# C20 - Dev Project Documentation

# Table of Contents

1. [Telephony Insight Assessment](https://paper.dropbox.com/doc/C20-Dev-Project-Documentation--CNYY70f7PYsbOBeTXWA9sBPCAg-9XLZt4hq6fhZsNyRaTnHN#:uid=613983930555513279861310&h2=Customer-Interaction-Analytics)
2. [LLM Performance Evaluation](https://paper.dropbox.com/doc/Cohort-20-Project-Documentation--CNScJDR5lQpnNnSnza1unRsXAg-9XLZt4hq6fhZsNyRaTnHN#:uid=986664061109269602207684&h2=LLM-Performance-Evaluation)
3. [Document Language Simplification](https://paper.dropbox.com/doc/Cohort-20-Project-Documentation--CNScJDR5lQpnNnSnza1unRsXAg-9XLZt4hq6fhZsNyRaTnHN#:uid=192939107693372673971347&h2=Document-Language-Simplificati)
4. [Framework for Physio Analysis using Machine Vision](https://paper.dropbox.com/doc/Cohort-20-Project-Documentation--CNScJDR5lQpnNnSnza1unRsXAg-9XLZt4hq6fhZsNyRaTnHN#:uid=543386019562889461434156&h2=Framework-for-Physio-Analysis-)
5. [Accessibility](https://paper.dropbox.com/doc/Cohort-20-Project-Documentation--CNScJDR5lQpnNnSnza1unRsXAg-9XLZt4hq6fhZsNyRaTnHN#:uid=454265529393268980494190&h2=Accessibility)

# Customer Interaction Analytics: Feasibility Assessment

The full documentation can be found [here](https://paper.dropbox.com/doc/Customer-Interaction-Analytics-Feasibility-Assessment--CNQx~xG3wm82OiyZF7q~ZUvFAQ-jFAtapuVohSw7pOU5NMzq).

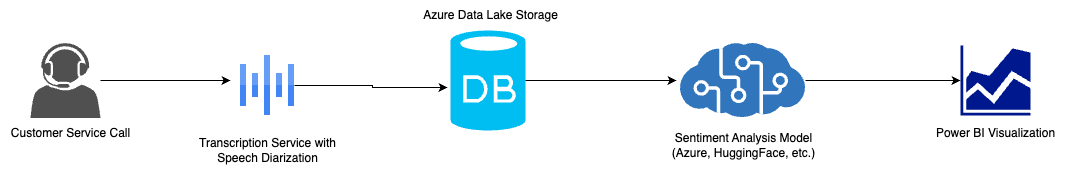
## Overview

The Feasibility Assessment project aims to gain insight through performing sentiment analysis on telephone interactions with customers. This includes converting speech to text, performing speech diarization, and performing sentiment analysis. To determine the most effective method of sentiment analysis, the following 3 technologies were compared:

1. Azure AI Language
2. Hugging Face models
3. Spark Notebooks and Vader

## Implementation

The flow of the project is visualized with the following diagram.



**Speech-to-Text Conversion and Speech Diarization**

Our project involved converting audio calls into text for sentiment analysis. The speech-to-text conversion was achieved using OpenAI’s Whisper, as it as high accuracy and fast processing.

Speech diarization, which is the partitioning of audio based on the speaker, was achieved using pyannote.

The main challenge with pyannote was its better performance on GPUs than CPUs, affecting processing speed. We can counter this by using VMs with NVIDIA graphics cards, improving efficiency but increasing resource requirements.

**Comparison of Sentiment Analysis Technologies**

1. Azure Analysis Architecture

Since the WSIB currently uses Azure systems, this is the framework that most closely aligns with the WSIB’s current tech stack and comfort zones.

There are 4 main components to this framework:

* Azure AI Language for Sentiment Analysis
* Azure Data Lake for storing both raw and analysed data
* Azure App Services to host a Flask API which automatically updates analysis every 24 hours
* Azure Logic App to run the auto-update service

1. Hugging Face Modelling

Many sentiment analysis models outside Azure can be found on Hugging Face.

To conduct sentiment analysis on each sentence in the dialogue, the 2 most downloaded English text classification models were chosen:

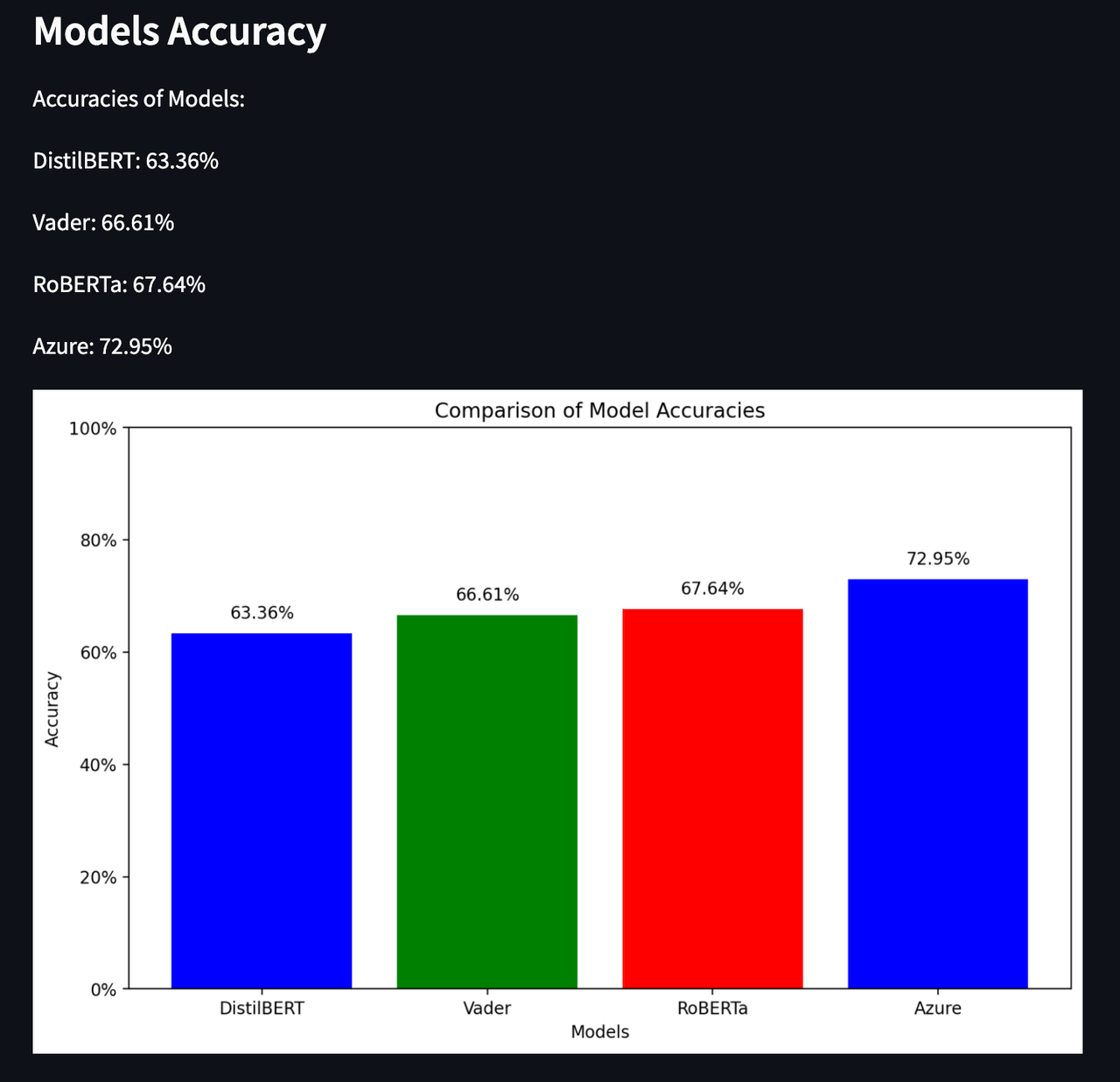
* [Twitter-roBERTA-base](https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest) model
* [DistilBERT-base-multilingual-cased-sentiments-student](https://huggingface.co/lxyuan/distilbert-base-multilingual-cased-sentiments-student) model

1. Apache Spark and VADER

There are 4 main components to this framework:

* Azure Data Lake Storage for data storage
* VADER library to analyze client sentiments
* Azure Data Factory for pipeline execution
* Apache Spark for efficient large-scale data processing

When comparing these 3 technologies for sentiment analysis, it is clear that Azure AI Language is the most accurate in analyzing sentiments.



## Next Steps

Visualization using Power BI

* Currently, the Streamlit Python framework is used to visualize the data through graphs
  + Chosen as it allows rapid development of data science web apps and visualizations
* Had issues with setting up Power BI as we were recently onboarded
* Ideally, the results of analysis are visualized using Microsoft Power BI as it is the tool that the WSIB currently uses

Topic Classification and Opinion Mining

* Although all technologies were able to perform sentiment analysis, they faced challenges with topic classification and opinion mining due to their more intricate nature
* Could research and explore the capabilities of each technology further to implement these 2 analysis strategies across all 3 technologies

# LLM Performance Evaluation

[Making the framework](https://paper.dropbox.com/doc/DnD-Script-Sprint-2--CM1hTv35vEsE0y04VpC14rooAg-X6vRCuoISaDNlh0T7tG90)

[LLM Testing](https://paper.dropbox.com/doc/Decks-and-Demos-Sprint-3--CM31RSg3_f1ebcw9BPv55q8SAg-xQ3ptaUCpDd7tQiR6P1LA)

[+Proposal for Performance Evaluation Framework to Assess LLMs](https://paper.dropbox.com/doc/YJrkspCoKSC1OovlCyW22)

## Overview

The goal of the LLM (large language model) Performance Evaluation project was to construct an evaluation framework to assess LLMs. This framework could be applied to various use cases at the WSIB involving LLMs, but the team focused on the particular use case of a chatbot Q&A assistant when referring to specific documentation. The team also tested the LLMs based on this framework, concluding that

## LLM Performance Evaluation Framework

When it comes to evaluating AI chatbot responses against manual human responses, we have three metrics:

1. **Semantic similarity score**

The semantic similarity score measures the similarity between the responses generated by chatbots to those provided manually. To do this, we can leverage the Hugging Face Sentence Transformer Model, where it utilizes transformer architecture to convert sentences into dense vector representations. These vectors encapsulate the semantic content of the corresponding sentences, where cosine similarity is then used to quantify the similarity between the two vectors. In short, cosine similarity measures the cosine angle between to vectors and outputs a number in range of -1 (completely dissimilar) to 1 (perfectly similar).

1. **Response time of AI to questions being asked**

Response time measure the efficiency and real-time applicability of the chatbot. The approach chosen was to use real-time measurements, which was achieved by implementing a system to record time taken by AI model to generate responses for each query.

2. **Relevance**

Relevance refers to the alignment of the information provided in the response with the data in the source material. It checks for any instances of ‘hallucination’ or deviation from the source, confirming that the output of the LLM is factually accurate according to the documents provided. Regarding tools for evaluation there are various options from LangChain and LlamaIndex as well as Retrieval Augmented Generation Assessment framework designed for evaluating RAG pipelines in LLM applications.

## Models to Test

In order to test our framework, the team compiled a list of 13 LLMs. 3 factors were considered when compiling this list:

1. Whether the model had API access or was downloadable
2. Integration with LangChain.
   * LangChain is a framework to make working with LLMs easier, so we omitted any models without direct LangChain capabilities.
3. If the model was available for use in Canada.
   * Some models were not available in Canada to due the country’s strict AI regulations. See [Problem Areas](https://paper.dropbox.com/doc/C20-Dev-Project-Documentation--CNYY70f7PYsbOBeTXWA9sBPCAg-9XLZt4hq6fhZsNyRaTnHN#:uid=731746082673105476030801&h2=Problem-Areas) for more.

The full list of models tested can be found [here](https://paper.dropbox.com/doc/Proposal-for-Performance-Evaluation-Framework-to-Assess-LLMs--CNas5cjyKek2uq2DIOyJYAArAg-YJrkspCoKSC1OovlCyW22#:uid=203296993468485037427458&h2=Models-to-Test).

## Test Datasets

When it comes to datasets for evaluating the performance of the LLMs, we have three main options:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Creating datasets manually | Creating datasets with LLMs | Using pre-built datasets |
| **Pros** | * Best quality of data (custom) | * Fast to create, can automate with scripts * Less time and resources than manual creation | * Least time and resource intensive * Access to larger datasets usually |
| **Cons** | * Difficult * Resource and time intensive | * Not as accurate as manual creation | * Least accurate |

## Testing the Framework

To implement this framework, there are 4 main parts to the workflow:

1. Scrape the corresponding Wikipedia articles from the *Natural Questions Dataset* using the BeautifulSoup integration in LangChain.
2. Vectorize the article content and store the embeddings in a vector store using a VectorStore and Embeddings model from LangChain. The Embeddings model will be standardized to reduce variable in the outputs.
3. Set up the Retriever and Chat Models in LangChain, and generate responses using each model for each question in the dataset.
4. Store outputs in a database, such as Azure Data Lake, and evaluate the outputs using evaluation criteria.

To test the various models, there are 2 approaches:

* For models that have integrations with LangChain, they will be implemented into the LLM Chain using the LLM or corresponding Chat Model object.
* For models that do not have integrations with LangChain but are available on Hugging Face, they will be downloaded and run on a VM hosted on Azure. They will be implemented using the Hugging Face LLM classes or the ChatHuggingFace class.

## Problem Areas

1. **Dataset had answers in a weird format that we couldn’t parse**
2. When checking the dataset, the visualization of the dataset used a custom Data Browser service to parse the answers. The answers were thus stored in a confusing tokenized format. We were unable to reverse engineer the process used to tokenize the answers, and as such, we did not have long answers we could use to compare for the evaluation.

1. **Availability of models in Canada**
2. Due to Canada’s strict regulations on artificial intelligence technologies, many LLMs were not made available in Canada at the time of testing. Due to the legal hurdles that companies must maneuver to introduce their AI products on the market, many popular LLMs that the team wanted to test were not available. This includes PaLM 2, Claude, and Amazon Q.

1. **Slow loading and inferencing of models**
2. Since the majority of the LLMs tested did not have API support, the team had to load the models from Hugging Face directly using the built-in Transformers library and pipelines. However, this proved to be a very time-consuming process, as these were very massive models. Some took upwards of 2 hours to load, which reduced the productivity and efficiency of the testing process significantly.

# Document Language Simplification

## Overview

The Document Language Simplification project aims to simplify WSIB communications to adhere to the Ontario standard of a Grade 6 reading level. Users can upload Microsoft Word documents or paste text, customize a prompt, and add guidelines for the simplification, which is done using Azure OpenAI’s GPT-4 model.

The application is hosted in the Azure environment [here](https://nice-desert-0adef170f.5.azurestaticapps.net/).

## Implementation

There are 5 main components to the framework:

1. **Azure Data Lake storage account**

The Azure Data Lake storage account called docsimplstorage holds 2 containers, files and simplified-files.

The “files” container holds a JSON file with the base prompt, keywords to replace, keywords to keep, and sample simplifications, all of which are fed to the model. It also holds the original files that are uploaded by the user.

The “simplified-files” container holds the simplified .docx and .pdf files.

Azure Data Lake storage was chosen as it aligns closely with WSIB’s Microsoft ecosystem.

1. **AI Studio deployment of a GPT4-32k model**

The OpenAI GPT-4-32k model is called doc-simpl-openai on Azure.

This model was chosen as it is the newest model with the largest context window of 32k. This means it was trained on a recent and large dataset.

1. **Python Flask API**

The python flask API connects the frontend, storage account and deployed model.

It is responsible for the following transactions:

* Uploading original .docx file to Azure Data Lake Storage
* Feeding original text to GPT-4 model for simplification
* Receiving simplified text from GPT-4 model
* Uploading the simplified .docx and .pdf files

Additionally, it uses the python-docx library to edit the original .docx files by replacing the text with the simplified versions. This library was considered as it allows manipulation of .docx files and can be easily integrated into the existing API. However this solution can be substituted with the ASP.NET Core Web API.

1. **ASP.NET Core Web API**

The ASP.NET Core Web API uses the Office Open XML SDK to edit the original .docx files by replacing the text with the simplified versions. It has the same function as the python-docx solution. This SDK was considered as it is part of Microsoft’s official offering, and the WSIB could access support from Microsoft readily.

**Comparing python-docx vs. Office Open XML**

|  |  |  |
| --- | --- | --- |
|  | **python-docx** | **Office Open XML** |
| **Pros** | Can integrate in existing flask API, no separate hosting or maintenance required | Uses official Microsoft-developed SDK, can access support from Microsoft |
| **Cons** | No accessible support | Requires separate hosting and maintenance on Azure |

Both solutions were explored in the PoC and are hosted on Azure.

1. **Frontend on Azure**

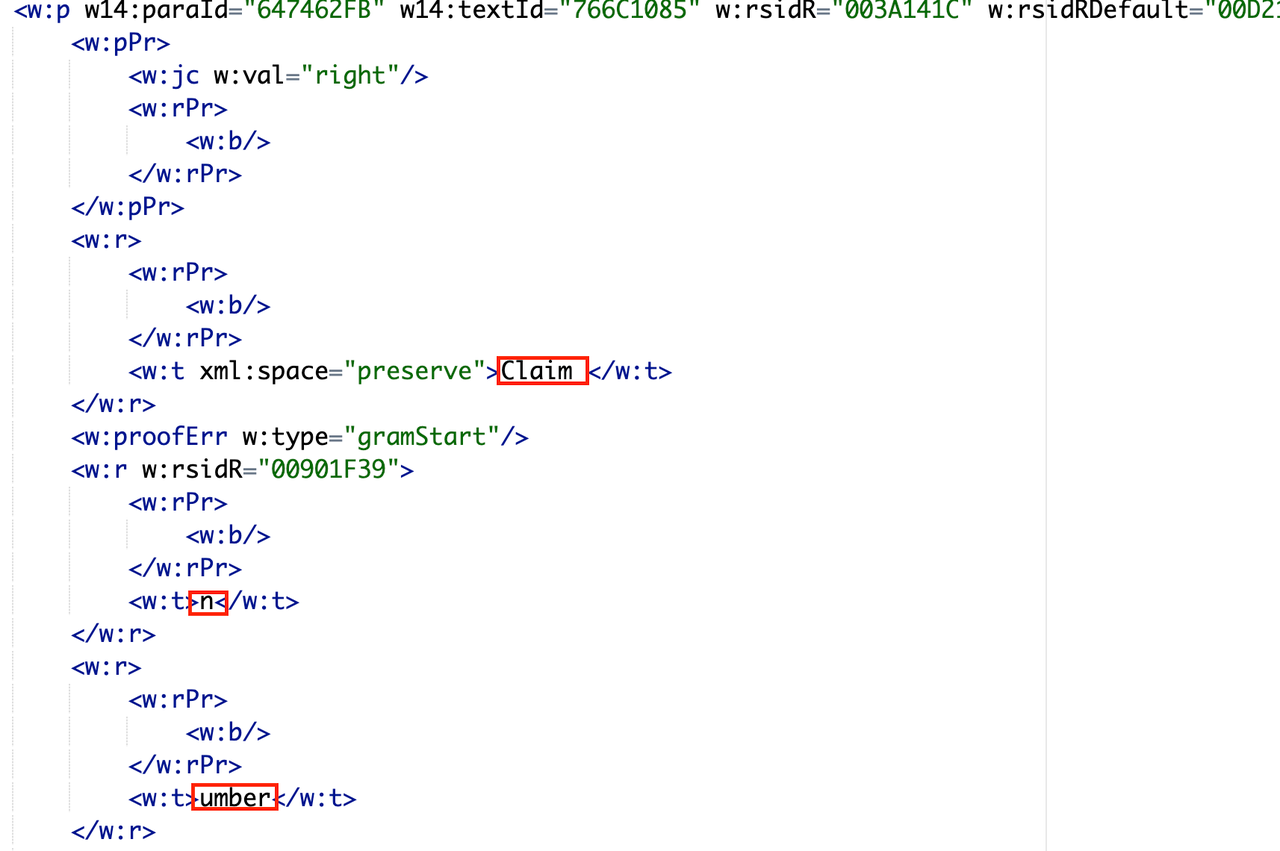
The frontend was developed using Vite. Vite was chosen as it optimized for fast web development projects.

The frontend uses the styled-components library to implement the designs of the design team. Dev Mode on Figma was very helpful in pinning down the exact CSS styling to apply.

## Problem Areas

1. **Maintaining the original formatting of the .docx file**

In XML, paragraphs are divided into runs, which can contain fragments of text, instead of full sentences. This made extracting and replacing text difficult as each run could have different styles that must be maintained (e.g. a portion of a sentence could be bolded).



1. **Creating a prompt**

Crafting a prompt is crucial in the usefulness of the GPT-4 model. Context and clear instruction must be provided.

There were several iterations of the prompt but the most effective changes were:

* Few-shot prompting, where example simplifications are provided to guide the model and allow it to learn from the examples.
* Telling it to return the original text if it cannot be simplified further. Previously, it returned an error message instead of the simplification, but since the API cannot tell the difference between this error and an actual simplification, these error messages were being added to the document.

1. **File Preview**

It is crucial that the user is able to preview the simplified document before downloading it.

Since no browser can render Word Documents, external libraries must be used. We explored:

* react-doc-viewer
* cyntler/react-doc-viewer
* react-file-viewer
* iframe

However, all these libraries require the uri to be publicly accessible. Although the Azure Data Lake storage blobs can be configured to have a public uri, this is not feasible for confidential WSIB documents.

We pivoted to converting the word documents to pdf format, as PDF files do not require a public uri. The cyntler/react-doc-viewer library was used to preview the documents.

1. .**docx to .pdf Conversion**

All the tools explored to convert Word Documents to PDF format were paid tools. These included:

* [PSPDFKit](https://pspdfkit.com/)
* [Apryse](https://apryse.com/)
* [Aspose](https://metrics.aspose.com/)

Aspose’s free version was used for the PoC, although it included heavy watermarking. It also failed to retain much of the original formatting. Future improvements would be to invest in a high-quality docx to pdf conversion tool.

1. **Hosting on Azure**

## How to access

**Running the frontend**

[Github Repository](https://github.com/WSIB-Innovation/document-simplification-frontend)

* Requirements
  + Node.js version 18+, 20+

1. Install dependencies

$ npm install

1. Run app

$ npm run dev

**Running the backend**

[Github Repository](https://github.com/WSIB-Innovation/document-simplification-backend)

* Requirements
  + Python3.8 or higher

1. Create virtual environment

$ python -m venv venv

$ source venv/bin/activate/

1. Install dependencies

$ pip3 install -r requirements.txt

1. Add environment variables

AZURE\_OPENAI\_ENDPOINT=

AZURE\_OPENAI\_API\_KEY=

DATALAKE\_ACCOUNT\_URL=

DATALAKE\_ACCOUNT\_KEY=

1. Run

flask run

# Physio Analysis using Machine Vision

[+DnD Script - Sprint 4](https://paper.dropbox.com/doc/84CLZnFe61wqA1P05mmC9)

[+Framework for Recognizing and Rating Physiotherapy Motions using Machine Vision](https://paper.dropbox.com/doc/3FOiCHZHdcJjgRbYwyg4Q)

## Overview

This proposal looks into an overall foundational framework for monitoring and evaluating musculoskeletal rehabilitation exercises. The goal was to develop a comprehensive framework for developing a Proof-of-Concept that evaluates the execution of a patient’s mandated physiotherapy based on a video feed, either pre-recorded or live.

## Framework

The framework has 5 components:

1. **OpenCV**

* Primary computer vision library to process visual data
* Enables the extraction of relevant information from video recordings

1. **MediaPipe**

* Pose estimation library, ideal for detecting joint based on a video feed
* Used to establish human body model points, facilitating the tracking of key joints and movements during rehabilitation exercises.

1. **Comparison Algorithm**

* Multilayer dense neural network classification algorithm chosen
  + Simple to implement
* Developed to assess the deviation of observed body model points from a dataset representing "ideal" points, allowing for quantitative evaluation of exercise performance
* Comparison variables (body model points):
  + Max elbow angle
  + Min elbow angle
  + Average elbow angle
  + Max knee angle
  + Min knee angle
  + Average knee angle
  + Max hip angle
  + Min hip angle
  + Average hip angle

1. **Scoring Mechanism**

* Quantifies the similarity between the observed body model points and the ideal dataset, providing a numerical assessment from 0-1 of the patient's rehabilitation performance

1. **Visualization of Results**

* Matplotlib, plotting library, was used to visualize the performance of a patient doing physiotherapy motions

The full framework can be found [here](https://paper.dropbox.com/doc/Framework-for-Recognizing-and-Rating-Physiotherapy-Motions-using-Machine-Vision--CNfpbdhupPZoN9s8d2mW5c0wAQ-3FOiCHZHdcJjgRbYwyg4Q).

## Problem Areas

1. **Dataset sourcing**

* Dataset [used](https://www.kaggle.com/datasets/mohamadashrafsalama/pushup) had mostly videos of the side view, which made evaluating other views (e.g. front) difficult and more inaccurate
* Dataset was very small with ~98 videos, providing less training data for the model, resulting in lower accuracy
* Since the videos were all of different lengths, we could not use all the angle measurements from the whole motion, instead opting to use the max, min, and average

## Next Steps

1. **Expand dataset**

* Either create primary data by recording pushups or finding more videos of pushups
* Allows model to increase learning, resulting in higher accuracy
* Use datasets with multiple angles or multiple cameras to increase accuracy across a wider variety of camera positions

## How to access

[**GitHub Repo**](https://github.com/WSIB-Innovation/mv-physio)

Requirements

* Python 3.11 or higher

1. **Create and activate a virtual environment**

$ python -m venv venv

$ source venv/bin/activate

1. **Install dependencies**

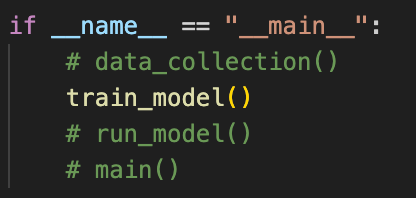
$ pip install -r requirements.txt

1. **Train model**

Uncomment train\_model.

Comment the other 3 function calls.

$ python3 main.py

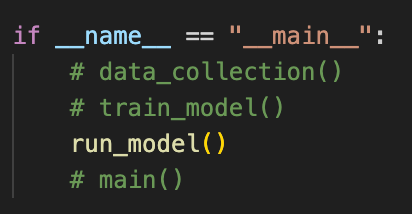


1. **Run model**

Uncomment run\_model.

Comment the other 3 function calls.

$ python3 main.py



1. **Run main**

Uncomment main()

Comment the other 3 function calls.

$ python3 main.py

# Accessibility