NLP zur Unterstützung von SNOMED CT Codierung

Technische Dokumentation

Studiengang:	Medizininformatik
Autoren:	Sebastian Kunz und Cyril Zgraggen
Betreuer:	Prof. Dr. Murat Sariyar
Auftraggeber:	SIWF Schweizerisches Institut für ärztliche Weiter- und Fortbildung
Experte:	Reto Mettler
Datum:	16.06.2022

Inhaltsverzeichnis

1	Einleitung	3
2	Finetuning	3
	2.1 Systemarchitektur	4
	2.1.1 Finetuning Modell 1	4
	2.1.2 Finetuning Modell 2 und 3	5
	2.2 Technologie	6
	2.3 Verwendete Libraries	6
	2.4 Grundlagentabellen erstellen	8
	2.5 Label Dictionaries erstellen	9
	2.6 Trainingsdatensatz vorbereiten	10
	2.7 Finetuning durchführen	11
	2.8 Evaluation eines Modells	12
3	Umsetzung	13
	3.1 Systemarchitektur	13
	3.2 Technologie	14
	3.3 Verwendete Libraries	15
	3.4 Komponenten des Prototyps	16
	3.5 Vorgänge bei der Verwendung des Prototyps	17
4	Inbetriebnahme	19
	4.1 Anforderungen an das Zielsystem	19
	4.2 Installationsanleitung	20
	4.3 Anwendungsanleitung	21
Abbildungsverzeichnis		24
5	Tabellenverzeichnis	25
6	Literatur	26

1 Einleitung

In diesem Dokument werden die technischen Aspekte und der softwaretechnische Aufbau für die Bereiche Finetuning und Umsetzung des Prototyps der Bachelorthesis «NLP zur Unterstützung von SNOMED CT Codierung» dargestellt. Die Anforderungen an die zwei Bereiche sind im Pflichtenheft der BSc. Thesis detailliert dargelegt und bildeten den Rahmen für die jeweilige Umsetzung.
Dieses Dokument beinhaltet zudem die Installations- und Anwendungsanleitung für den Prototyp, welcher im Rahmen der Umsetzung entwickelt wurde.

Diese technische Dokumentation ist in die Teile «Finetuning», «Umsetzung» und «Inbetriebnahme» aufgeteilt.

Ein Grundgedanke der Arbeit war, dass unsere Arbeiten erweiterbar sein sollen und diejenigen, welche die Produkte unserer Arbeit in Händen halten, befähigen sollen neue Modelle zu erstellen oder diese in anderen Applikationen einzubinden.

2 Finetuning

Die Erstellung oder das Finetuning einer NLP Pipeline bildet das Kernstück der BSc Thesis.
Grundsätzlich sind für das Finetuning zwei Teilaufgaben notwendig:

- Datenaufbereitung für das Supervised Learning (Data collection)
- Durchführung des Trainings (Supervised Learning)

Die Datenaufbereitung bildet die Vorarbeit für das Training und beinhaltet drei Teilaufgaben:

- Erstellen der Grundlagentabellen (Sammlung der Trainingsdaten und SNOMED CT Konzept Tabelle)
- Erstellung des Trainingsdatensatzes
- Erstellen eines Label Verzeichnis (engl. Label Dictionary)

Für jede Teilaufgabe ist ein dezidiertes Jupyter Notebook kreiert worden. Die Notebooks sind so aufgebaut, dass sie fortlaufend ausgeführt werden können. Das Zusammenspiel der einzelnen Resultate der Notebooks, welche aus den Teilaufgaben entstehen, werden in den jeweiligen Unterkapiteln dargestellt.

2.1 Systemarchitektur

Während der Realisierungsphasen wurden auf Grund von Ressourcenproblemen bei der Hardware zwei unterschiedliche Architekturen verwendet. Die ursprünglich geplante und zum Schluss auch mehrheitlich verwendete Architektur ist «Finetuning Modell 2 und 3». Mit dieser Architektur wurde auch die komplette Datenaufbereitung realisiert. Hingegen wurde die Architektur «Finetuning Modell 1» nur für das Finetuning des ersten Modells genutzt.

2.1.1 Finetuning Modell 1

Basis dieser Architektur stellt die DGX Station von Nvidia dar. Das ist ein Server, der von der BFH betrieben wird und nur über das BFH Netzwerk zugänglich ist. Der Server stellt pro Sitzung 1 von 4 GPU Tesla V100-DGXS-32GB zur Verfügung. Über ein webbasiertes Interface kann in einem Docker Container ein Jupyter Notebook gestartet und genutzt werden.
Voraussetzung für eine Nutzung des Notebooks sind ein bestehender Trainingsdatensatz (Kapitel 2.6) und ein Label Dictionary (Kapitel 2.5).
Hugging Face ist eine Internet-Community. Sie bietet über eine implementierbare Schnittstelle Zugriff auf ein Archiv von vortrainierten KI-Modelle, welche genutzt oder für ein Finetuning verwendet werden können.
Die Ergebnisse aus dem Finetuning [Kapitel 2.7] sind die trainierten Gewichte des Modells aus dem Finetuning (saved weights). Diese werden auf dem Server gespeichert.

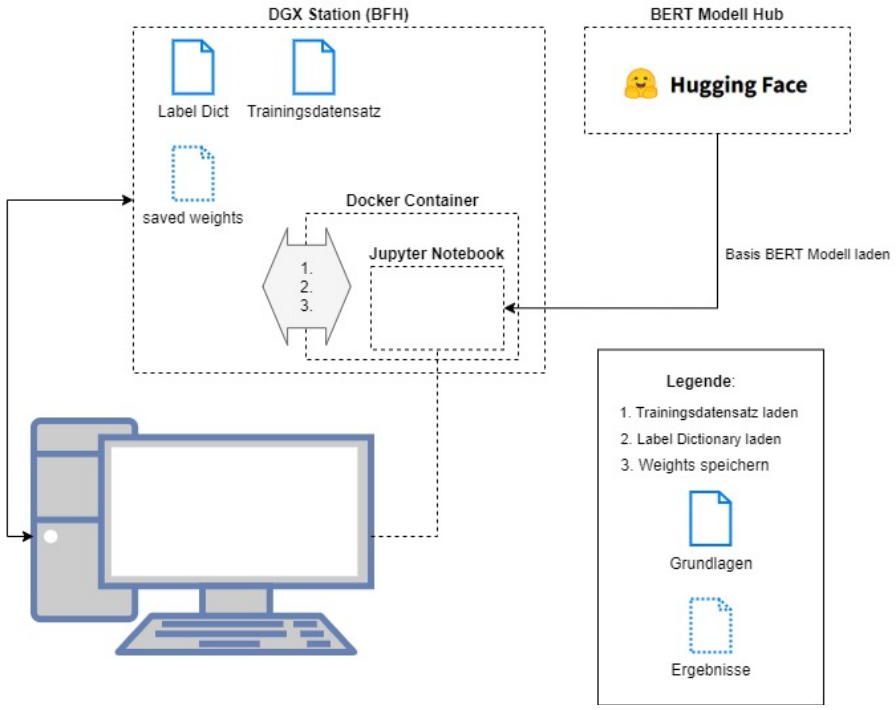

Abbildung 1: Architektur Finetuning Modell 1

2.1.2 Finetuning Modell 2 und 3

Basis dieser Architektur ist Google Colab. Auch dies basiert auf Jupyter Notebooks, welche via webbasiertes Interface bedient werden., es bietet jedoch keine Langzeitspeicherung von Ergebnissen. Google Colab bietet eine Integration von Google Drive an, damit die Daten aus der Cloud, der anwendenden Person, verwendet oder in die Cloud gespeichert werden können. Die kostenlose Version bietet den Zugang auf eine K80 GPU und 16 GB RAM, die Laufzeit ist auf 12 Stunden für eine Sitzung limitiert. Da diese Ressourcen für unsere Arbeiten nicht ausreichten, entschieden wir uns während der Entwicklung für ein kostenpflichtiges Upgrade auf Google Colab PRO+. Durch dieses Upgrade konnte auf die Leistung einer GPU T4 oder P100 und 52 GB RAM zurückgegriffen werden. Zudem verlängerte sich die Sitzungszeit auf 24 Stunden und die Sitzung konnte im Hintergrund ausgeführt werden.
Da diese Architektur nicht nur für das Training, sondern auch für das Preprocessing der Daten, verwendet werden kann, sind hier die Voraussetzungen für eine Nutzung des Notebooks neben den Trainingsdaten ein SNOMED CT Release Package, welches ein Abbild der Struktur von SNOMED CT zu einem bestimmten Zeitpunkt darstellt. Die Trainingsdaten müssen zu den Grundlagentabellen umgewandelt werden [Kapitel 2.4]. Diese werden wiederum für die Erstellung des Label Dictionary [Kapitel 2.5] benötigt. Für das effektive Training [Kapitel 2.7] benötigt es den Trainingsdatensatz [Kapitel 2.6] sowie das Label Dictionary und als Resultat aus dem Finetuning ergeben sich die Gewichte des Modells (saved weights).

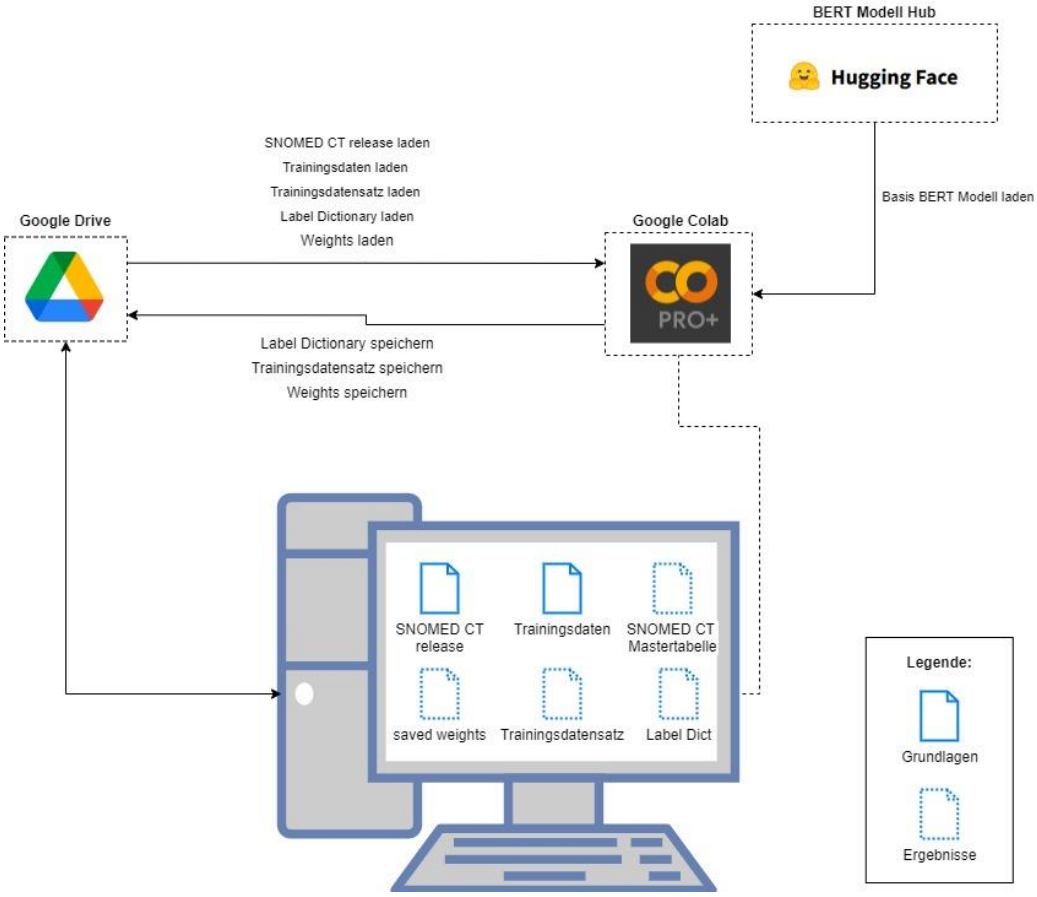

Abbildung 2: Finetuning Modell 2 und 3

2.2 Technologie

Als Technologie für den Bereich Finetuning wurde Jupyter Notebook genutzt.
«*Jupyter Notebook*» ist eine webbasierte interaktive Umgebung mit der Jupyter-Notebook-Dokumente erstellt werden können. Ein Jupyter-Notebook-Dokument ist ein JSON-Dokument mit einem versionierten Schema, dass aus einer Liste von Eingabe- und Ausgabezellen besteht, die jeweils Code, Text und Plots enthalten können. Die Dateinamensendung ist «.ipynb». Ein Jupyter Notebook kann aus der Browseroberfläche in verschiedene Formate konvertiert werden (1).
Die in den Jupyter Notebooks verwendete Programmiersprache ist Python. Python unterstützt mehrere Programmierparadigmen, z.B. objektorientierte oder funktionale Programmierung. Zusätzlich ist sie dynamisch typisiert und wird daher meist als Skriptsprache verwendet. Die Standardbibliothek ist mehrheitlich plattformunabhängig. Python bietet eine einfache Erweiterbarkeit der Standardbibliothek über eine Paketverwaltung, welche die Bibliotheken von einem Paketpool herunterlädt und die Abhängigkeiten der Pakete/Bibliotheken verwaltet.
Für das Paketmanagement wurde pip und entsprechend als Paketpool PyPI verwendet.
Die in dieser Arbeit verwendeten Bibliotheken sind alle frei zugänglich.

2.3 Verwendete Libraries

Programmbibliothek	Beschreibung	Verwendungszweck im Projekt
csv	Gehört zu den Standardbibliothek von Python und ist für die Verarbeitung von .csv Dateien.	Wird für die Verarbeitung des SNOMED CT Release Package verwendet.
pandas	Pandas bietet Funktionalität für die Verarbeitung und Analyse von Daten. Insbesondere enthält sie Datenstrukturen und Operatoren für den Zugriff auf numerische Tabellen und Zeitreihen (2).	Wird für die Arbeit mit Dataframes verwendet.
numpy	NumPy bietet eine simple Handhabung von Vektoren, Matrizen oder generell grossen mehrdimensionalen Arrays. Zusätzlich bietet NumPy eigene Datenstrukturen an.	Wird für die Übersetzung von PyTorch Tensor in Arrays verwendet.
json	Gehört zu den Standardbibliothek von Python und ist für die Verarbeitung von JSON Objekten zuständig.	Die Konzepte von JSON und Dictionary sind in Python sehr ähnlich. Zudem ist JSON ein Standardformat für den Datenaustausch mit REST APIs. Um die Anforderung einer REST API von Beginn an mit einzubeziehen, wurden die Label Dictionaries als JSON Dateien kreiert.
openpyxl	Openpyxl bietet Funktionalität für das Lesen und Schreiben von Excel Dateien.	Die im Finetuning erstellten Tabellen werden in Excel gespeichert, um eine nachträgliche manuelle Bearbeitung zu ermöglichen.
medcat	MedCAT ist ein Open Source Projekt, das entwickelt wurde, um Informationen aus elektronischen Gesundheitsakten zu extrahieren und diese mit biomedizinischen	MedCAT wurde bei der Verarbeitung bzw. dem Auslesen des SNOMED CT Release Packages verwendet.

	Ontologien z.B. UMLS oder SNOMED-CT zu verknüpfen	
fast_ml	Fast-ML ist für das Preprocessing und Einteilen eines Datensatzes in Trainings-, Validierungs- und Testdaten.	Wird im Rahmen der Einteilung des Datensatzes in Trainings-, Validierungs- und Testdaten verwendet.
sklearn	Scikit-learn ist eine Bibliothek für maschinelles Lernen (eng. Machine Learning ML). Sie bietet verschiedene Klassifikations-, Regressions- und Clustering-Algorithmen an. Dank Scikit-learn können obengenannte Aufgaben ebenfalls ausgewertet und visualisiert werden.	Scikit-learn wird im Rahmen dieser Arbeit zur Evaluation eines Modells mit dem Testdatensatz nach dem Training genutzt.
transformers	Transformers ist die Schnittstelle von Hugging Face. Über diese Bibliothek lassen sich vortrainierte ML-Modelle herunterladen. Zudem bietet Hugging Face über diese Schnittstelle verschiedene Architekturen für das Finetuning an. Transformers kann in Zusammenarbeit mit PyTorch, TensorFlow oder JAX zusammen für das Pre-Training, Finetuning und Verwenden von Modellen verwendet werden.	Als Basis-Modell unserer Arbeit wird BioBERT über diese Schnittstelle ins System geladen. Mittels der BertForSequenceClassification-Erweiterung wird dem Basismodel ein weiterer linearer Layer als Classifier angehängt. Dieser bildet die Basis für das Training unserer Modelle.
torch	PyTorch ist eine auf maschinelles Lernen ausgerichtete Bibliothek. Sie dient der Erstellung von neuronalen Netzen und der Verwendung von GPUs	Wird im Rahmen der Erstellung/Training des BERT-Classifiers verwendet und während dem Training für die Verwendung der GPU genutzt.
matplotlib	Matplotlib bietet viele Möglichkeiten mathematische Darstellungen zu generieren und anzufertigen.	Nach dem Training kann mit dieser Bibliothek der Verlauf des Trainings dargestellt werden.
tqdm	Tqdm ist für die Darstellung einer Progressbar während eines Loops.	Wird für die Darstellung des Fortschritts innerhalb des Trainings genutzt.

Tabelle 1: Verwendete Libraries im Finetuning

2.4 Grundlagentabellen erstellen

Dieses Jupyter Notebook beinhaltet drei Aufgaben für die Erstellung und Bereinigung der Daten, die für die Erstellung der Trainingsdatensätze und der Label Dictionaries nötig sind:

1. Extrahieren der benötigten SNOMED CT Konzepte aus dem SNOMED CT release Package
2. Erstellung der Tabelle der SNOMED CT Konzepte, die das Modell klassifizieren können, soll. Die Tabelle enthält die eineindeutige Zuordnung der SNOMED CT Codes und dessen Beschreibungen (Hauptbeschreibung und Synonyme). Die Tabelle wird im weitererführenden Text als «Mastertabelle» bezeichnet.
3. Erstellung der Tabelle aus den Queries, welche vom SIWF parametriert wurden. Die Queries werden auseinandergenommen und dem übergeordneten Konzept zugeordnet. Sie dient der Erweiterung des Trainingdatensatzes.

Die Tabellen, die durch das Notebook entstehen, müssen für die weitere Prozessierung gespeichert werden. Im Code ist eine Speichermethode für die Speicherung als Excel-Datei implementiert.

Die Tabellen werden wie folgt aufgebaut:

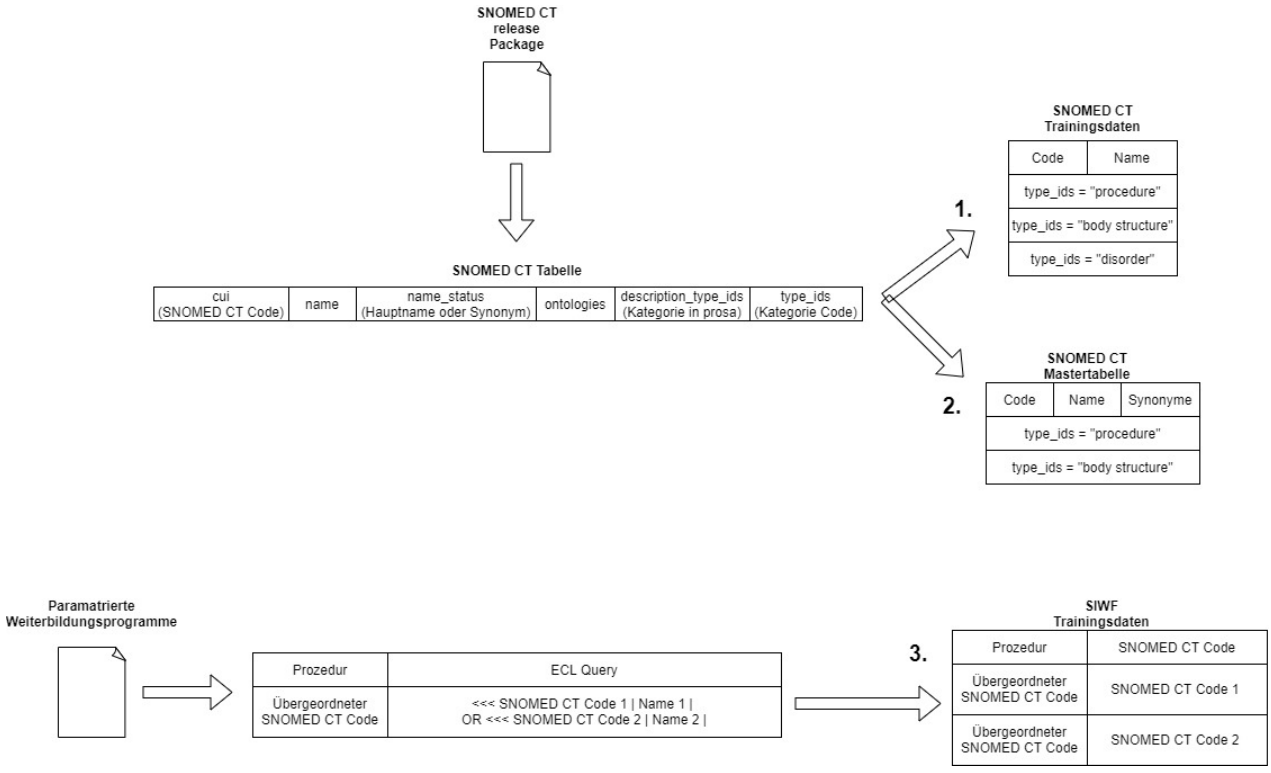

Abbildung 3: Grundlagentabellen erstellen

2.5 Label Dictionaries erstellen

Dieses Jupyter Notebook nutzt den für das Training vorgesehenen Trainingsdatensatz und die Mastertabelle.

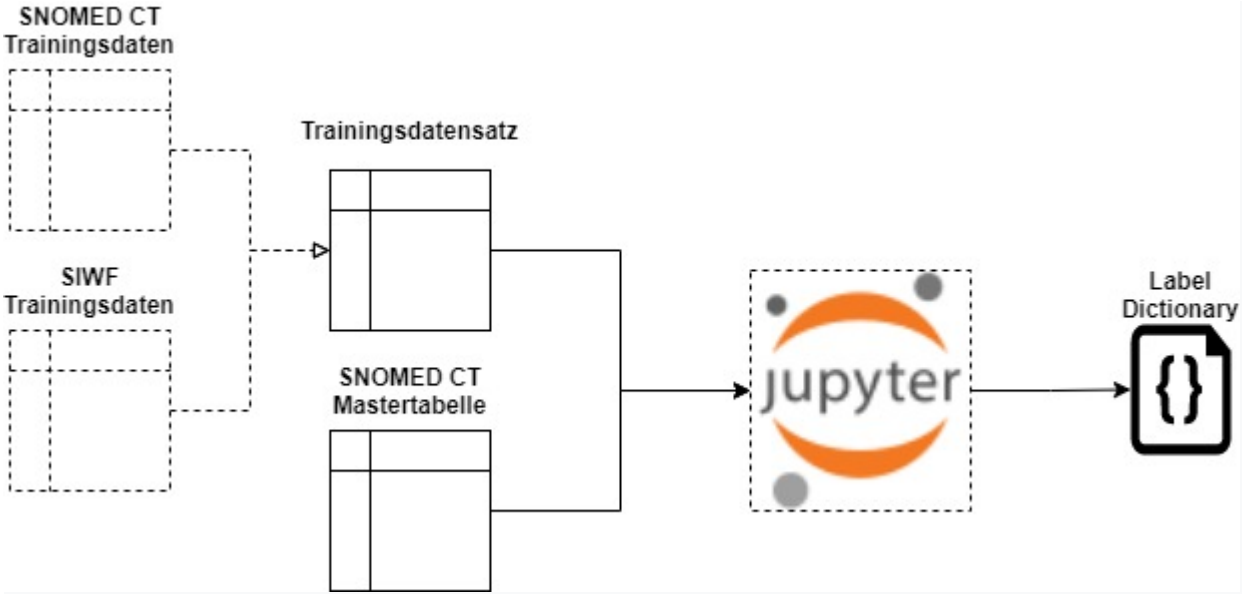

Abbildung 4: In- und Output des Jupyter Notebook «Label Dictionary erstellen»

Die Mastertabelle dient als Filter, um den Classifier nur auf die in der Mastertabelle enthaltenen SNOMED CT Konzepte zu trainieren. SNOMED CT Konzepte, die nicht in der Mastertabelle enthalten sind, aber im Trainingsdatensatz vorkommen, werden als «none oft he trained concepts» indexiert.
Das Label Dictionary hat folgende Struktur:

```
{
  "SNOMED CT Code" : {
    "name" : Name des Codes,
    "synonym" : ["synonym 1", "synonym 2"],
    "code" : "SNOMED CT Code",
    "index" : index
  }
}
```


```
"104001": {
    "name": "Excision of lesion of patella (procedure)",
    "synonym": [
        "Excision of lesion of patella",
        "Local excision of lesion or tissue of patella"
    ],
    "code": "104001",
    "index": 0},
"115006": {
    "name": "Removable appliance therapy (procedure)",
    "synonym": [
        "Removable appliance therapy",
        "Fit removable orthodontic appliance",
        "Insertion of removable orthodontic appliance"
    ],
    "code": "115006",
    "index": 1},
"119000": {
    "name": "Thoracoscopic partial lobectomy of lung (procedure)",
    "synonym": [
        "Thoracoscopic partial lobectomy of lung"
    ],
    "code": "119000",
    "index": 2},
```


Abbildung 5: Auszug Label Dictionary Modell 2

Ausser dem Index, der als Ganzzahl repräsentiert wird, werden die Einträge im Dictionary aus Zeichenfolgen (Strings) oder Zeichenfolge-Listen repräsentiert.
Für das Training wird eine Zuordnung der zu trainierenden SNOMED CT Konzepte mit einem Index benötigt. Dieser Index stellt während des Trainings das eindeutige Label für dieses Konzept dar. Der Index repräsentiert beim Classifier zudem nach der Verarbeitung eines Textes die Position des Konzeptes in der Output-Liste des Modells. Somit übernimmt das Label Dictionary die Übersetzung zwischen Modell und SNOMED CT. Entsprechend wird es beim Training und Verwendung des Modells/Pipeline eingesetzt. Abgespeichert wird das Label Dictionary als JSON Datei.

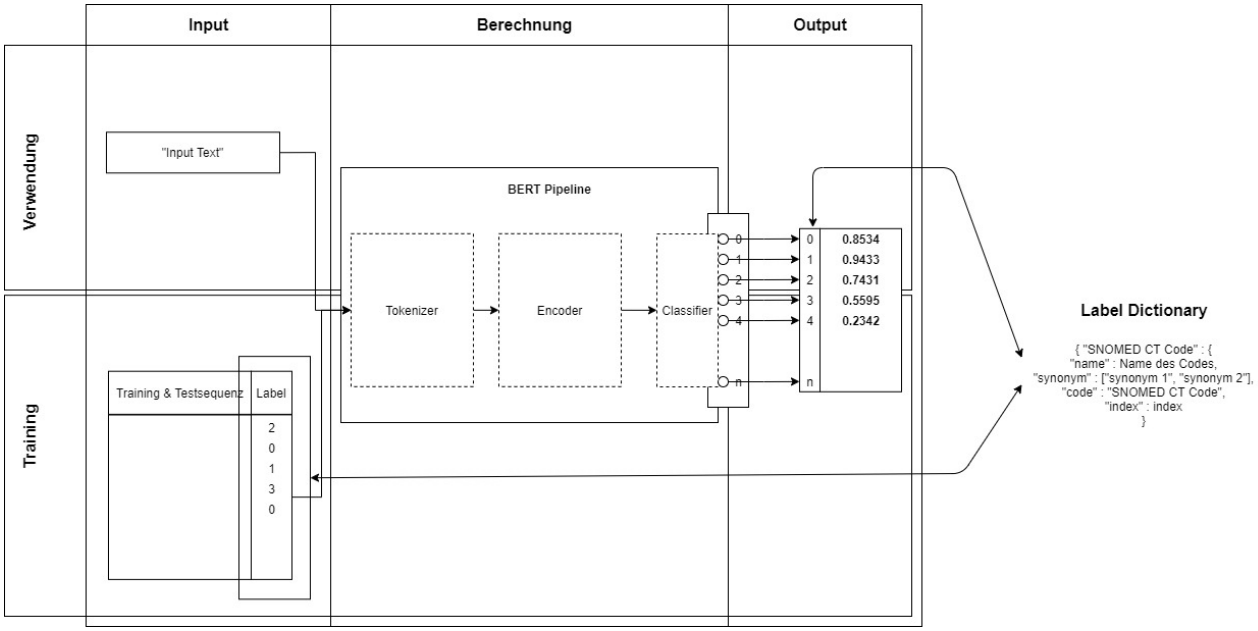

Abbildung 6: Funktion des Label Dictionary

2.6 Trainingsdatensatz vorbereiten

Dieses Jupyter Notebook benötigt eine Tabelle, in welcher die Trainingsdaten zusammengeführt sind. Die Tabelle muss die Spalten «Text» und «Code» enthalten. Der Text stellt den Input dar, der das Modell erhält, und der Code ist das SNOMED CT Konzept, das durch den Text beschrieben wird.
Das Jupyter Notebook hat verschiedene aufeinander ab folgende Aufgaben:

1. Die Zeilen mit dem aus dem Label Dictionary enthaltenen Index als Label erweitern.
2. Aufteilen der Zeilen in Trainings-, Validations- und Testdatensatz.
3. Sicherstellen, dass jedes im Label Dictionary enthaltenen Konzept mindestens einmal im Trainingsdatensatz vorkommt.
4. Erweitern der Zeilen mit dem in der Aufteilung entsprechenden Keywords «train», «val» oder «test».
5. Abspeicherung der Tabelle für das Training als Excel Datei

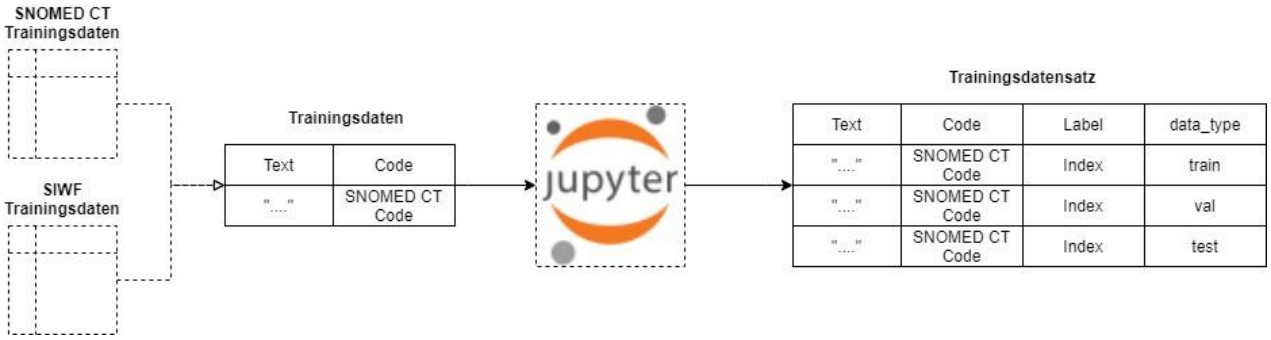

Abbildung 7: In- und Output vom Jupyter Notebook «Trainingsdatensatz vorbereiten»

2.7 Finetuning durchführen

Dieses Jupyter Notebook stellt das effektive Training eines BertForSequenceClassification Modells dar.
Das Basis-BERT-Modell kann über Hugging Face, im Rahmen der Möglichkeiten, welche im Hub verfügbar sind, frei gewählt werden. Das Modell wird über eine Schnittstelle der Bibliothek «transformers« direkt ins Projekt geladen.
Das Basis-BERT-Modell hat zwei Aufgaben: Einerseits dient der darin enthaltene Tokenizer für die Übersetzung der Texte in die Word Embeddings des Modelles und andererseits wird es als Basis für das BertForSequenceClassification Modell verwendet. Die Transformers Klasse «BertForSequenceClassification erweitert das Basis-BERT-Modell mit einem PyTorch Linear Layer. Dieser Layer repräsentiert den Classifier und wird im folgenden Training für die Klassifizierungsaufgabe angelernt. Die Grösse des linearen Layers wird durch das Label Dictionary bestimmt. Das bedeutet, dass der lineare Layer gleichviele Neuronen wie Anzahl Labels im Verzeichnis hat.
In unserem Fall wurden alle Trainings mit BioBERT, genauer «dmis-lab/biobert-base-cased-v1.2» realisiert.

Für das Training wird der vorbereitete Datensatz und das für dieses Modell entwickelte Label Dictionary benötigt.

Der Datensatz wird in die drei Datensets «train», «val» und «test» eingeteilt und durch den Tokenizer für das Modell verarbeitbar gemacht. Die Sets werden wiederum in Batches eingeteilt.
Der gesamte Batchbestand eines Datensets wird über einen Dataloader (Iteratoren) zur Verfügung gestellt.
Die Trainingsmethode iteriert nun für jede Epoche durch den Dataloader und erhält so die einzelnen Batches. Die Batches werden auch wieder über jedes in ihm enthaltene Element iteriert.
Diese Elemente werden dem Modell zur Verarbeitung übergeben, das Resultat analysiert und über Backpropagation die Gewichte angepasst.
Anhand der durchschnittlichen Fehlerrate einer Epoche wird die Epoche mit dem bisherig gespeicherten Modell verglichen und das bessere Modell abgespeichert. Somit wird sichergestellt, dass das beste Modell gespeichert wird und der Speicherplatz geschont wird.

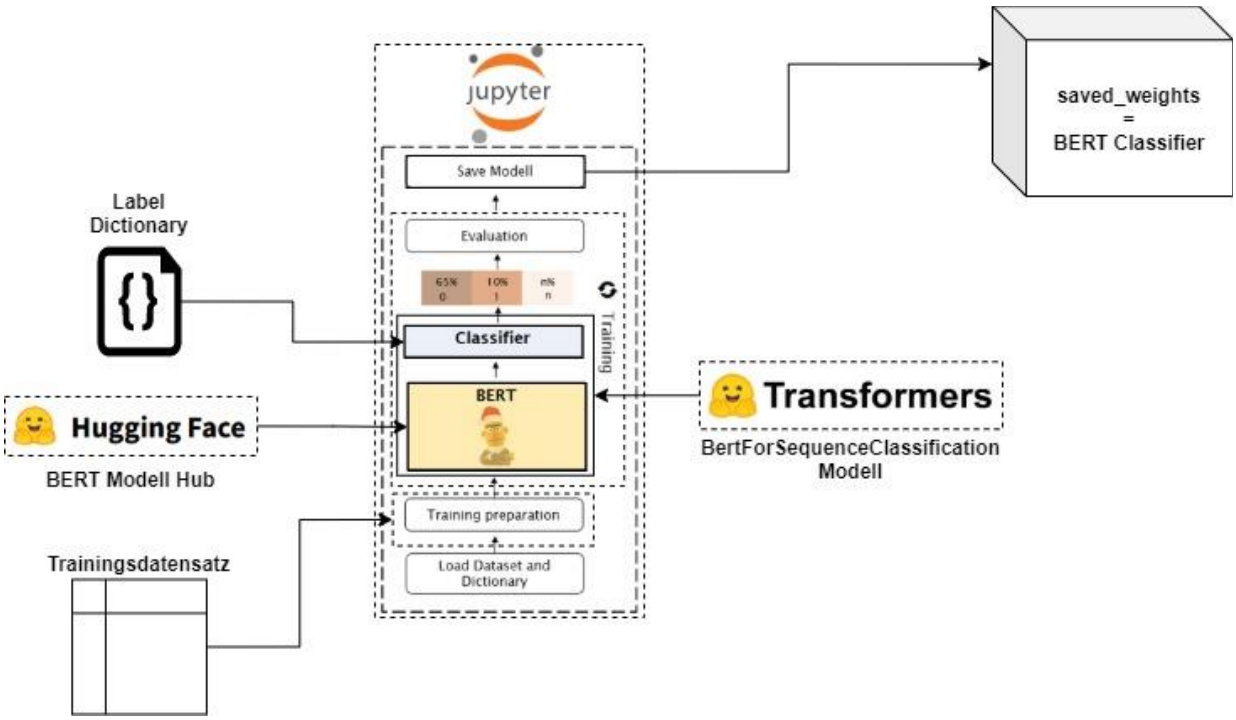

Abbildung 8: Funktionsweise des Jupyter Notebook «Finetuning durchführen»

Evaluation während Finetuning
Während des Trainings wird am Schluss jeder Epoche das Modell mit einer Evaluationsmethode ausgewertet. Der Output dieser Methode ist mitunter eine Liste der richtigen Labels des Evaluationsdatensatzes und eine Liste der Labels, die berechnet wurden.
Für die Berechnung des Evaluations F1 wird die F1_score_func Methode von Scikit Learn verwendet. Diese berechnet anhand der wahren Labels und der prognostizierten Labels den F1 Wert.

Die gleiche Scikit Learn Methode kommt bei der Evaluation des Testdatensatzes am Ende des Trainings zum Einsatz. Zusätzlich wird noch mittels classification_report aus Scikit Learn eine Übersicht über die Klassifizierungspräzision des Modells gemacht. Hierbei ergibt sich eine Darstellung, in welcher jedes Label isoliert betrachtet werden kann. Dabei wird die Precision, der Recall und der F1-Scores jedes einzelnen Labels sowie über den ganzen Datensatz die Accuracy berechnet.
Die Resultate werden nur im Jupyter Notebook dargestellt.

2.8 Evaluation eines Modells

Dieses Jupyter Notebook entstand, um eine Einschätzung über den gesamten Datensatz machen zu können.
Für die Verarbeitung wird der Name des Basis-BERT-Modells benötigt, das Label Dictionary des Modells und die trainierten Gewichte des Modells.
Bei der Erstellung wird dem Modell noch ein PyTorch Softmax Layer angehängt. Dieser normalisiert den Output des Modells und dessen Verteilung.

Für die Evaluation wird der gesamte Datensatz (ohne Trainingseinteilung) iteriert und jeweils jeder einzelne Text dem Modell als Input gegeben.
Vom Output des Modells wird die Vorhersage des wahren Labels, die höchste Vorhersage, das Label der höchsten Vorhersage und der SNOMED CT Code der Tabelle angehängt. Für die Übersetzung des Indexes des Outputs benötigt es wiederum das Label Dictionary.
Dies wird für die weitere Verarbeitung als Excel gespeichert.

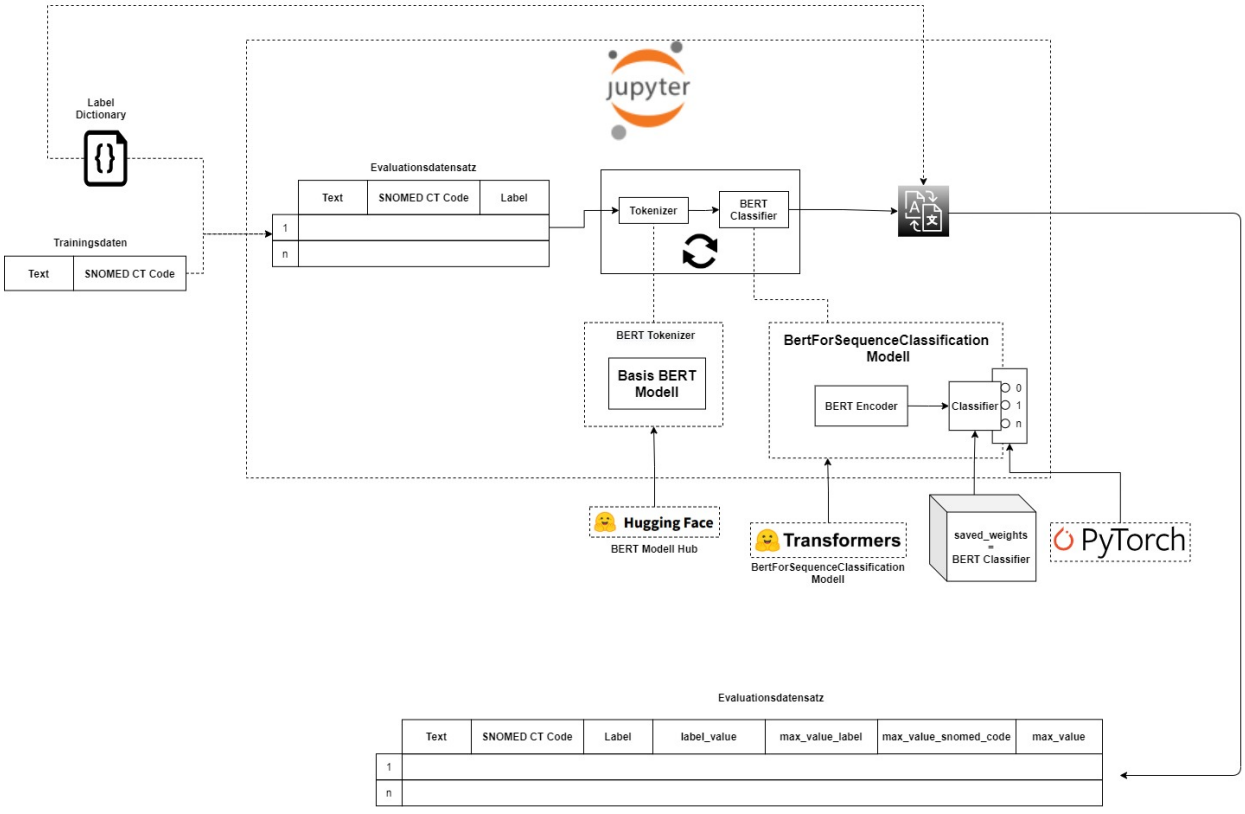

Abbildung 9: Funktionsweise des Jupyter Notebook «Evaluation eines Modells»

3 Umsetzung

Die Umsetzung stellt den Prototyp und dessen Funktionsweise dar. Der Prototyp wurde für die Nutzung der entwickelten Modelle und Darstellung der Resultate im Rahmen der Nutzung durch das SIWF entworfen und umgesetzt.

3.1 Systemarchitektur

Basis der Architektur für den Prototyp ein ausführbares Python Projekt, welches alle für die Ausführung notwendigen Dateien enthält. Ein Computer, welcher Internetzugang und einen Python Interpreter hat, gewährleistet die korrekte Ausführung.
Ein Vorteil davon ist, dass der Prototyp unabhängig vom Umsystem auf die nötigen Ressourcen zugreifen kann und so die Struktur des Umsystems nicht kennen muss.
Um die Ausführbarkeit der Pythonscripts zu gewährleisten, bietet sich eine virtuelle Bibliotheksverwaltung an.
Durch diese Art der Umsetzung wird auch die einfache Erweiterbarkeit mit zusätzlichen, zukünftig entwickelten Modellen gewährleistet.

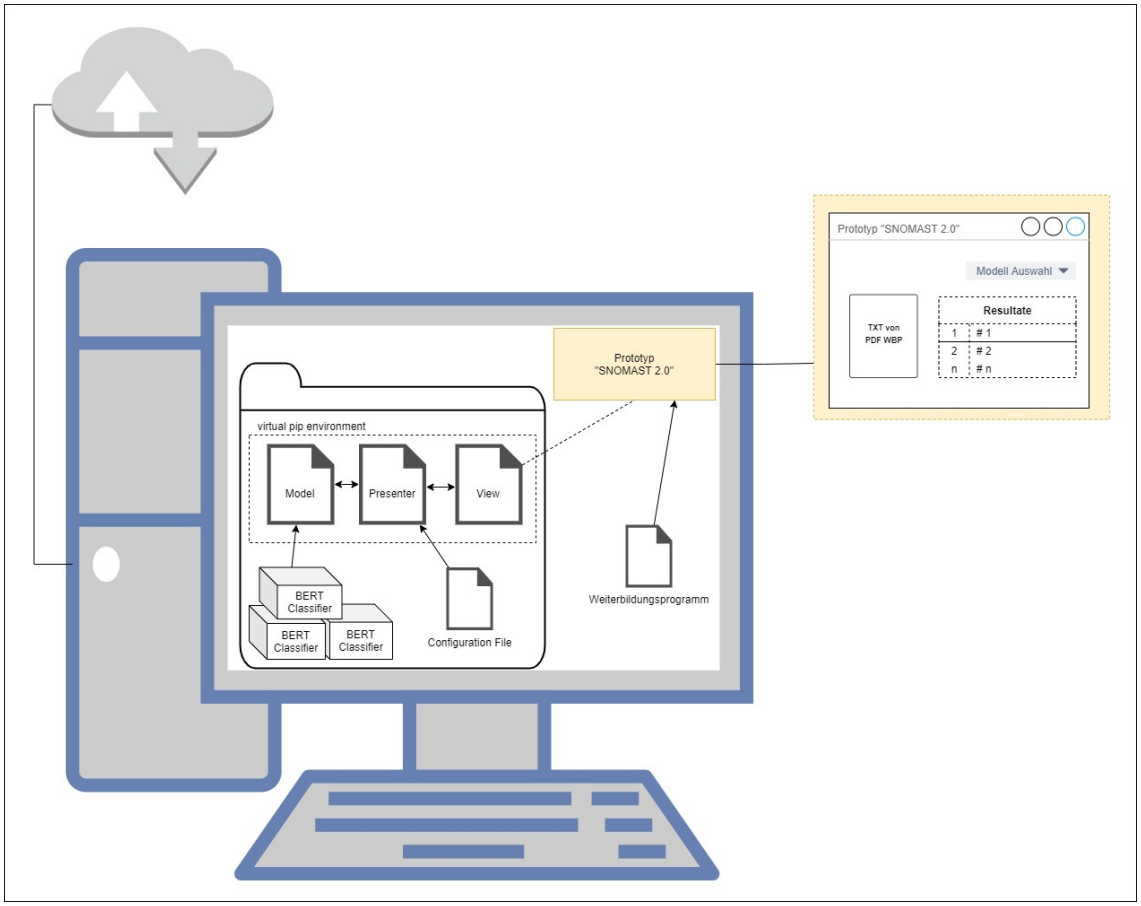

Abbildung 10: Systemarchitektur des Prototyps für die lokale Verwendung

3.2 Technologie

Der Prototyp wurde in der Programmiersprache Python entwickelt.
Als Entwicklungsumgebung wurde PyCharm von JetBrains in der Community Version verwendet.

Das Python Projekt wurde als pipenv Projekt erstellt. Pipenv ist ein Tool für die virtuelle Bibliotheksverwaltung. Während der Entwicklung wird ein Pipfile manuell erweitert. Diese Datei ersetzt das herkömmliche requierments.txt oder die Einrichtung eines virtuellen Projektes und verwaltet die für die Ausführung benötigte Pythonversion, die Bibliotheken und deren Abhängigkeiten, sowie Anbindungen an andere Skripts.
Bei der Ausführung eines Pipenv, wird eine virtuelle Python-Umgebung (Interpreter & Bibliotheksverwaltung) mit den im Pipfile enthaltenen Angaben kreiert und das Projekt mit dieser Umgebung ausgeführt. Somit entfällt für die anwendenden Person das Anpassen der lokalen Python-Umgebung auf die Anforderungen des Projekts.
Für die Versionsverwaltung wurde Git bzw. GitHub genutzt.
Aufgrund der Dateigrösse der BERT Classifier-Dateien, konnten diese nicht mittels der Versionsverwaltung verwaltet werden.
Somit lässt sich auch der fertige Prototyp nicht über diese Plattform zugänglich machen. Es zeigte sich durch die Evaluation mit dem Auftraggeber, dass durch die einfache Systemarchitektur des Prototyps, dieser mit allen notwendigen Dateien sehr einfach über andere Clouddienste geteilt werden kann.

Für das GUI wurde auf das Framework PyQt5 gesetzt. PyQt ist eine Python-Anbindung für das plattformübergreifende GUI Toolkit QT. PyQt bietet über eine Bibliothekserweiterung auch Zugriff auf den PyQt Designer an. Der PyQt Designer ist ein grafischer GUI Editor, welcher es der anwendenden Person erlaubt, ein GUI per Drag&Drop zu erstellen. Nach der Erstellung des GUI kann der daraus entstandene Python-Code mit den benötigten Funktionalitäten/Methoden erweitert werden.

3.3 Verwendete Libraries

Programmbibliothek	Beschreibung	Verwendungszweck im Projekt
PyQT5	PyQT ist ein GUI-Toolkit auf Basis von Qt	PyQT wurde für die Erstellung des GUI des Prototyps «SNOMAST 2.0» verwendet.
transformers	Transformers ist die Schnittstelle von Hugging Face. Über diese Bibliothek lassen sich vortrainierte ML Modelle herunterladen. Zudem bietet Hugging Face über diese Schnittstelle verschiedene Architekturen für das Finetuning an. Transformers kann in Zusammenarbeit mit PyTorch, TensorFlow oder JAX zusammen für das Pre-Training, Finetuning und Verwenden von Modellen verwendet werden.	Um im Prototyp die entwickelten Modelle nutzen zu können, benötigt es einerseits den BERT Tokenizer des Basis BERT Modells und andererseits wird bei der Erstellung eines Modelles die Klasse BertForSequenceClassification geladen, mit dem Basis BERT Modell und mit den entwickelten Gewichten bestückt.
torch	PyTorch ist eine auf maschinelles Lernen ausgerichtete Bibliothek. Sie dient der Erstellung von neuronalen Netzen und der Verwendung von GPUs	Dem BertForSequenceClassification-modell wird bei der Verwendung des Modells ein PyTorch Softmax Layer angehängt. Dieser wird für die Normalisierung des Outputs und dessen Verteilung verwendet.
deepl	Python Package welches für die Nutzung der DeepL API benötigt wird.	Wird im Prototyp «SNOMAST 2.0» für die Übersetzung des im Modell zu verarbeitenden Textes in die englische Sprache verwendet.
fitz	Fitz oder PyMuPDF ist ein Package, welches die Verarbeitung von PDFs ermöglicht.	Wird im Prototyp für das Einlesen und Anzeigen der PDF-Dokumente verwendet.
pandas	Pandas ist für die Verarbeitung und Analyse von Daten. Insbesondere enthält sie Datenstrukturen und Operatoren für den Zugriff auf numerische Tabellen und Zeitreihen.(2)	Wird für die Arbeit mit Dataframes verwendet.
numpy	NumPy bietet eine einfache Handhabung von Vektoren, Matrizen oder generell grossen mehrdimensionalen Arrays. Zusätzlich bietet sie eigene Datenstrukturen an.	Wird für die Übersetzung von PyTorch Tensor in Arrays verwendet.
json	Gehört zu den Standardbibliothek von Python und ist für die Verarbeitung von JSON Objekten zuständig.	Wird für die Verwendung des Label Dictionary verwendet.

Tabelle 2: Verwendete Libraries des Prototyps

3.4 Komponenten des Prototyps

Da beim Prototyp mit Python gearbeitet wurde, wurden die einzelnen Komponenten des Projektes skriptbasiert gehalten.
Die Umsetzung wurde im Rahmen des MVP (Model-View-Presenter) Patterns gestaltet. In diesem Softwareentwurfsmuster soll die View lediglich Daten von den Nutzerinnen und Nutzern sammeln und ihnen Daten anzeigen. Das Modell ist für die Verarbeitung der Daten zuständig und der Presenter ist die Schnittstelle zwischen der Ansicht (View) und dem verarbeitenden Modell (Model).
Die Komponenten «View», «Logic», «Reader» und «Model» stellen im Prototyp eigenständige Python-Dateien dar. Der Presenter ist in zwei Dateien aufgeteilt. Der «Reader» enthält lediglich Hilfsmethoden für die «View», um die URL der zu importierenden Dateien auszulesen und die Texte in der «View» darzustellen. Hingegen agiert die «Logic» als Schnittstelle zwischen «View» und «Model».

Für die systemunabhängige Paketverwaltung ist im Prototyp das Pipfile bzw. Pipfile.lock hinterlegt.

Da die Texte, welche die Nutzerinnen und Nutzer dem Model zur Verarbeitung übergeben ins Englische übersetzt werden müssen, wurde die DeepL API explizit als eigene Komponente mit aufgeführt. Auch wenn sie nur als verwendete Bibliothek der «Logic» im Prototyp enthalten ist.
Auch als Komponente aufgeführt ist die Bibliothek von Hugging Face «Transformers». Sie beinhaltet auch eine API, welche zuständig ist für das Herunterladen des Basis-BERT-Modells. Somit ist sie ein wesentlicher Bestandteil des «Modells» bzw. bei der Erstellung einer Instanz der selbstentwickelten BertClassifier Klasse.

Der Prototyp enthält zusätzlich das «Configuration File» in Form einer JSON Datei. Es dient der Verwaltung der zur Verfügung stehenden Modelle und kann manuell mit neuen Modellen erweitert werden. Für jedes Modell, welches im «Configuration File» deklariert wurde, muss im Prototyp das entsprechende Label Dictionary und die Gewichte des Modells abgelegt werden.
Das «Configuration File» hat folgende Struktur:

```
{
  "Modellnamen" : {
    "name" : "Modellnamen",
    "description" : "Beschreibung des Modelles"
    "label_dict_url" : "URL des Label Dictionary des Modells",
    "train_snapshot_url" : "URL des model_state_dict des Modells",
    "base_bert_name": "Name des Basis BERT Modells"
  }
}
```

```
"Modell 1": {
  "name": "Modell 1",
  "description": "Das Modell 1 ist ein BERT-Modell mit einem binaeren Klassifikator.
                  Es berechnet anhand des Inputs, ob es sich um ein Konzept der Kategorie
                  'procedure' und/oder 'body structure' (SNOMED CT) oder der Kategorie 'disorder'
                  (NOT SNOMED CT) handelt.",
  "label_dict_url": "l_dict_m1.json",
  "train_snapshot_url": "saved_weights_M1_v1.model",
  "base_bert_name": "dmis-lab/biobert-base-cased-v1.2"
},
"Modell 2": {
  "name": "Modell 2",
  "description": "Das Modell 2 ist ein BERT-Modell mit einem Multiclass Klassifikator.
                  Das Modell 2 umfasst 89'855 Klassen. Bis auf eine Klasse, repraesentiert
                  jede Klasse ein SNOMED CT Konzept der Kategorie 'procedure' oder 'body structure'.
                  Das Modell 2 berechnet anhand des Inputs, die n-Anzahl wahrscheinlichsten Konzepte
                  aus den 89'855 Klassen.",
  "label_dict_url": "l_dict_m2.json",
  "train_snapshot_url": "saved_weights_M2_v1.model",
  "base_bert_name": "dmis-lab/biobert-base-cased-v1.2"
},
"Modell 3": {
  "name": "Modell 3",
  "description": "Das Modell 3 ist ein BERT-Modell mit einem Multiclass Klassifikator.
                  Das Modell 3 umfasst 368 Klassen. Bis auf eine Klasse, repraesentiert
                  jede Klasse ein SNOMED CT Konzept der Kategorie 'procedure' oder 'body structure',
                  die bereits vom SIWF in den Weiterbildungsprogrammen identifiziert wurden.
                  Das Modell 3 berechnet anhand des Inputs, die n-Anzahl wahrscheinlichsten
                  Konzepte aus den 368 Klassen.",
  "label_dict_url": "l_dict_m3.json",
  "train_snapshot_url": "saved_weights_M3_v1.model",
  "base_bert_name": "dmis-lab/biobert-base-cased-v1.2"
}
```

Abbildung 11. Auszug aus dem Configuration File des Prototyps

Durch das folgende Komponentendiagramm wird das Zusammenspiel verdeutlicht.

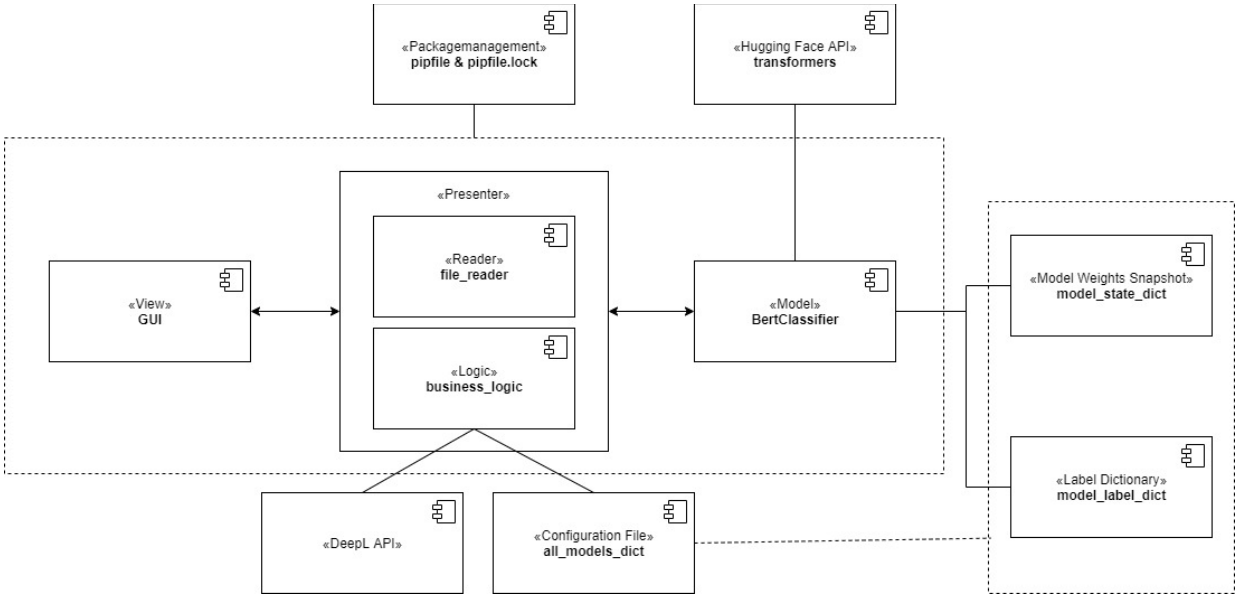

Abbildung 12: Komponentendiagramm des Prototyps

3.5 Vorgänge bei der Verwendung des Prototyps

Um die Zusammenarbeit der verschiedenen Komponenten zu verdeutlichen, werden durch die folgenden Sequenzdiagramme die Abläufe von zwei Anwendungsfällen etwas genauer betrachtet. Der erste Anwendungsfall stellt den Import einer PDF oder TXT Datei zur Ansicht im Prototyp dar.

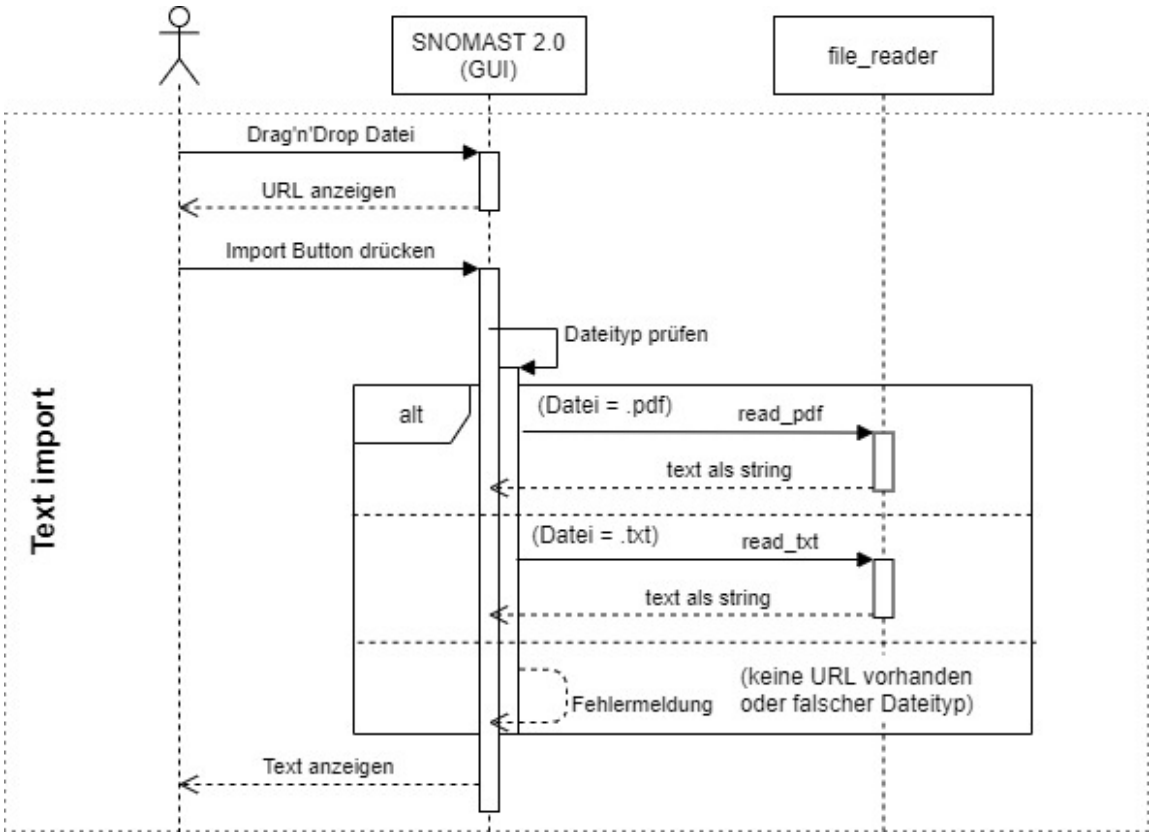

Abbildung 13: Sequenzdiagramm für Import eines Textes in den Prototyp

Der zweite Anwendungsfall verdeutlicht den Ablauf der Nutzung eines BERT Modells/Classifier.

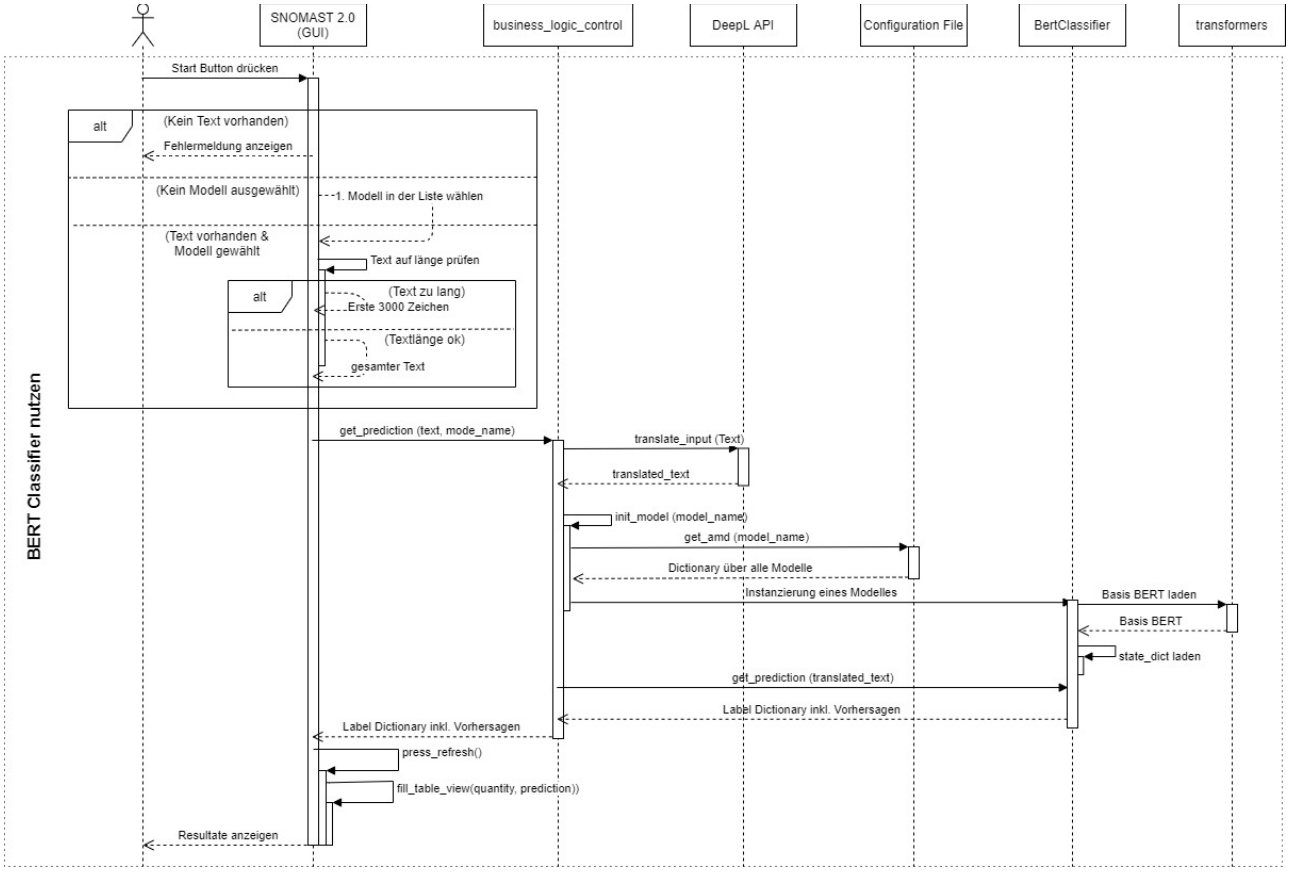

Abbildung 14: Sequenzdiagramm für den Ablauf einer Konzept-Vorhersage

4 Inbetriebnahme

Der Quellcode für das Finetuning und den Prototyp ist auf GitHub in privaten Repositories zu finden. Da der Prototyp mit den entwickelten Modellen von der Datenmenge her zu gross ist, ist zusätzlich ein Projektordner auf Google Drive abgelegt:

Projektbereich	Repository
Finetuning	https://github.com/KunzS85/Finetuning.git
Umsetzung	https://github.com/KunzS85/SNOMAST_2.0.git
	Google Drive

Tabelle 3: Repositories

Um Zugriff auf die privaten Repositories zu erhalten, kann eine Anfrage per E-Mail an kunzs15@bfh.ch oder zgrac1@bfh.ch gesendet werden.

4.1 Anforderungen an das Zielsystem

Die Anforderungen an das Zielsystem werden in zwei Bereiche geteilt. Anforderungen, um ein Finetuning durchzuführen und Anforderungen, welche sich aus der Nutzung des Prototyps ergeben.

Anforderung für das Finetuning
Die Anforderung an das Zielsystem für das Finetuning liegt in einem aktiven Google Konto, Internetzugriff und einem installierten Browser.
Über das Google Konto ist eine Verknüpfung von Google Colab und Google Drive möglich. Eine Integration von Google Drive auf dem Zielsystem ist nicht notwendig, da es sich um einen Cloud-Dienst handelt und die Daten auch über die Website bzw. den Browser heruntergeladen werden können.
Auch Google Colab ist über den Browser zugänglich.
Zusätzlich empfiehlt es sich über das Google Konto ein Abonnement für Google Colab Pro+ zu lösen. Dadurch stehen mehr Server-Ressourcen zur Verfügung.

Anforderung für den Prototyp
Die Anforderungen an das Zielsystem bestehen primär in einem funktionsfähigen Python Interpreter und einem damit verbundenen Paketverwaltungssystem.
Über das Paketverwaltungssystem sollte im Minimum pipenv installiert sein. Zusätzlich benötigt der Prototyp für eine korrekte Ausführung eine Internetverbindung, um Zugriff auf die benötigten APIs zu haben.

4.2 Installationsanleitung

Der Prototyp enthält ein README.md. Dieses Dokument kann mit jedem Texteditor eingesehen werden und enthält folgende Anleitung:

«Die Nutzung des Projektes bedingt, dass auf dem Rechner der anzuwendenden Person ein Python Interpreter installiert ist und ein Packageverwaltungssystem zur Verfügung steht.

Das Projekt wurde unter der Python Version 3.8 entwickelt. Unter dieser Version wurde auch die Lauffähigkeit getestet. Unter Python Versionen > 3.8 sollte der Code jedoch auch funktionieren. Hierfür muss jedoch im Pipfile die Version unter dem Punkt [requires] angepasst werden.

Für die Übersetzung von Texten in die englische Sprache wird ein Authentifikations-Schlüssel von DeepL benötigt. Dieser ist im Python-Script "business_logic_control" unter DEEPL_AUTH_KEY auf Zeile 10 hinterlegt.
Aktuell ist nur ein freier Schlüssel hinterlegt, bei welchem die Kapazitätsgrenze für Übersetzungen zur Neige gehen könnte oder nicht mehr aktuell ist. Die maximale Übersetzungskapazität ist bei einem freien Schlüssel 500'000 Zeichen im Monat. Daher ist es sinnvoll, diesen mit einem eigenen Schlüssel zu ersetzen. Dies kann problemlos in einem Texteditor gemacht werden. Er muss in Anführungszeichen eingeschlossen werden.

Um die virtuelle Packageverwaltung zu nutzen, muss zuerst pipenv global installiert werden. Unter pip ist dies mittels "pip install pipenv" zu bewerkstelligen und muss nur einmalig ausgeführt werden.

Um das GUI zu starten, muss mittels Kommandozeile in den Projektordner navigiert werden.
Im Projektordner kann mit dem Befehl "pipenv shell" die virtuelle Packageverwaltung aufgerufen werden.
In der virtuellen Umgebung werden mittels "pipenv install" alle benötigten Packages und Abhängigkeiten installiert.

Um das GUI zu starten, muss in der virtuellen Umgebung "python snomast.py" ausgeführt werden.

Nun startet sich das GUI auf und steht für die Verwendung zur Verfügung.

Nach der Benutzung, wenn das Fenster des GUI geschlossen wurde, muss mittels "exit" in der Kommandozeile die virtuelle Umgebung geschlossen werden»

4.3 Anwendungsanleitung

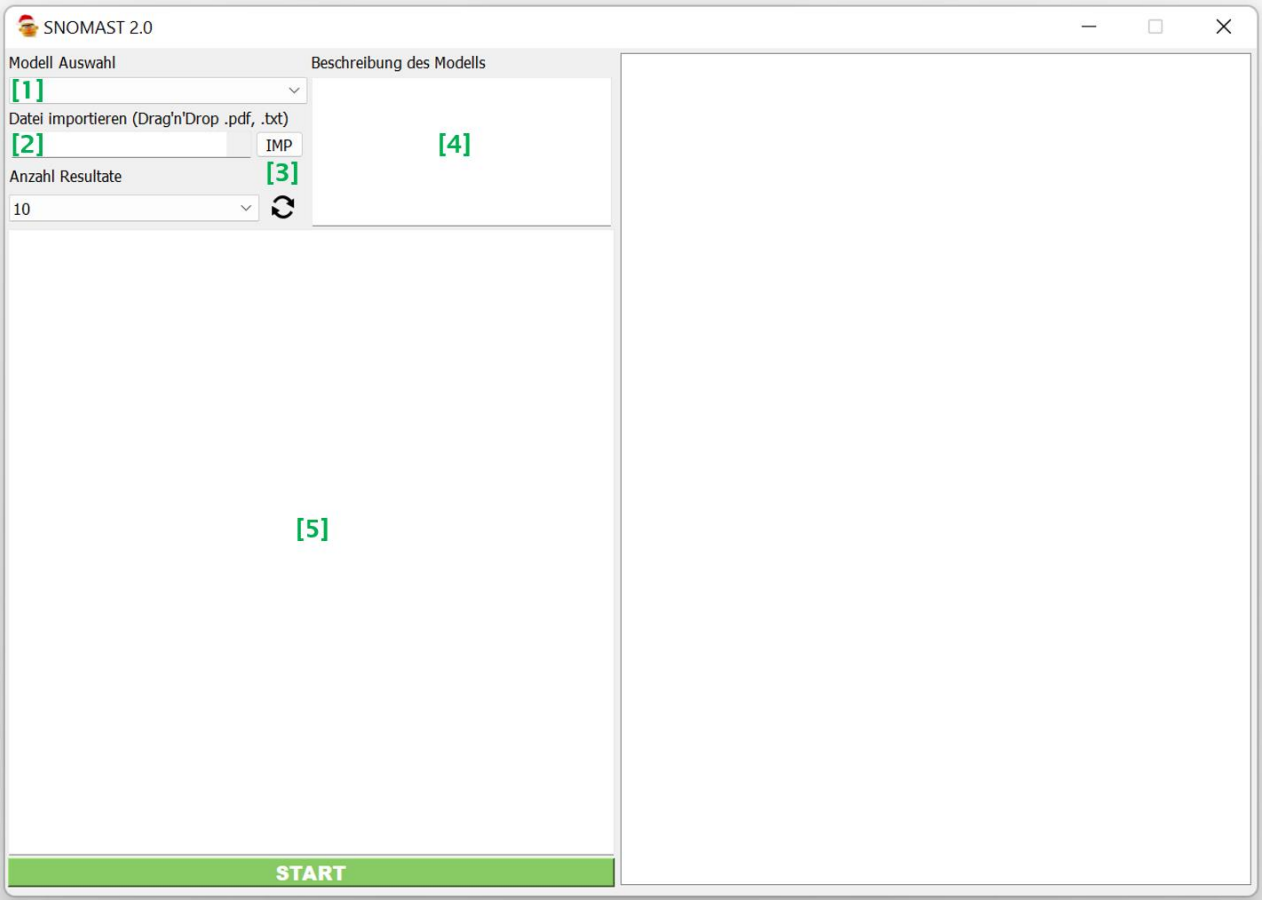

Abbildung 15: Anwendungsanleitung (1)

Nr.	Beschreibung
1	Dropdownmenü für die Wahl eines Modells. Über das Dropdownmenü kann aus der Liste der verfügbaren BERT Classifier gewählt werden. Falls kein Modell bei der Ausführung des START Buttons gewählt wurde, wird automatisch das erste Modell der Liste verwendet.
2	Importfeld für Dokumente. Per Drag&Drop einer Datei in diesem Feld, kann die URL des Dokumentes hinterlegt werden.
3	Import Button. Per Click auf den Button „IMP“, wird der Import des Dokumentes, welches im Importfeld [2] hinterlegt ist, gestartet. Falls kein Dokument hinterlegt ist und der Button betätigt wird, wird im Anzeigefeld [4] eine Fehlermeldung angezeigt.
4	Anzeigefeld für die Beschreibung des Modells. Wenn ein Modell im Dropdownmenü [1] ausgewählt wird, wird automatisch die Beschreibung des Modells angezeigt.
5	Anzeigefeld für Text. Hier kann der Text entweder über die Tastatur eingegeben, der Text über Ctrl + V hineinkopiert oder über den Import angezeigt werden.

Tabelle 4: Anwendungsanleitung (1)

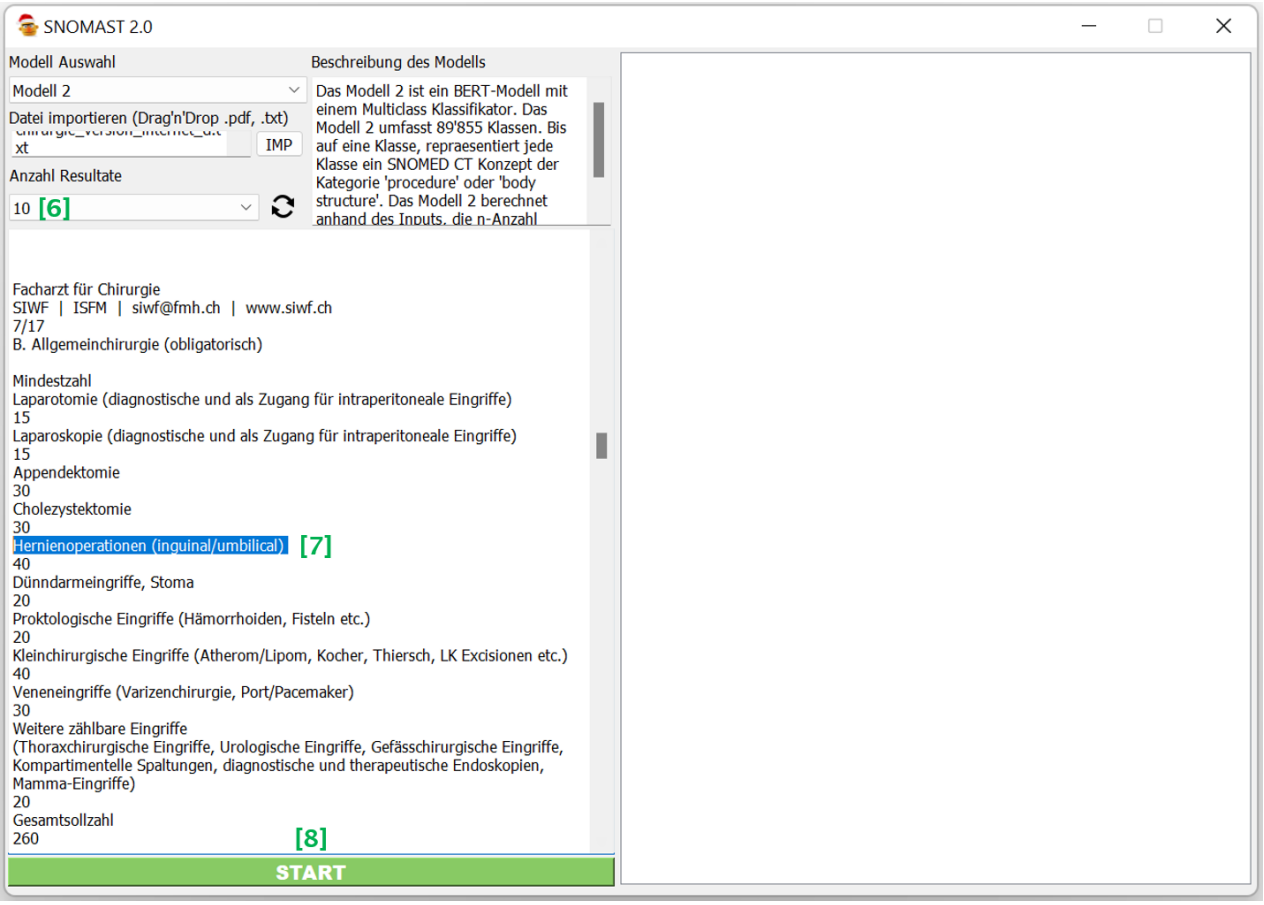

Abbildung 16: Anwendungsanleitung (2)

Nr.	Beschreibung
6	Dropdownmenü für die Auswahl angezeigter Resultate. Über dieses Dropdownmenü kann die Anzahl an angezeigten Resultaten in der Resultaten Tabelle [9] ausgewählt werden.
7	Im Anzeigefeld für Texte [5] kann durch Markieren eines Textteiles dieser Textteil für die Verarbeitung durch das Modell ausgewählt werden. Falls kein Text markiert wurde, werden die ersten 3000 Zeichen des im Anzeigefeld stehenden Textes verarbeitet.
8	Start Button. Durch Betätigen des Start Buttons wird der Text aus dem Anzeigefeld [5][7] mit dem ausgewählten Modell [1] verarbeitet.

Tabelle 5: Anwendungsanleitung (2)

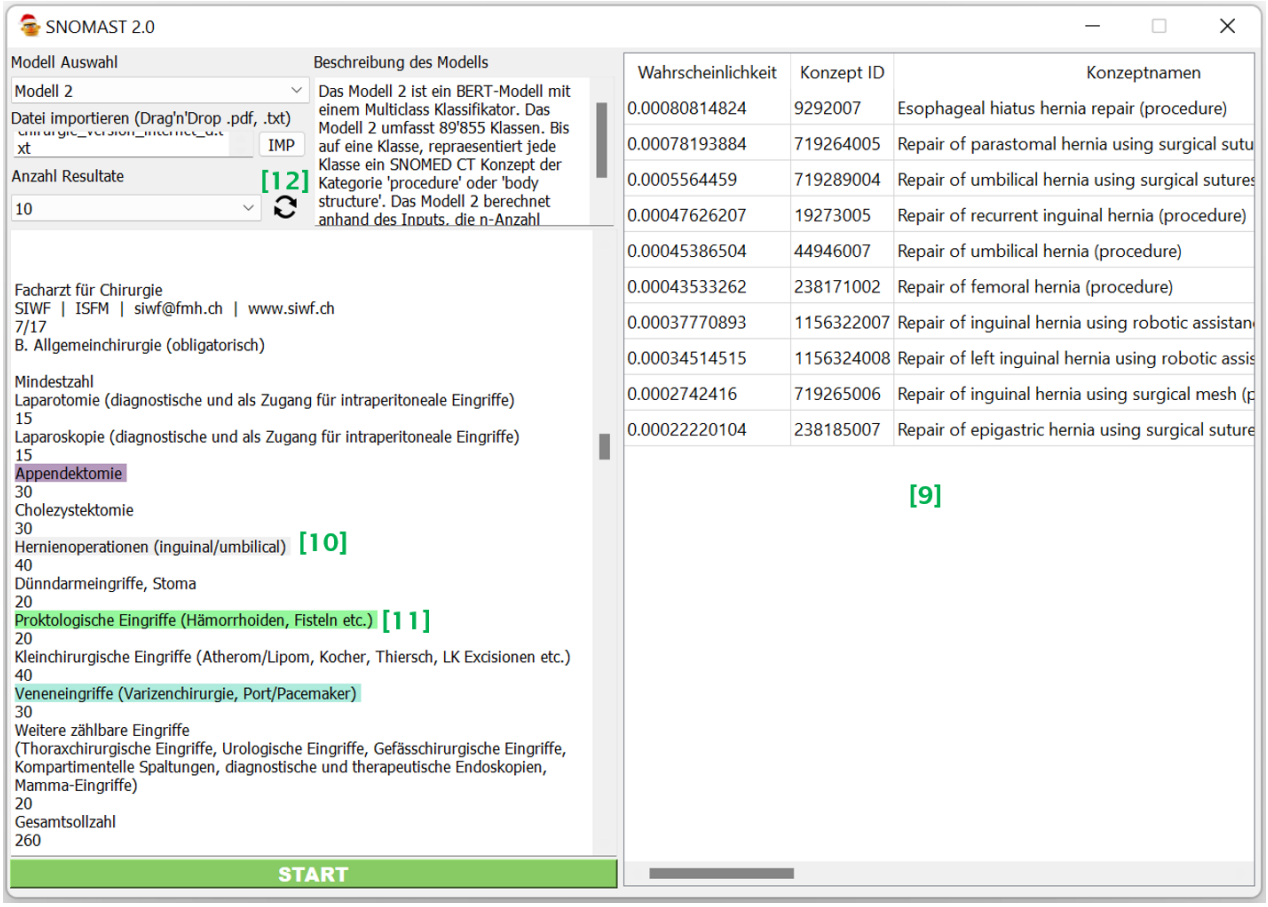

Abbildung 17: Anwendungsanleitung (3)

Nr.	Beschreibung
9	Resultate Tabelle. Die Anzahl Resultate [6] der Berechnung des Textes [10] durch das ausgewählte Modell [1] werden in dieser Tabelle in absteigender Reihenfolge ihrer Wahrscheinlichkeit angezeigt. Zusätzlich enthält die Tabelle die Information um welches SNOMED CT Konzept (Code & Text) es sich handelt, sowie einer Kombination aus Code und Text für die Verwendung in ECL-Queries.
10	Der verarbeitete Text ist gräulich hinterlegt.
11	Textteile, welche zu einem früheren Zeitpunkt verwendet wurden, werden farblich gekennzeichnet.
12	Resultat Refresh Button. Die Anzahl an angezeigten Resultaten der aktuellen Berechnung kann durch verändern der Anzahl im Dropdown [6] und anschliessender Betätigung des Refresh Buttons geändert werden.

Tabelle 6: Anwendungsanleitung (3)

Abbildungsverzeichnis

Abbildung 1: Architektur Finetuning Modell 1	4
Abbildung 2: Finetuning Modell 2 und 3	5
Abbildung 3: Grundlagentabellen erstellen	8
Abbildung 4: In- und Output des Jupyter Notebook «Label Dictionary erstellen»	9
Abbildung 5: Auszug Label Dictonary Modell 2	9
Abbildung 6: Funktion des Label Dictionary	10
Abbildung 7: In- und Output vom Jupyter Notebook «Trainingsdatensatz vorbereiten»	10
Abbildung 8: Funktionsweise des Jupyter Notebook «Finetuning durchführen»	11
Abbildung 9: Funktionsweise des Jupyter Notebook «Evaluation eines Modells»	12
Abbildung 10: Systemarchitektur des PRototyp für die lokale Verwendung	13
Abbildung 11. Auszug aus dem Configuration File des Prototyp	16
Abbildung 12: Komponentendiagramm des Prototyps	17
Abbildung 13: Sequenzdiagramm für Import eines Textes in den Prototyp	17
Abbildung 14: Sequenzdiagramm für den Ablauf einer Vorhersage eines Textes	18
Abbildung 15: Anwendungsanleitung (1)	21
Abbildung 16: Anwendungsanleitung (2)	22
Abbildung 17: Anwendungsanleitung (3)	23

5 Tabellenverzeichnis

Tabelle 1: Verwendete Libraries im Finetuning	7
Tabelle 2: Verwendete Libraries des Prototyp	15
Tabelle 3: Repositories	19
Tabelle 4: Anwendungsanleitung (1)	21
Tabelle 5: Anwendungsanleitung (2)	22
Tabelle 6: Anwendungsanleitung (3)	23

6 Literatur

1. Project Jupyter; 2021 [Stand: 19.05.2022]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Project_Jupyter&oldid=212804667
2. pandas (Software); 2022 [Stand: 19.05.2022]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Pandas_(Software)&oldid=222007563