25	0.27	853
Hypertension NOS	0.20	0.15	0.17	455
Intermed coronary synd	0.09	0.05	0.06	241
Old myocardial infarct	0.31	0.31	0.31	352
Prim cardiomyopathy NEC	0.16	0.15	0.15	699
Subendo infarct, initial	0.36	0.36	0.36	661
micro avg	0.40	0.36	0.38	7582
macro avg	0.25	0.23	0.24	7582
weighted avg	0.38	0.36	0.37	7582

Figure 4.13: BERT-large top 10 most frequent labels metrics - token-level

	precision	recall	f1-score	support
Atrial fibrillation	0.66	0.70	0.68	1155
Cardiac dysrhythmias NEC	0.17	0.15	0.16	278
Crnry athrscl natve vssl	0.15	0.17	0.16	307
Prim cardiomyopathy NEC	0.12	0.10	0.11	240
Subendo infarct, initial	0.27	0.28	0.28	240
micro avg	0.44	0.45	0.44	2220
macro avg	0.27	0.28	0.28	2220
weighted avg	0.43	0.45	0.44	2220

Figure 4.14: BERT-large top 5 most frequent labels metrics - span-level

	precision	recall	f1-score	support
Atrial fibrillation	0.68	0.66	0.67	2692
Cardiac dysrhythmias NEC	0.22	0.17	0.20	740
Crnry athrscl natve vssl	0.23	0.20	0.22	902
Prim cardiomyopathy NEC	0.19	0.14	0.16	733
Subendo infarct, initial	0.43	0.39	0.41	717
micro avg	0.48	0.43	0.45	5784
macro avg	0.35	0.31	0.33	5784
weighted avg	0.46	0.43	0.44	5784

Figure 4.15: BERT-large top 5 most frequent labels metrics - token-level

Upon reviewing these results, it becomes evident that the comparison between the top 10 most frequent labels and the top 15 most frequent labels does not yield substantial differences. However, when restricting the analysis to only the top 5 most frequent labels and the top 10 most frequent labels, a notable increase in the F1-score is observed. This can be attributed to the considerably higher frequency of the top 5 most frequent labels compared to other categories.

It is highly probable that if we were to graphically represent the F1-score while considering subsets of labels, specifically the top 5k, 10k, 15k, and so forth, we would observe a significant decline in the F1-score between the top 5k and top 10k categories, with diminishing changes as we extend further into the label hierarchy.

Conclusions

In conclusion, the approach presented in this thesis has shed light on the innovative utilization of Large Language Models (LLMs) for the extraction of ICD-9 entities from domain-specific text, such as Mimic-III. By addressing the data scarcity issue within this context, it has paved the way for valuable insights. The extracted data serves as a form of Knowledge Distillation, facilitating the adaptation of more compact and cost-effective models like BERT, renowned for its prowess in tasks like Named Entity Recognition (NER).

While it is evident that Knowledge Distillation requires a substantial amount of data to achieve optimal performance, this approach has showcased its effectiveness, particularly in the case of the five most frequent labels. These promising results were achieved despite the limited amount of available data.

Looking ahead, future work will focus on the continuation of the distillation process, with an emphasis on extracting a larger volume of data. The aim is to further enhance the model's entity recognition capabilities. It is hoped that this expansion of data will lead to a more balanced distribution of labels, ultimately bolstering the model's performance in this critical task.

In addition to expanding the volume of data, a crucial avenue for future improvement lies in enhancing the quality of the extracted data, especially in terms of the association between entities and their assigned types. An intriguing approach to achieve this enhancement could involve leveraging the capabilities of the LLM itself. By tasking the LLM with verifying the correctness of entity-type associations, we can establish a more robust and reliable framework. This self-auditing mechanism would not only contribute to improved data quality but also foster a deeper level of trust in the entity recognition process

Ringraziamenti

Ringrazio in primis il prof. Gianluca Moro che mi ha offerto questa opportunità di tesi estremamente caratterizzante e formante per gli studi di Artificial Intelligence che seguirò alla Magistrale. Un ringraziamento speciale al co-relatore Dottor Giacomo Frisoni per l'estremo aiuto ogni giorno (e notte) fino alla data di consegna. Vorrei anche ringraziare Alessio Cocchieri per l'impegno che ha messo nell'aiutarmi con il lavoro della tesi.

Un ringraziamento e un abbraccio vanno alla mia famiglia che ha sempre supportato i miei studi a prescindere dai risultati ottenuti.

Ringrazio tutti i nuovi amici che ho incontrato durante questi tre anni di Università, perché pur sempre sono persone che hanno contribuito alla mia crescita.

In particolare vorrei ringraziare Benedetta e Valentina con cui ho condiviso la maggior parte della mia vita universitaria.

Ringrazio ovviamente i miei fantastici coinquilini: Francesco, Rachele, Martina e Anna.

E un ringraziamento super speciale va alla mia ragazza Giulia. *Salvatore*

5 Ottobre 2023

Acknowledgments

I thank first and foremost Prof. Gianluca Moro, who offered me this extremely characterizing and educational thesis opportunity for my studies in Artificial Intelligence that I will pursue in the Master's program. A special thanks to my co-advisor, Dr. Giacomo Frisoni, for his extreme help every day (and night) until the submission date. I also want to thank Alessio Cocchieri for the effort he put into helping me with the thesis work.

A thanks and a hug go to my family, who has always supported my studies regardless of the results achieved.

I thank all the new friends I have met during these three years of university because they have all contributed to my growth.

In particular, I would like to thank Benedetta and Valentina, with whom I have shared most of my university life.

Of course, I thank my fantastic roommates: Francesco, Rachele, Martina, and Anna.

And a super special thanks goes to my girlfriend, Giulia. *Salvatore*

5 October 2023

Bibliography

[1] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE Trans. Knowl. Data Eng.*, 34(1):50–70, 2022.

[2] Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen, et al. Universalner: Targeted distillation from large language models for open named entity recognition. *CoRR*, abs/2308.03279, 2023.

[3] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, et al. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[4] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. *CoRR*, abs/2201.12023, 2022.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, et al. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[7] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023.

[8] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD*, pages 535–541. ACM, 2006.

[9] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[10] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *EMNLP*, pages 1317–1327. ACL, 2016.

[11] Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, et al. Lamini-lm: A diverse herd of distilled models from large-scale instructions. *CoRR*, abs/2304.14402, 2023.

[12] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, et al. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019.

[13] Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, et al. Want to reduce labeling cost? GPT-3 can help. In *EMNLP*, pages 4195–4205, Punta Cana, Dominican Republic, November 2021. ACL.

[14] Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, et al. Symbolic knowledge distillation: from general language models to commonsense models. In *NAACL*, pages 4602–4625, Seattle, United States, July 2022. ACL.

[15] Ryan Smith, Jason A. Fries, Braden Hancock, and Stephen H. Bach. Language models in the loop: Incorporating prompting into weak supervision. *CoRR*, abs/2205.02318, 2022.

[16] Priyanka Agrawal, Chris Alberti, Fantine Huot, Joshua Maynez, et al. Qameleon: Multilingual QA with only 5 examples. *CoRR*, abs/2211.08264, 2022.

[17] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, et al. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *ACL (Findings)*, pages 8003–8017. ACL, 2023.

[18] Zhen Guo, Peiqi Wang, Yanwei Wang, and Shangdi Yu. Dr. llama: Improving small language models on pubmedqa via generative data augmentation. *CoRR*, abs/2305.07804, 2023.

[19] Zheng Yuan, Zhengyun Zhao, Haixia Sun, Jiao Li, Fei Wang, and Sheng Yu. Coder: Knowledge infused cross-lingual medical term embedding for term normalization, 2021.

[20] Veysel Kocaman, Cabir Celik, Damla Gurbaz, Gursev Pirge, Bunyamin Polat, Halil Saglamlar, Meryem Vildan Sarikaya, Gokhan Turer, and David Talby. John_Snow_Labs@SMM4H'22: Social media mining for health

(#SMM4H) with spark NLP. In *Proceedings of The Seventh Workshop on Social Media Mining for Health Applications, Workshop & Shared Task*, pages 44–47, Gyeongju, Republic of Korea, October 2022. Association for Computational Linguistics.

[21] A. Johnson, T. Pollard, and R. Mark. Mimic-iii clinical database (version 1.4). *PhysioNet*, 2016.