NExt ApplicationS of Quantum Computing

D6.10: WP6 QNLP Report

Document Properties

Contract Number	951821
Contractual Deadline	M38 (27/10/2023)
Dissemination Level	Public
Nature	Other
Editors	Richard A. Wolf, ICHEC Pablo Suárez Vieites, ICHEC Conor Dunne, ICHEC Yanis Lalou, ICHEC Pablo Lauret Martínez de Rituerto, ICHEC Daiga Deksne, Tilde Mārcis Pinnis, Tilde Emil Dimitrov, ICHEC Venkatesh Kannan, ICHEC
Authors	Richard A. Wolf, ICHEC Pablo Suárez Vieites, ICHEC Conor Dunne, ICHEC Yanis Lalou, ICHEC Pablo Lauret Martínez de Rituerto, ICHEC Daiga Deksne, Tilde Mārcis Pinnis, Tilde Emil Dimitrov, ICHEC Venkatesh Kannan, ICHEC
Reviewers	Andrés Gómez, CESGA Vedran Dunjko, ULEI
Date	31/10/2023
Keywords	Quantum Natural Language Processing, prototype
Status	Final
Release	0.3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

History of Changes

Release	Date	Author, Organisation	Description of Changes
0.0	09/08/2023	Conor Dunne (ICHEC)	Template initiated
0.1	04/10/2023	Richard A. Wolf (ICHEC), Pablo Suárez Vieites (ICHEC), Pablo Lauret Martínez de Rituerto (ICHEC), Yanis Lalou (ICHEC), Mārcis Pinnis (Tilde), Conor Dunne (ICHEC), Emil Dimitrov (ICHEC), Daiga Deksne (Tilde), Venkatesh Kannan (ICHEC)	Submitted for first review
0.2	17/10/2023	Richard A. Wolf (ICHEC), Pablo Suárez Vieites (ICHEC), Pablo Lauret Martínez de Rituerto (ICHEC), Yanis Lalou (ICHEC), Mārcis Pinnis (Tilde), Conor Dunne (ICHEC), Emil Dimitrov (ICHEC), Daiga Deksne (Tilde), Venkatesh Kannan (ICHEC)	Submitted for second review
0.3	27/10/2023	Richard A. Wolf (ICHEC), Pablo Suárez Vieites (ICHEC), Pablo Lauret Martínez de Rituerto (ICHEC), Yanis Lalou (ICHEC), Mārcis Pinnis (Tilde), Conor Dunne (ICHEC), Emil Dimitrov (ICHEC), Daiga Deksne (Tilde), Venkatesh Kannan (ICHEC)	Final version

Table of Contents

1. Introduction

The NEASQC project aims at showcasing and advancing the capabilities of *noisy intermediate-scale quantum* (NISQ) era algorithms through the development of practically relevant use-cases. Under the symbolic AI and graph algorithms work package WP6, the use-case WP6.1 that we present here focuses on hybrid *quantum natural language processing (QNLP)*. The objective of this use-case is to investigate and develop hybrid classical-quantum approaches to *natural language processing* (NLP) and compare their performance with classical counterparts. In this deliverable D6.10 we present a series of variations of former algorithms and their application to the task of binary sentiment analysis.

At its core, a large part of D6.10 is centered around the task of sentence classification which, together with clustering, is pivotal to a wide range of applications including sentiment analysis (Wankhade et al., 2022), intent detection (Weld et al., 2022) and language identification (Burchell et al., 2023). The ability to classify and cluster sentences effectively is not only vital for streamlining information retrieval and content organisation, but is also a key application in fields such as healthcare (Demner-Fushman et al., 2009), finance (Kalra & Prasad, 2019), government policy (Jin & Mihalcea, 2022) or in certain versions of recommendation systems (Al-Ghuribi & Noah, 2021; Rich, 1979).

The project aims to investigate possibilities and limitations of hybrid quantum-classical methods for such tasks. Due to current hardware limitations, it seems unlikely that quantum methods could present major computational or accuracy benefits over their existing high-performing classical counterparts. This project aims at investigating the possibility of using hybrid classical-quantum NLP approaches on near-term hardware. Thus, we have worked with low numbers of qubits in line with the capabilities of such devices, and we plan to introduce noise into our models in future.

This report begins by laying out the theoretical foundations that underpin our work in Sec. 3, followed by a description of our models and the datasets used in Sec. 4. The results and details of our experiments are then displayed in Sec. 5 and subsequently analysed and discussed in Sec. 6. The report then ends with a discussion of the limitations of our models and future work in Sec. 7, followed by a bibliography and an appendix (Sec. 8) with further information on the theoretical aspects and user instructions for running the discussed models.

2. W.P.6.1 status updates

2.1. General

Various software and model updates have been implemented and developed in this deliverable D6.10. All of which can be found on the project's github. Firstly, the models corresponding to D6.3, *pre-Alpha*, and D6.7, *Alpha*, have been updated and expanded. Recent updates to the lambeq library have been integrated into both models. The latest version of these respective models can be found in the `dev` branch of the online Github repository. Secondly, a *Beta* model (D6.10) has been developed that uses a quantum version of the k-nearest neighbours algorithm (Fix & Hodges, 1989) to categorise sentences. The details and sample results of these models are discussed in the next sections.

2.2. Models

2.2.1. Pre-Alpha models

The pre-Alpha model has now been split into two sub models. One, which will be named from now on *pre-Alpha 1*, uses the same software libraries as before with the addition of two new features: allowing to change the random seed for the initial variational parameters of the quantum circuits in the optimisation process and to compute predictions during the optimisation process (before only the loss was computed). The second, named *pre-Alpha 2*, integrates the lambeq library which gives a streamlined pipeline for representing a sentence's grammar as a quantum circuit. It also makes use of the pennylane library for quantum computation. The latest versions of pre-Alpha 1 and pre-Alpha 2 can be respectively found here and here.

2.2.2. Alpha models

In order to explore the possibilities of the general approach of Alpha, the main model has been divided into three models that follow a dressed quantum circuit architecture. These models can be found in the repository and are named Alpha 1, Alpha 2, and Alpha 3.

2.2.3. Beta 1 model

A so-called *Beta 1* model has been implemented which classifies sentences using a quantum version of the k-nearest neighbours algorithm that uses a *swap test* (Lloyd et al., 2013) to compute the distance.

2.2.4. Classical models

To have a reference against which to compare the quantum solution, the following classical classifiers were implemented:

1. Feed-forward neural network classifiers from the TensorFlow package applied to BERT or sentence transformers embeddings.
2. A convolutional neural network classifier from the TensorFlow package applied to FastText embeddings.

3. Background and approach

3.1. Background

QNLP has emerged recently as a new area of research that aims to use *quantum computing* (QC) for NLP tasks. *DisCoCat*, a category-theory framework for QNLP was first introduced in (Abramsky & Coecke, 2004) closely followed by (Abramsky & Coecke, 2008) and (Coecke et al., 2010). This model proposes a unification of the vector space models of meaning (distributional models (Schütze, 1998)) and the models that take into account the grammar of the sentences using the algebra of *pregroups* (compositional models)(Lambek, 2008) for representing the meaning of a sentence.

The DisCoCat computational paradigm follows three steps in order to calculate a sentence's meaning (Coecke et al., 2010) :

1. Compute the tensor product of the vectors $\vec{w_i}$ which represent the meaning of each word in the sentence. This results in a vector $\overrightarrow{words} = \vec{w_0} \otimes ... \otimes \vec{w_k}$.
2. Construct a linear map that represents the grammatical structure of the sentence using Lambek pregroups.
3. Compute the meaning of the sentence by applying the linear map to the $\overrightarrow{words}$ vector.

The formulation of the DisCoCat model in terms of compact closed categories means that there exists a structural relation with quantum theory (Abramsky & Coecke, 2004). In fact, in this model, sentence representations can be seen as tensor networks (Orús, 2014).

An algorithm to implement the DisCoCat model for QNLP tasks was first proposed in (Zeng & Coecke, 2016). In this work, it is the vector representation of the DisCoCat model that is used as the input for quantum clustering algorithms (Wiebe et al., 2014) for computing sentence similarity. The main drawback of this algorithm is that it assumes the use of quantum memory such as QRAM (Giovannetti et al., 2008), making it non NISQ-friendly.

Nevertheless, the ideas presented in (Zeng & Coecke, 2016) can still be used to implement the DisCoCat model in current NISQ devices. This is done by using quantum variational algorithms. The *Ansatz* (plural: *Ansätze*) for these algorithms (i.e. the set of quantum gates containing adjustable parameters that acts on the initial quantum state) can be chosen so that it resembles the vector product structure of the DisCoCat model. Apart from being implementable on NISQ devices, variational quantum algorithms offer the possibility of using a new feature space (Havlıček et al., 2019) in which to encode the DisCoCat model.

A description of the full-stack pipeline for implementing QNLP algorithms for classification tasks in NISQ devices can be found in (Meichanetzidis et al., 2020) and (Lorenz et al., 2023). We briefly summarise their steps here :

1. Parse sentences to diagrams that represent the DisCoCat derivation of the sentence[1].
2. Rewrite the diagrams. The structure of compact closed categories in DisCoCat allows one to make use of rewrite rules, which are a series of operations that when applied to a diagram yield a simplified equivalent diagram. These rewrite rules thus help minimise the depth of a circuit and the number of gates, as well as far-off connectivity to a certain extent.
3. Choose the Ansatz for our variational quantum circuit that resembles the entanglement structure of the DisCoCat representation. The Ansatz achieves this via the use of Hadamard and CNOT gates. Additionally, the Ansatz will also introduce rotation gates, whose corresponding rotation angles will be the variational parameters to optimise in our model. The choice of Ansatz can be varied depending on the gates we want to use, and on what type of qubit connectivity we want in our circuit. A description of different circuit architectures, including measures of expressivity and entangling capability can be found in (Sim et al., 2019).
4. Use a quantum compiler to translate the quantum circuit into machine-specific instructions.
5. Run the circuit on a quantum computer for a given number of shots.

[1] This can be done using tools like NLTK (Bird et al., 2009) or SpaCy (Honnibal & Johnson, 2015).

6. Compute the predicted labels for our sentences based on the results obtained. The predictions will be made based on post-selection measurements on some subset of the qubits of the circuit, meaning that some post-processing will need to be done.

An image showing the described pipeline can be found in fig. 1.

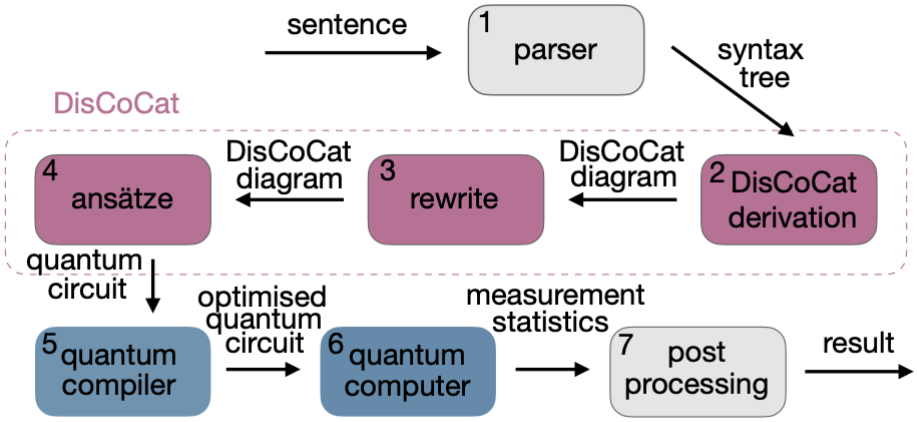

***Figure 1**: Diagram with the general pipeline of a QNLP algorithm for classification tasks. Image taken from (Lorenz et al., 2023)*

3.2. Approach

Here we employ the work of (Abramsky & Coecke, 2008; Coecke et al., 2010) to encode sentence grammar into a parameterised quantum circuit (PQC). By using controlled gates and Hadamard gates, one can introduce superposition and entanglement properties to a quantum circuit, making it possible to leverage the DisCoCat framework. These possibilities are however constrained by the current limitations of NISQ computers.

Based on those limitations, the proposed task of binary sentiment analysis for short sentences of less than 6 words was addressed.

This work presents a total of 6 hybrid classical-quantum models used to perform binary sentiment analysis. Pre-Alpha 1 and 2 served as precursors to the Alpha models, and implement a sentence classification model based solely on the PQCs associated with each sentence in the dataset. These models, however, are limited in their use as they can only classify sentences containing words they have already been trained on. Alpha 1 and 2 extended this model to combine associated BERT word/sentence embeddings for each sentence with the same PQCs, which allowed to handle words in validation and test that have not been seen during training, thus addressing the limitation of the pre-Alpha models.

Alpha 3 uses BERT sentence embeddings with a common quantum circuit independent of the input sentence. This model is the most similar to a classical neural network.

Finally Beta 1 employs an alternative approach that uses a quantum nearest neighbours algorithm on dimensionality-reduced BERT sentence embeddings to perform clustering. This algorithm is akin to the classical nearest neighbours except the for Euclidean distance which is computed with a quantum circuit thanks to the swap test (Kang et al., 2019). This process should, in theory, be computationally less intensive than using a classical calculation of the Euclidean distance for large datasets (Sarma et al., 2019).

4. Experiments

4.1. Tasks

Considering the current limitations of quantum computing methods, such as limitations on the size of potential quantum circuits, short text classification tasks were chosen, such as short text sentiment analysis and short text domain classification.

4.1.1. Binary sentiment analysis

Sentiment analysis (Wankhade et al., 2022) is an NLP technique that aims to extract subjective information from human-written text.

One of the tasks analysed in the field of sentiment analysis is what is called the study of the *polarity* of a text. It consists of computationally determining how positive or negative are the connotations of a given block of text towards a given entity or topic. This has been historically made possible by identifying rules, patterns and the presence of specific words in a piece of text that is being analysed (e.g., (Hutto & Gilbert, 2014)). However, current methods rely on artificial neural network classifiers that have been trained using supervised or semi-supervised machine learning methods (e.g., (Thakkar & Pinnis, 2020)). Recently, large language models have been shown to achieve state-of-the-art for sentiment analysis (e.g., (Wu et al., 2023; W. Zhang et al., 2023)).

For the task at hand, the focus will be on binary classification of the polarity of short sentences. The dataset used will consist of Amazon review titles, so the polarity of a customer's opinion towards some purchased product is analysed. This means that the objective will be to discern if a given title of a review is expressing a positive or negative feedback of the product. Having this into account, if 0 and 1 are assigned for negative and positive sentiments, one would expect for the sentences *"This is a horrible book"* and *"I love this TV show"* a prediction of 0 and 1 respectively. In the next sections, it will be detailed how different models use information from the sentences in order to perform the task of binary sentiment analysis.

4.2. Hybrid NISQ-classical models

4.2.1. Pre-Alpha 1

The pre-Alpha 1 model follows the work and ideas described in sec. 3.2 and sec. 3.1. This model was already implemented in this work package (Villalpando et al., 2021) in a previous deliverable. The algorithmic structure of the model is described below.

The first step is associating each word appearing in the dataset with some set of random initial variational parameters. For each sentence, the parameters associated to each word appearing on it will be injected into a PQC. This circuit will have an entanglement structure based on the grammar of the sentence, as dictated by the DisCoCat formalism. As dictated by the diagrammatic calculus of compact closed categories (Coecke & Kissinger, 2018), each word will be assigned a type. Namely, the noun type N, sentence type S, or some combination of both. The number of qubits that are assigned to each type (q_n and q_s) in the PQCs can be adjusted.

In fig. 2 the quantum circuit that pre-Alpha 1 generates for the sentence *"John likes Mary"* is shown. For this example $q_n = q_s = 1$. The two first and two last qubits represent the word *John* and *Mary* and their associated parameters are θ_i and ψ_i respectively. The word *likes* is type $N^r S N^l$ (we will describe what this means later in sec. 4.4.1), and that is why it is represented by middle qubit in the PQC, having the parameters Φ_i associated. Note that if a word appears on different sentences its parameters will not vary, i.e. the parameters θ_i associated to *John* will be the same if the word appears in a different sentence.

It can be seen in fig. 2 that only the third qubit is being measured. This is the sentence qubit of the circuit. The sentence qubit corresponds to the predicted class of the sentence. The sentence qubit is the only degree of freedom used to encode the classification information, all other qubits are *post-selected* on the state 0. 0 is selected because of the way qubit values are initialised by default in the models, but typically post-selection could be performed on any value that is consistent with the encoding. A box with

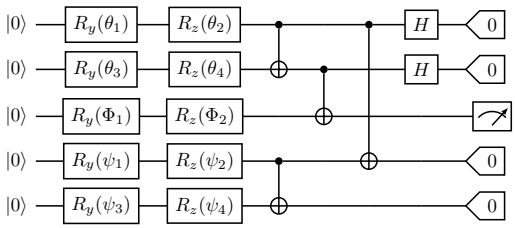

***Figure 2**: Pre-Alpha 1 variational quantum circuit for the sentence "John likes Mary"*

a 0 was drawn in the rightmost part of post-selected qubit wires of fig. 2 to emphasise this post-selection procedure. Thus, for this case only the states $|00000\rangle$ and $|00100\rangle$ are being analysed, and the prediction for the sentence will be the value of the sentence qubit in the state that is measured with the highest occurrence in our experiments. A pipeline flowchart showing the different steps in pre-Alpha 1 model is shown in fig. 3.

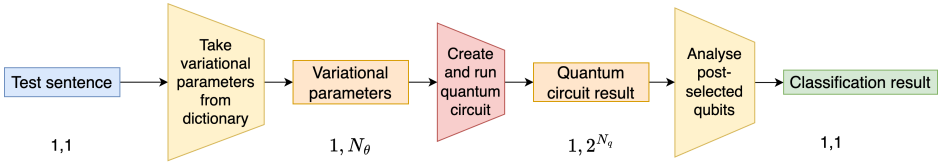

***Figure 3**: Pre-Alpha 1 pipeline*
The numbers shown under the rectangles represent the vector dimension. N_θ is the number of variational parameters of the circuit and N_q the number of qubits

The algorithm for the training stage of the pre-Alpha 1 model is displayed in alg. 1. Let S be the sentences in our dataset (with the suffixes tr, va and te, for the train, validation and test datasets), Y the list of labels (lowercase y will denote one element of the list), r the initial random seed for the variational parameters, O a classical SciPy optimiser, $epochs$ the number of epochs for the optimisation and $L(\cdot,\cdot)$ the *binary cross-entropy* loss (BCE) (Shannon, 1948). Also, θ_i is used for denoting the vector containing the variational parameters for all words of the dataset in step i and $\theta_{s,i}$ for the vector containing the variational parameters at step i that are present in sentence s.

Algorithm 1 Pre-Alpha 1

Input: $(S, Y, r, O, epochs)$
  Set r to create initial variational parameters θ_0
  Store θ_0 in a dictionary d
  for i in epochs **do**
    $loss \leftarrow 0$
    for s in S_{tr} **do**
      Create and run quantum circuit $qc(s, \theta_{s,i})$
      Compute prediction $\hat{y} \leftarrow qc(s, \theta_{s,i}))$
      Compute $L(\hat{y}, y)$
      $loss$ += $L(\hat{y}, y)$
    end for
    $loss$ = $loss/len(S_{tr})$
    Perform optimiser step $O(loss, \theta_i)$
  end for

The backend used for simulating the quantum circuits in pre-Alpha 1 model is myQLM.

4.2.2. Pre-Alpha 2

The release of the lambeq library (Kartsaklis et al., 2021) allowed the creation of the so called pre-Alpha 2 model, which uses this library to implement the pipeline of pre-Alpha 1.

The use of lambeq software comes with a series of advantages and customisation options for the model. One of the most notable advantages comes when building the PQCs, as it allows using different Ansätze for the quantum circuits seamlessly. Furthermore, the number of rotation gates associated to a single qubit and the number of variational layers in the circuit can be varied. One example of a PQC generated by pre-Alpha 2 can be seen in fig. 4.

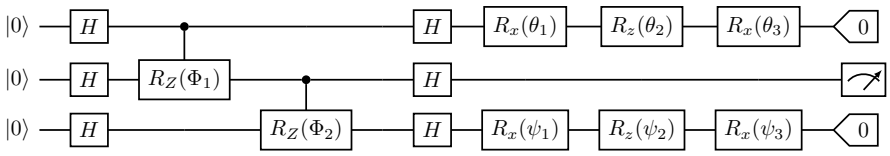

***Figure 4**: A choice of pre-Alpha 2 variational quantum circuit for the sentence "John likes Mary"*

Some differences can be seen when comparing this circuit to the one generated by pre-Alpha 1 in fig. 2. The number of rotation gates associated to a single qubit gate was chosen to be 3 instead of 2, and the chosen Ansatz was the *instantaneous quantum polynomial* (IQP) Ansatz (Shepherd & Bremner, 2008), which introduces more Hadamard gates and two new variational parameters Φ_1 and Φ_2. It stands out from fig. 4 that the number of qubits used has been reduced from 5 to 3 when compared to fig. 2. This is because lambeq can perform rewriting operations via the application of functorial transformations on the sentence diagrams. This allows for the simplification of the PQC and the reduction of the number of qubits.

Pre-Alpha 2 allows a choice of Ansätze whereas this was fixed for pre-Alpha 1. The backend used to simulate PQCs is also different as pre-Alpha 2 uses pennylane. Finally, this model offers the possibility of running on a *graphics processing unit* (GPU) to accelerate the optimisation stage, a feature which wasn't supported in pre-Alpha 1. The lambeq library also provides GPU support for the parsing of sentences into DisCoCat diagrams, a feature which has not been tested in our experiments but that we hope to look at in future.

The training algorithm for the pre-Alpha 2 model is detailed in alg. 2. Let S be the sentences in the dataset (with S_{tr} being the training dataset), Y the sentence labels, D the sentence pregroup diagrams, r the random seed for the initial variational parameters, O a classical gradient descent optimiser, l_r the learning rate of the optimiser, $epochs$ the number of epochs for optimisation, A the selected PQC Ansatz, q_n the number of noun qubits, q_s the number of sentence qubits, n_l the number of layers of our circuit QC and b the batch size. Moreover, some lambeq objects will be used: `lambeq.Dataset\equiv Dat`, `lambeq.Model\equiv M`, `lambeq.Trainer\equiv T`.

Algorithm 2 Pre-Alpha 2

Input: $(S_{tr}, Y, D, r, O, epochs, A, q_n, n_l, b)$
  Set $q_s = 1, l_r = 0.001$
  for s in S **do**
    Create $D(s) \leftarrow qc(D(s), \theta_{s,0}, A, q_n, q_s, n_l)$
    Append to QC
  end for
  Create $Dat(QC, Y, b)$
  Create $M(QC)$
  Create $T(M, loss, O, epochs, r, l_r, dev)$
  Fit $T(M, loss, O, epochs, r, l_r, dev)$ with $Dat(QC, Y, b)$

Once the model is trained, predictions can be made, as it can be seen in fig. 5.

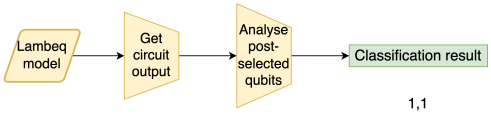

***Figure 5**: Pre-Alpha 2 pipeline*
The numbers shown under the rectangles represent the vector dimension

4.2.3. Alpha 1

Alpha serves as an extension of the pre-Alpha models, aiming to eliminate the constraint of relying solely on previously encountered words within the training dataset. To achieve this, we use *word and sentence embeddings*, which we generate using a BERT encoder (Devlin et al., 2019).

With the Alpha 1 model, we input words into this BERT encoder and extract the [CLS] token from its output. This token will then be used as an embedding of our input word. The [CLS] token is widely acknowledged as the aggregate sequence representation of its input (Devlin et al., 2019).

The underlying concept involves mapping these embeddings onto corresponding quantum parameters within PQCs by passing the embeddings through a computational kernel. Given that the number of quantum parameters linked to each word in a sentence is relatively small (1-6), while our embeddings are considerably larger (768 dimensions), this kernel must perform dimensionality reduction. Ideally, this process should be trainable based on the loss generated from the quantum circuits' output, enabling us to optimize the process.

With this type of model, we can input any test sentence and/or word that can be encoded into BERT embeddings into our PQC. Thus we have opted to construct three distinct Alpha models, all following the same *dressed quantum circuit* architecture (Mari et al., 2020).

Here we describe a specific model that combines the lambeq software package (Kartsaklis et al., 2021), the Pennylane QC framework (Bergholm et al., 2022) and the PyTorch machine learning framework (Paszke et al., 2019).

The Alpha 1 model combines the architecture of a classical network with the architecture of the pre-Alpha 2 model, as shown in fig. 6. We use the lambeq library to generate PQCs based on the input sentence, and use BERT word embeddings as one of the inputs of the model. Before using them as parameters in the quantum circuit, we reduce their dimensionality through *Principal Component Analysis* (PCA) (Maćkiewicz & Ratajczak, 1993) and linear layers. Then, we sum up the word embeddings of each sentence and feed them into a quantum circuit.

Algorithm 3 Alpha 1

Input: $(S_{tr}, Y, \hat{Y}, E, PCA, M1, F, M2, L, O, n)$
  for epoch in epochs **do**
    for s in S_{tr} **do**
      for $word$ in s **do**
        $e_1 \leftarrow E(word)$
        $e_2 \leftarrow PCA(e_1)$
        $e_3 \leftarrow M1(e_2)$
        Store e_3 in e_l
      end for
      $e_4 \leftarrow \sum_l e_l$
      $s_c \leftarrow F(s)$
      $w \leftarrow$ the 1st n values of e_2
      Update s_c parameters with w
      Run s_c
      Store output of s_c in e_5
      $\hat{y} \leftarrow M2(e_5)$
      Store $\hat{y}$ in $\hat{Y}$
    end for
    $Loss \leftarrow L(Y, \hat{Y})$
    Perform an optimizer step $O(S_{tr}, Loss)$
  end for

The algorithm for the Alpha 1 model is detailed in alg. 3. Let S be the sentences in our dataset (with S_{tr} being the training dataset), Y the labels, $\hat{Y}$ the predictions, $E(\cdot)$ the BERT encoder, $PCA(\cdot)$ the PCA transformation, $M1(\cdot)$ the function associated to the 1st multilayer perceptrons (MLP layer) (Murtagh, 1991), $F(\cdot)$ the combined lambeq functions, $M2(\cdot)$ the function associated to the 2nd MLP layer, $L(\cdot,\cdot)$ the BCE loss, $O(\cdot,\cdot)$ the Adam optimizer and n the number of trainable parameters for each sentence circuit.

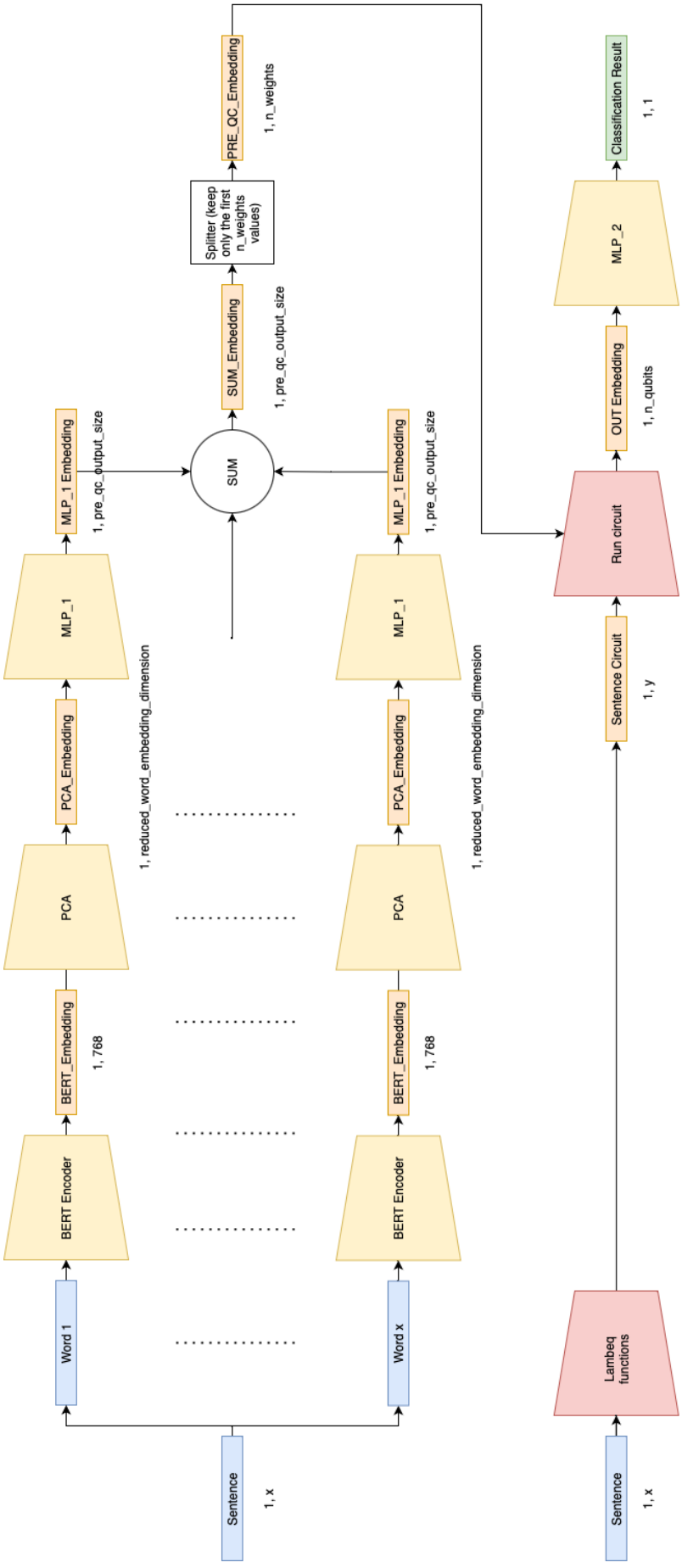

***Figure 6**: Alpha 1 pipeline*

4.2.4. Alpha 2

The Alpha 2 model combines the architecture of a classical network with the architecture of the pre-Alpha 2 model, as shown in fig. 7. This model is very similar to Alpha 1 since only their input and input pre-processing steps are different. However, while Alpha 1 is using words as its input, Alpha 2 is using sentences. These sentences are used to generate PQCs and sentence BERT embeddings. After reducing the embeddings' dimensionality through linear layers, we populate the parameters of the generated quantum circuits with them.

The training algorithm for the Alpha 2 model is detailed in alg. 4. Let S be the sentences in our dataset (with S_{tr} being the training dataset), Y the labels, $\hat{Y}$ the predictions, $E(\cdot)$ the BERT encoder, $M1(\cdot)$ the function associated to the 1st MLP layer, $F(\cdot)$ the combined lambeq functions, $M2(\cdot)$ the function associated to the 2nd MLP layer, $L(\cdot,\cdot)$ the BCE loss, $O(\cdot,\cdot)$ the Adam optimizer and n the number of trainable parameters for each sentence circuit.

Algorithm 4 Alpha 2

Input: $(S_{tr}, Y, \hat{Y}, E, M1, F, M2, L, O, n)$
  for epoch in epochs **do**
    for s in S_{tr} **do**
      $e_1 \leftarrow E(s)$
      $e_2 \leftarrow M1(e_1)$
      $s_c \leftarrow F(s)$
      $w \leftarrow$ the 1st n values of e_2
      Update s_c parameters with w
      Run s_c
      Store output of s_c in e_3
      $\hat{y} \leftarrow M2(e_3)$
      Store $\hat{y}$ in $\hat{Y}$
    end for
    $Loss \leftarrow L(Y, \hat{Y})$
    Perform an optimizer step $O(S_{tr}, Loss)$
  end for

4.2.5. Alpha 3

The Alpha 3 model combines the architecture of a classical neural network with a PQC, as shown in fig. 8. Its training algorithm is detailed in alg. 5.

Let $\mathcal{S}|card(\mathcal{S}) = k$ be the set of k training sentences from the dataset, let $Y \in \mathbb{N}^k$ be vector of labels and $\hat{Y} \in \mathbb{N}^k$ the vector of predictions such that $\hat{y}(s) \in \mathbb{N}$ is the predicted label for s. Let $\mathbf{E}$ denote the encoder -here BERT- such that $\mathbf{E}(s)$ is the BERT-based encoding of sentence s. Let $\mathbf{F}_1$ and $\mathbf{F}_2$ be the functions associated to the first and second MLP layers respectively. Finally, let $\mathbf{C}$ denote the PQC, ω the trainable variational parameters of the PQC, $\mathbf{L}$ the BCE loss, $N \in \mathbb{N}^+$ the desired number of epochs, d the number of qubits in the Ansatz and $\mathbf{O}$ the Adam optimizer.

The algorithm takes sentences and passes them through a series of functions, and outputs a classification result for these sentences.

The first function applied to the any sentence s is a BERT sentence embedding $\mathbf{E}$ which encodes the sentence into its BERT representation $e_0 \in \mathbb{R}^{768}$. Note that this encoding is unique and will not be trained throughout the process. Thus, for any sentence, its BERT encoding remains unchanged at all epochs.

The second step $\mathbf{F}_1$ feeds the BERT encoding of the sentence e_0 into a MLP. The weights of the MLP are initialised at random and trained. Therefore, for any given input embedding e_0, the outputs $\mathbf{F}_1(e_0)$ will be different for different epochs. Note that the algorithm we give here assume no batches are used. $\mathbf{F}_1$ thus takes a BERT embedding as input and outputs a d-dimensional vector where d corresponds to the number of qubits in the Ansatz, so that each qubit will receive exactly one rotation parameter from the embedding. In its current version, Alpha 3 uses a 3-qubit Ansatz and thus $e_1 = \mathbf{F}_1(e_0) : e_1 \in \mathbb{R}^3$.

An extra intermediary step before feeding the output of the first MLP into the PQC is to rescale the vector components so that they range between $\frac{-\pi}{2}$ and $\frac{\pi}{2}$. After which they are then passed on to the PQC.

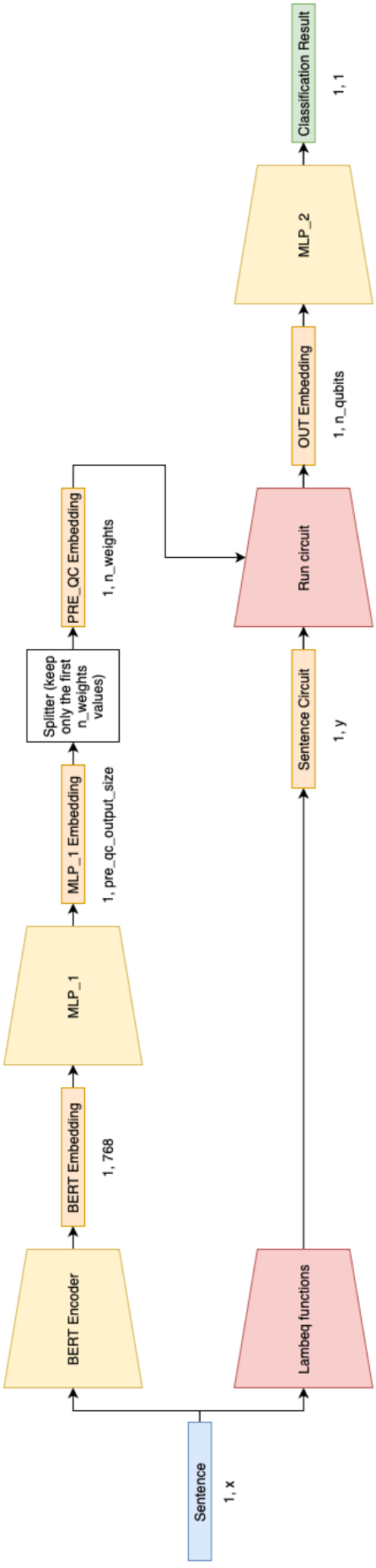

Figure 7: Alpha 2 pipeline

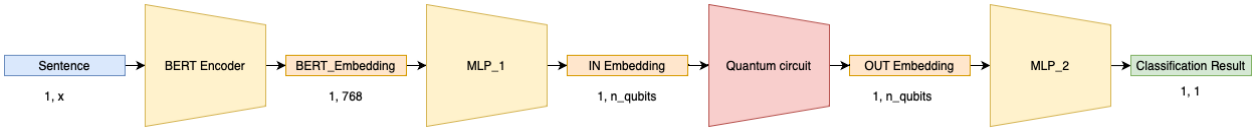

***Figure 8**: Alpha 3 pipeline*

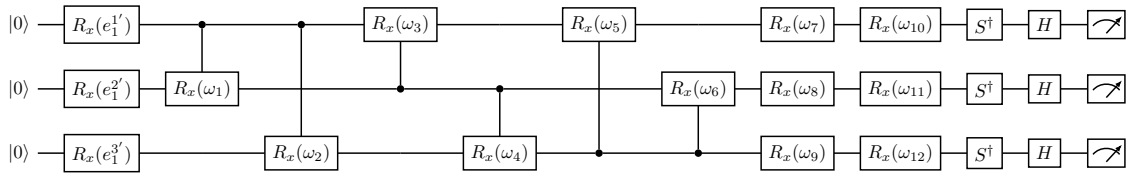

***Figure 9**: Quantum circuit favouring expressivity with $n_{qubit} = 3$ (Sim et al., 2019)*

The third step is quantum. Here the rescaled output of the first MLP e_1' is fed into the Ansatz to create a PQC noted $\mathbf{C}$. The distinctive feature of this model is that the ansatz is fixed once and for all, contrary to Alpha 2 and 1 which used sentence-specific Ansätze. The current Ansatz used is an adapted version of circuit number 6 in (Sim et al., 2019), shown in fig. 9. The version used in this work is made of 3 qubits instead of 4 as presented in Sim *et.al.*, yet the all-to-all entanglement pattern is preserved. Alpha 3 uses the embedding information contained in e_1 to parameterise the first R_x wall. Note that the other rotation parameters for subsequent simple and controlled rotations are not based on the embedding information. Those remaining parameters are initialised at random and are trained through back-propagation. The measurement of the circuit in the Y-basis then yields e_2, a d-dimensional vector containing the expected value of each qubit.

This result is then fed again through another MLP, yielding $e_3 = \mathbf{F}_2(e_2) : e_3 \in \mathbb{R}^2$.

The 2-dimensional vector resulting from the second MLP is then fed through a softmax to obtain a 2D probability vector. The predicted label will then be the index of the highest probability.

Algorithm 5 Alpha 3 - training

Input: $(\mathcal{S}, Y \in \mathbb{N}^k, N \in \mathbb{N}^+, \mathbf{E}, \mathbf{F}_1, \mathbf{F}_2, \mathbf{C}, \mathbf{L}, \mathbf{O})$
  Initialise the vector of predictions $\hat{Y} \in \mathbb{N}^k$
  Initialise the cumulative loss $l \in \mathbb{R}$
  for i ranging from 0 to N **do**
    for s in $\mathcal{S}$ **do**
      $e_0 \in \mathbb{R}^{768} \leftarrow \mathbf{E}(s)$
      $e_1 \in \mathbb{R}^d \leftarrow \mathbf{F}_1(e_0)$
      $e_1' \in \mathbb{R}^d_{[\frac{-\pi}{2}, \frac{\pi}{2}]} \leftarrow \frac{\tanh(e_1)\cdot\pi}{2}$ rescaling
      $e_2 \in \mathbb{R}^d_{[-1,1]} \leftarrow \mathbf{C}(e_1', \omega)$
      $e_3 \in \mathbb{R}^2 \leftarrow \mathbf{F}_2(e_2)$
      $\hat{y}(s) \in \mathbb{N}_{[0,1]} \leftarrow$ arg max(softmax(e_3))
      Update $\hat{Y}$ based on $\hat{y}(s)$
      Update l
      Perform an optimizer step with $\mathbf{O}$
    end for
  end for

4.2.6. Beta 1

Inspiration for this model came from the *q-means algorithm* (Kerenidis et al., 2018), a quantum adaptation of the classical *k-nearest neighbours* (KNN) algorithm (Fix & Hodges, 1989). The Beta 1 model implements the simplest possible quantum adaptation to the KNN algorithm, in which only the distance estimation step is performed using QC, with a swap test as proposed in (Lloyd et al., 2013). The decision to use QC for this step is due to the fact that the distance computation is the most computationally expensive step of the classical algorithm. The data-flow pipeline showing the algorithm is given in fig. 10.

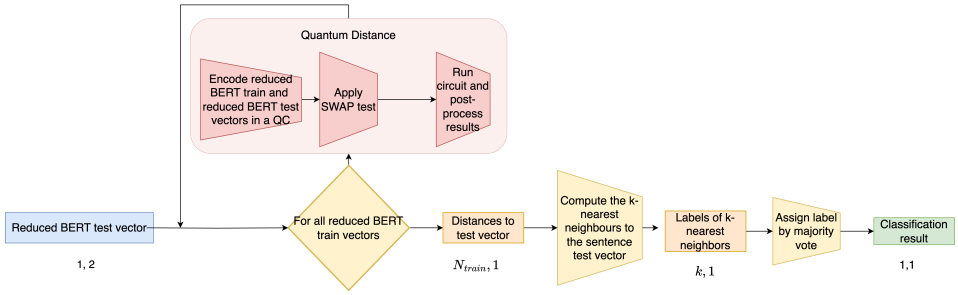

***Figure 10**: Beta 1 pipeline*

The numbers below the rectangles representing arrays define their dimension. N_{train} represents the number of sentences in the training dataset and k the number of neighbours we are taking into account

As one can see in fig. 10, the inputs of the algorithm are reduced BERT embeddings for each sentence. The input was named reduced BERT embeddings because a PCA was applied to each sentence embedding in order to reduce its dimension from 768 to 2. The distance is computed between each test vector and all the rest of the train vectors. To do so, angle encoding is used to encode the vectors in the quantum circuit, followed by a swap test and a post-processing of the results that allows for the computation of the Euclidean distance. Once the distances to all train vectors have been computed, the k closest vectors are computed and then the label of the test vector is assigned by majority vote. A detailed description of the use of the swap test for KNN implemented on superconducting qubits can be found in (Sarma et al., 2019).

The Beta 1 algorithm can be seen in alg. 6. As there is no training stage needed for this algorithm (the labels are assigned just by majority vote on the k-nearest neighbors), what is shown in alg. 6 is a description of how a label is assigned to a test sentence, instead of the training of process shown for previous models.

Let S be a sentence (with S_{tr} being the train dataset and S_{te} being our test dataset), and k the number of k neighbours. The superscript r will be used to name the reduced vector embeddings after PCA, and s_i for naming the sentence vector of which its prediction is being computed.

Algorithm 6 Beta 1

Input: S, Y, k
  $S^r_{tr} \leftarrow PCA(S_{tr})$
  $s^r_i \leftarrow PCA(s_i)$
  for s^r_j in S^r_{tr} **do**
    Build a SWAP-test circuit c
    Normalise s^r_j, s^r_i
    Encode s^r_j, s^r_i into a c
    Run c and get $dist_{i,j}$
    Append $dist_{i,j}$ to D
  end for
  Find k minimum values in D
  Assign label with majority vote

4.3. Classical NLP Models

We benchmark quantum algorithms in comparison with classical NLP models – classifiers trained using (at this time) current neural network-based methods. In classical NLP, current state-of-the-art classification methods (Gasparetto et al., 2022) rely on pre-trained (large) language models that are either fine-tuned for classification purposes or used as sentence and or word embedding (i.e., vectorisation) mechanisms on top of which a smaller neural network classifier is trained to perform the down-stream classification task. For a sentence, embedding can be performed either for each word (e.g., using skip-gram models (Mikolov et al., 2013), positional embeddings from BERT (Kenton & Toutanova, 2019), or any other pre-trained language model that can output embeddings for each word in a sentence) or sentence embedding methods. Sentence embeddings can be acquired from, e.g., BERT using either the “CLS” token’s embeddings or by

averaging the positional word embeddings or using any other pre-trained language model that outputs sentence representations (e.g., Laser (Schwenk & Douze, 2017), LaBSE (Feng et al., 2022), etc.).

In our experiments, we use both word and sentence embedding methods. Depending on the embedding unit - sentence or word - we then build different neural network classifiers on top of the embedding vectors that are treated as input to the classifiers.

For sentence-level embeddings, we performed two experiments where we compared sentence vectorization using pre-trained embedding models *bert-base-uncased* and *all-distilroberta-v1* from the Huggingface Transformers library (Wolf et al., 2020). For the classification of the vectorized sentences, we used a multi-class logistic regression model that consists of a feed-forward neural network with one densely connected feed-forward layer with a softmax activation function (see fig. 11). The model is rather simple since the sentence-level embeddings already come from a complete Transformer-based neural network and we deal with relatively small datasets for which more complex architectures may not be useful/productive.

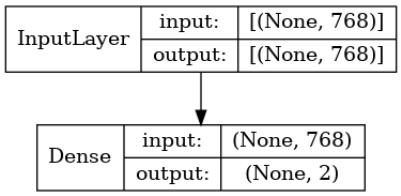

***Figure 11**: The architecture of the feed-forward neural network*

For word-level embeddings, We also performed two experiments and compared the pre-trained embedding models *bert-base-uncased* (from the Transformers library) and *cc.en.300.bin* (from the fastText library (Joulin et al., 2016)). For classification, we use a convolutional neural network. Figure 12 depicts the architecture of the network. It shows that the input (word) vectors are passed through multiple convolutional filters (*Conv1D* is a temporal convolution layer) with max pooling (*MaxPooling1D* is a layer that performs max pooling for one-dimensional temporal data). Then, outputs are concatenated (using the layer *Concatenate*) and flattened into a vector (using the layer *Flatten*). Then, to prevent over-fitting, we apply dropout with a probability of 50% (using the layer *Dropout*. Dropout randomly sets input units to 0. Finally, we use a densely connected feed-forward layer (*Dense*) with a softmax activation function.

To compare how well the pre-trained embeddings (if at all) help the short text classification task, we performed also one experiment without using pre-trained embedding vectors. Input vectors for words in this model are learned at the model training time. For classification, we use a bidirectional long short-term memory neural network (LSTM; see fig. 13 for the architecture of the neural network). Since we do not use pre-trained embeddings, we have to feed words directly into the neural network (using a one-hot vectorizer). This also means limiting the input vector (or vocabulary) to a fixed size. For that, we limit the input vocabulary to 5000 most frequent words/tokens and treat the rest as unknown (i.e., we assign them to the token '<oov>'). The embedding and bidirectional LSTM layer (*Bidirectional(LSTM)*) dimensionality is set to 128. Similarly to other models, we add a dense feed-forward layer with the softmax activation function to acquire the classification results.

All classical NLP neural network models output a probability distribution over the two possible output classes for sentiment analysis (negative or positive). In other words, the output is a two-dimensional vector where the first score indicates how likely the input data represented a negative sentiment and the second score – a positive sentiment.

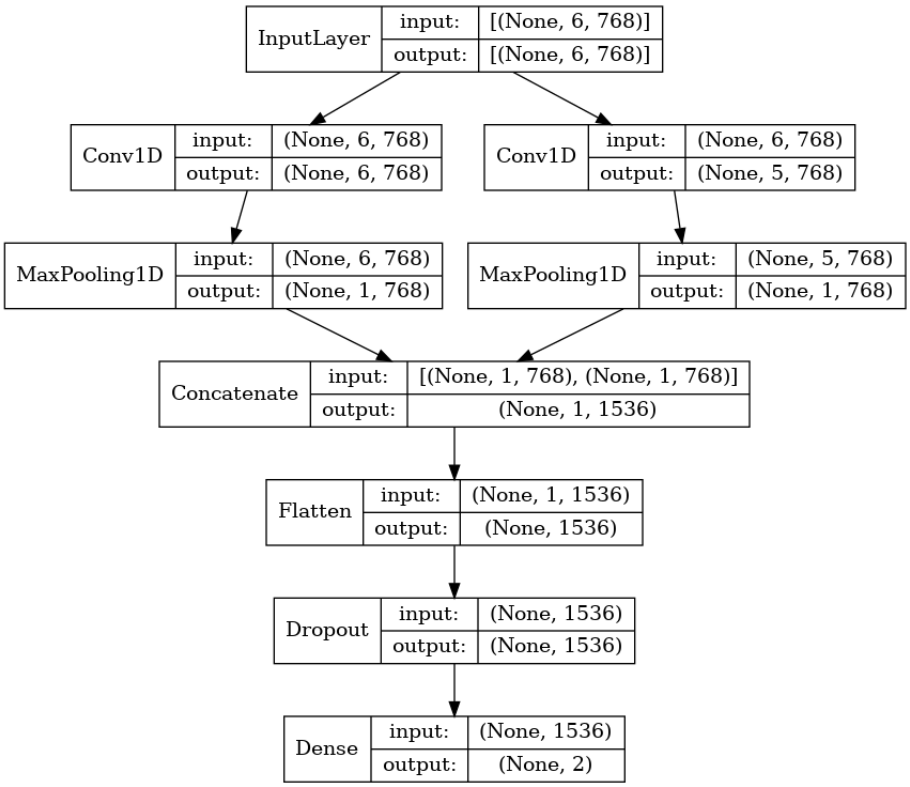

***Figure 12**: The architecture of the convolutional neural network*

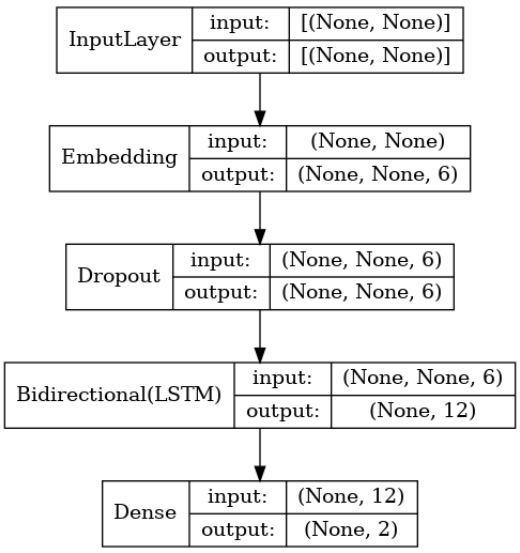

***Figure 13**: The architecture of the bidirectional LSTM neural network*

4.4. Datasets

Quantum computing models investigated in the project rely on encoding data using a circuit designed for that data. Several technical limitations arise due to this approach, which then put constraints on what data can be currently processed in QNLP. Some of the restrictions that we have identified are: sentences cannot be long (up to 6 tokens/words) and they must follow pre-selected syntactic structures for which circuits have been designed. Since natural language is typically more complex with longer sentences (of about 20 words), there aren't many datasets that satisfy these constraints. However, there are some applicable for the NLP task on short text classification (Alsmadi & Gan, 2019; Song et al., 2014) that are potential

candidates. The datasets we identified as potential candidates for our experiments were:

- Sentiment analysis datasets:
  - **Amazon Fine Food Reviews** (McAuley & Leskovec, 2013), a dataset consisting of 568,454 food reviews. It has a summary field that summarises each review in a short sentence and a sentiment score - an integer ranging from 1 to 5.
  - **Amazon Reviews** (X. Zhang et al., 2015), a dataset consisting of 3,600,000 reviews from amazon. Each review has a title and a sentiment polarity score assigned (either 1 for negative or 2 for positive).
- Topic classification datasets:
  - **Topic Labeled News Dataset** (NewsCatcher, 2020), a dataset consisting of 108,774 news articles. We are interested in two fields - the topic the article belongs to and the title of the article. There are eight topics in total - business, entertainment, health, nation, science, sports, technology, and world.
  - **AG News Classification Dataset** (X. Zhang et al., 2015), a dataset consisting of 120,000 news articles. Each article consists of a topic, title, and description. There are four topics in total - world, sports, business, and science/technology.

The datasets were normalised, filtered and pre-processed for further use as follows:

1. We normalised punctuation using the Moses (Koehn et al., 2007) punctuation normalisation script[1].
2. We tokenised sentences using the Spacy Tokenizer[2].
3. We discarded all sentences longer than 6 tokens. We have decided to select only sentences with a maximum of 6 words. Indeed, since the number of qubits needed to execute the circuit created with lambeq is correlated with the number of words in each sentence, we decided to add this maximum 6-word constraint to our dataset. This is to ensure compliance with the capabilities of NISQ-era devices and to enable the simulation of all quantum computations on our classical hardware.
4. We parsed each sentence with the BobCat parser from lambeq (Kartsaklis et al., 2021).
5. We filtered the datasets to contain only sentences of a pre-defined set of sentence structures (see sec. 4.4.1).
6. We split the datasets into training, validation (or development), and evaluation (or testing) sets (see sec. 4.4.3).

4.4.1. Sentence structures

Pregroups in DisCoCat formalism are used to represent the grammar of sentences. Pregroups are partially ordered *monoids* (meaning that they are equipped with monoid multiplication with unit 1) whose elements p have defined a *left adjoint* p^l and a *right adjoint* p^r such that the following relations hold:

$$p^l \cdot p \leq 1 \quad p \cdot p^r \leq 1$$

One can define grammatical types by freely generating pregroups that are used to capture the relationships and compositional aspects of meaning in language. The basic grammatical types one word can have are the following: (Coecke et al., 2010):

n: noun
s: declarative statement
j: infinitive of the verb
σ: glueing type

The grammar of a sentence can be represented in pregroup language by the composition of the types of each word. After the relations between adjoints are applied, if one is left with an s type, we say that the

[1] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/normalize-punctuation.perl
[2] https://spacy.io/api/tokenizer

sentence is *grammatical*. Here is a basic example taken from (Coecke et al., 2010). The sentence *John likes Mary* has the following type assignment:

$$John \quad likes \quad Mary$$

$$n \quad (n^r s n^l) \quad n$$

Using the adjoint relations of pregroups, and the fact that these are associative, one can verify that:

$$n(n^r s n^l)n \to s n^l n \to s \to s$$

where we have replaced the $\leq$ with an arrow $\to$, and removed the dot between types. This shows that the sentence is grammatical. Short texts often do not strictly follow grammar rules and syntactic parsers do not always output correct sentence structures even for grammatically correct sentences. For instance, the short text sentences (titles) in tab. 1 were not recognised as grammatically correct. However, these are normal, valid sentences in natural uncontrolled language (e.g. titles of news articles, reviews, or used in spoken language). Since the restriction to only use grammatically valid sentences discarded quite a lot of useful examples (e.g., over 34% of AG News and over 46% of NewsCatcher examples after length filtering), we used a relaxed version of these grammatical structures when selecting our sentences. Given the constraint of natural language datasets, we chose to allow for sentences which are treated as valid in most NLP settings though not grammatically correct from a pure syntactic perspective. Examples of such include *Ultraviolet = Terrible* which has the structure $(n \quad n^r n)$ and can not be reduced to the type s.

Example sentence	Grammatical structure
Essential Keto tips for beginners	n[n[n[(n/n) n[(n/n) n]]] (n\\n)[((n\\n)/n) n[n]]]
Stages of Lung Cancer	n[n[n] (n\\n)[((n\\n)/n) n[n[(n/n) n]]]]
Letters	n
Americans and Freedom	n[n (n\\n)[conj n]]
Rehabbing his career	(s\\n)[((s\\n)/n) n[(n/n) n]]

***Table 1**: Examples of sentences not recognised as grammatically valid*

The sentences in our datasets follow one of the 5 grammatical structures depicted in the tab. 2.

	grammatical structure
Structure 1	$n \quad (n^r s n^l) \quad n$
Structure 2	$n \quad (n^r n) \quad (n^r s n^l) \quad n$
Structure 3	$n \quad (n^r n) \quad (n^r s)$
Structure 4	$n \quad (n^r s n^l) \quad (n^r n n^l) \quad (n^r n)$
Structure 5	$n \quad (n^r s n^l) \quad (n^r n n^l) \quad (n^r n n^l) \quad (n^r n)$

***Table 2**: Description of the different grammatical structures available in our datasets*

Below are some examples for each sentence structure we use:

- **Structure 1** :
  - *Simple is good.*
  - *Ultraviolet = Terrible.*
  - *Reagan was right.*
- **Structure 2** :
  - *My son was thrilled.*
  - *The Myth Is Better.*
  - *This album is good.*
- **Structure 3** :

  - *Drill bits suspect.*
  - *mariah voice perfected.*
  - *SECOND EXTRAORDINARY READ.*[3]

- **Structure 4** :
  - *I love these books.*
  - *i like this cd.*
  - *they sound the same.*
- **Structure 5** :
  - *This is an absolute steal.*
  - *this is a bad product.*
  - *They ship the wrong cable.*

Due to the constraints of how entanglement is embedded into the circuit for pre-Alpha models (namely through hard-coded entanglement structure of the PQC), we limited the number of sentence structures we could support to 5. In order to choose which 5 structures to work with, the length-filtered dataset candidates were analysed. Each sentence structure with less than 6 words was listed and a frequency count was performed. With this information in hand, the top 5 sentence structures with highest frequency were chosen. To allow for proper training over all supported sentence structures, we chose the dataset which had the highest number of sentences of the least frequent top-5 structure (see tab. 3).

Data set	Raw	Filtered	Sent. struct.	Filtered to 5 sent. struct.
Amazon Fine Food Reviews	568,455	36,803	4,402	1,995
Amazon Reviews	3,600,000	1,679,957	43,335	21,067
Topic Labeled News Dataset	108,774	3,390	1,074	12
AG News Classification Dataset	120,000	40,065	4,291	396

***Table 3**: Dataset statistics before and after filtering*

4.4.2. Datasets composition

Due to technical and time constraints, we created a sub-set of the Amazon Reviews data set. We name this subset as dataset B, though it isn't a new dataset but simply a subset of the main dataset (A) (reduced_amazonreview dataset) and the original (normalised, filtered, and pre-processed) dataset as dataset A (amazonreview dataset).

	$\#(D)$	structure 1	structure 2	structure 3	structure 4	structure 5
dataset A	21067	$6614 - 31.40\%$	$4176 - 19.82\%$	$3665 - 17.40\%$	$3595 - 17.06\%$	$3017 - 14.32\%$
dataset B	4714	$1263 - 26.79\%$	$1113 - 23.61\%$	$619 - 13.13\%$	$937 - 19.88\%$	$782 - 16.59\%$

***Table 4**: Number and percentage (over the whole dataset) of sentence structure per dataset*

	$\#(D)$	sentences labelled 0	sentences labelled 1
dataset A	21067	$10425 - 49.48\%$	$10642 - 50.52\%$
dataset B	4714	$2348 - 49.81\%$	$2366 - 50.19\%$

***Table 5**: Number and percentage (over the whole dataset) of sentence label per dataset*

[3]The lambeq BobCat parser is case-insensitive by default. We used this default setting on all our experiments.

4.4.3. Dataset splits

Each dataset was split into train, validation and tests sets. Any single sentence could only be part of a single of those subsets (no overlap).

	dataset A	dataset B
train, whole dataset	$18961 - 90.00\%$	$3300 - 70.00\%$
validation, whole dataset	$1053 - 5.00\%$	$707 - 15.00\%$
test, whole dataset	$1053 - 5.00\%$	$707 - 15.00\%$
train, structure 1	$6082 - 91.96\%$	$914 - 72.37\%$
validation, structure 1	$291 - 4.40\%$	$169 - 13.38\%$
test, structure 1	$241 - 3.64\%$	$180 - 14.25\%$
train, structure 2	$3628 - 86.88\%$	$745 - 66.94\%$
validation, structure 2	$246 - 5.89\%$	$192 - 17.25\%$
test, structure 2	$302 - 7.23\%$	$176 - 15.81\%$
train, structure 3	$3542 - 96.64\%$	$466 - 75.28\%$
validation, structure 3	$56 - 1.53\%$	$80 - 12.93\%$
test, structure 3	$67 - 1.83\%$	$73 - 11.79\%$
train, structure 4	$3096 - 86.12\%$	$655 - 69.90\%$
validation, structure 4	$273 - 7.59\%$	$132 - 14.09\%$
test, structure 4	$226 - 6.29\%$	$150 - 16.01\%$
train, structure 5	$2613 - 86.61\%$	$520 - 66.50\%$
validation, structure 5	$187 - 6.20\%$	$134 - 17.13\%$
test, structure 5	$217 - 7.19\%$	$128 - 16.37\%$

***Table 6**: Number and percentage of sentences in train, validation and test sets per structure and per dataset*

	dataset A	dataset B
0-label, whole dataset	$10425 - 49.48\%$	$2348 - 49.81\%$
1-label, whole dataset	$10642 - 50.52\%$	$2366 - 50.19\%$
0-label, train	$9419 - 49.68\%$	$1657 - 50.21\%$
1-label, train	$9542 - 50.32\%$	$1643 - 49.79\%$
0-label, validation	$462 - 43.87\%$	$342 - 48.37\%$
1-label, validation	$591 - 56.13\%$	$365 - 51.63\%$
0-label, test	$544 - 51.66\%$	$349 - 49.36\%$
1-label, test	$509 - 48.34\%$	$358 - 50.64\%$

***Table 7**: Number and percentage of sentences per label in train, validation and test sets (per dataset)*

4.5. Technical specifications

4.5.1. Hardware

All of our experiments, with the exception of those involving the Pre Alpha 1 model, were run on Ireland's national national *high-performance computing* platform, Kay, which is managed by the Irish Centre for High-End Computing (ICHEC). The experiments using only *central processing unit* (CPU) were ran on a single node within Kay's main cluster component, which consists of 336 nodes, each comprising 2x 20-core 2.4GHz Intel Xeon Gold 6148 (Skylake) processors, with 192 GiB of random-access memory (RAM), a 400 GiB local SSD for scratch space and a 100Gbit OmniPath network adaptor. The experiment that made use of a *graphics processing unit* (GPU) was ran on a single GPU within a node in Kay's GPU component, which is composed of 16 nodes with the same specifications as those in the cluster component, each equipped with two NVIDIA Tesla V100 16GB PCIe (Volta architecture) GPUs. Software dependencies were packaged in Conda environments for each model. The libraries used and their versions are specified in our repository in the `pyproject.toml` file.

Because of technical limitations, pre-Alpha 1 experiments were ran on laptops, and the results were then combined into a single file. This model was developed for a previous deliverable with an old version of the myQLM library, which was producing a number of errors when trying to use it within a Conda environment on Kay. Due to this and the fact that our machines have different technical specifications, we do not provide timing information for this model, as no meaningful time comparisons can be made with the other experiments. We may look to update the model to make it compatible with Kay in the future, but this is currently not a priority.

5. Results

A total of four experiments were conducted which are detailed in this chapter. Their contents are depicted below, tab. 8.

	dataset	#runs	#epochs	CPU/GPU	models
Experiment 1	B	30	150	CPU	pre-Alpha & Alpha models
Experiment 2	A	10	150	CPU	pre-Alpha 2 & Alpha models
Experiment 3	A	30	150	GPU	Alpha 3 model
Experiment 4	A & B	1	1	CPU	Beta model

***Table 8**: Description of the different experiments*

5.1. Forenotes

5.1.1. Metrics

The core two metrics used to evaluate model performance were the loss and accuracy for the binary classification task.

5.1.1.1. Loss

For each experiment, we compute the BCE loss between the true labels y and the prediction $\hat{y}$ of our models.

y is a binary variable $\in \{0, 1\}$ which follows a Bernoulli distribution: $P(Y = y \mid x, \theta) = \hat{y}^y(1-\hat{y})^{1-y}$, with θ the trainable parameters of our model.

Thus we want to find the parameters θ that maximise the likelihood of y given the input x:

$$\max_{\theta} P(Y = y \mid x, \theta) = \hat{y}^y(1-\hat{y})^{1-y} \tag{5.1}$$

Which corresponds to finding the parameters θ that maximise the log-likelihood of y given the input x:

$$\max_{\theta} log(P(Y = y \mid x, \theta)) = log(\hat{y}^y(1-\hat{y})^{1-y}) = ylog(\hat{y}) + (1-y)log(1-\hat{y}) \tag{5.2}$$

Since our goal is to have a function to minimize and not maximize as a loss, we take the contrary of the log-likelihood. Which gives us the BCE loss:

$$\mathcal{L}(y, \hat{y}) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y})) \tag{5.3}$$

This process can be applied to an entire dataset of size N, thus giving us the cost function:

$$\mathcal{J}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i) \tag{5.4}$$

The lower the loss, the better the model performs.

Note: For a binary classification task, if the training dataset is perfectly balanced, during the first epoch $\hat{y} \approx 0.5$ for every sample. Thus $\mathcal{J}(\theta) \approx \frac{1}{N}(\sum_{i=1}^{N/2} -log(1-0.5) + \sum_{i=1}^{N/2} -log(0.5)) \approx -log(\frac{1}{2}) = 0.6931$.

5.1.1.2. Accuracy

Accuracy is the most commonly used metric for classification tasks (Blagec et al., 2021). It quantifies the proportion of correctly predicted labels out of the total labels in a dataset.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{5.5}$$

	positive prediction	negative prediction
positive label	true positive (TP)	false negative (FN)
negative label	false positive (FP)	true negative (TN)

With a perfectly balanced dataset, the higher the accuracy is, the better the model performs.

5.1.2. Heuristics

After conducting preliminary experiments, we observed that with the current hyperparameters used for each model, our losses reached a plateau after approximately 100 epochs. Therefore, we decided to run every experiment for 150 epochs to make sure we weren't missing any possible improvements. We also monitored performance with loss and accuracy to ensure the models were not over-fitting and didn't need any early stopping.

As mentioned briefly in the hardware specifications in sec. 4.5.1, the runtime for pre-Alpha 1 is not considered representative or comparable with other models and was thus discarded.

For Alpha models and the Beta model, the BERT embeddings are computed beforehand to simplify and accelerate the workflow. Thus Alpha and Beta runtimes do not include the generation of the BERT embeddings, as is common practice in classical NLP.

Note that Alpha 3 uses multiple CPU cores during the training while the other models are only using 1 CPU core. This should be taken into account when comparing performances.

5.2. Experiment 1

Experiment 1 is meant to serve as the common baseline for every model which involves learning (thus excepting beta). It takes into account the constraints of every model, which are either constraints related to processing units or constraints related to runtime. Therefore this experiment is made of 30 runs on a CPU node for each model using the smallest dataset: B.

model	experiment 1						
	runtime (min)	train loss	valid loss	test loss	train acc	valid acc	test acc
pre-Alpha 1	n/a, see 4.5.1	1.04 ± 0.06	1.14 ± 0.08	0.69 ± 0.00	0.54 ± 0.02	0.50 ± 0.02	0.54 ± 0.03
pre-Alpha 2	275.39 ± 4.29	0.24 ± 0.02	0.85 ± 0.05	0.34 ± 0.01	0.90 ± 0.01	0.59 ± 0.03	0.60 ± 0.03
Alpha 1	240.51 ± 2.73	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.50 ± 0.01	0.51 ± 0.02	0.50 ± 0.01
Alpha 2	236.62 ± 2.58	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.50 ± 0.01	0.51 ± 0.02	0.50 ± 0.01
Alpha 3	2.68 ± 0.05	0.42 ± 0.08	0.43 ± 0.07	0.42 ± 0.08	0.81 ± 0.08	0.8 ± 0.08	0.81 ± 0.08

***Table 9**: Experiment 1 results*
dataset B, 30 runs, CPU, all pre-Alpha and Alpha models

In this experiment Alpha 1 and Alpha 2 models maintain almost the same loss and accuracy during all the training process. Both display an almost purely random binary classification performance, with the training-test accuracies hovering around 0.5 and the training-validation-test losses hovering around 0.7. Moreover, note that both models have extremely similar performances as can be seen in figs. 19 & 20.

The pre-Alpha 1 model also maintains almost the same loss and accuracy during all the training process. The difference with Alpha 1 and 2 is that in this case the performance is slightly better.

While the pre-Alpha 2 model reaches a train accuracy of ≈ 0.9 and a test and validation accuracies of ≈ 0.6. These results may be due to overfitting. The pre-Alpha 2 model has likely learned the training data too well, leading to a lack of generalization and a decrease in performance on unseen data. Using regularization techniques or stopping the training of this model before the 150 epochs threshold could improve the test and validation results.

Finally the Alpha 3 model reaches a test accuracy of ≈ 0.82 with a very low variance.

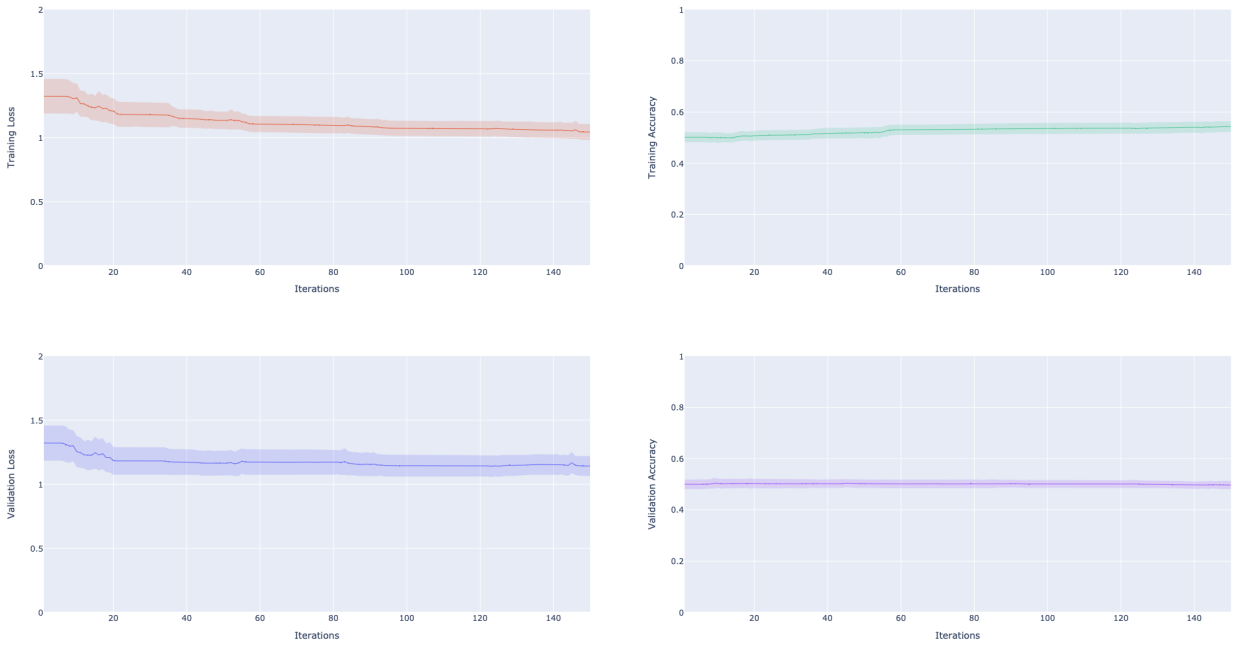

***Figure 14**: Pre-Alpha 1 - experiment 1*
dataset B, 30 runs, CPU

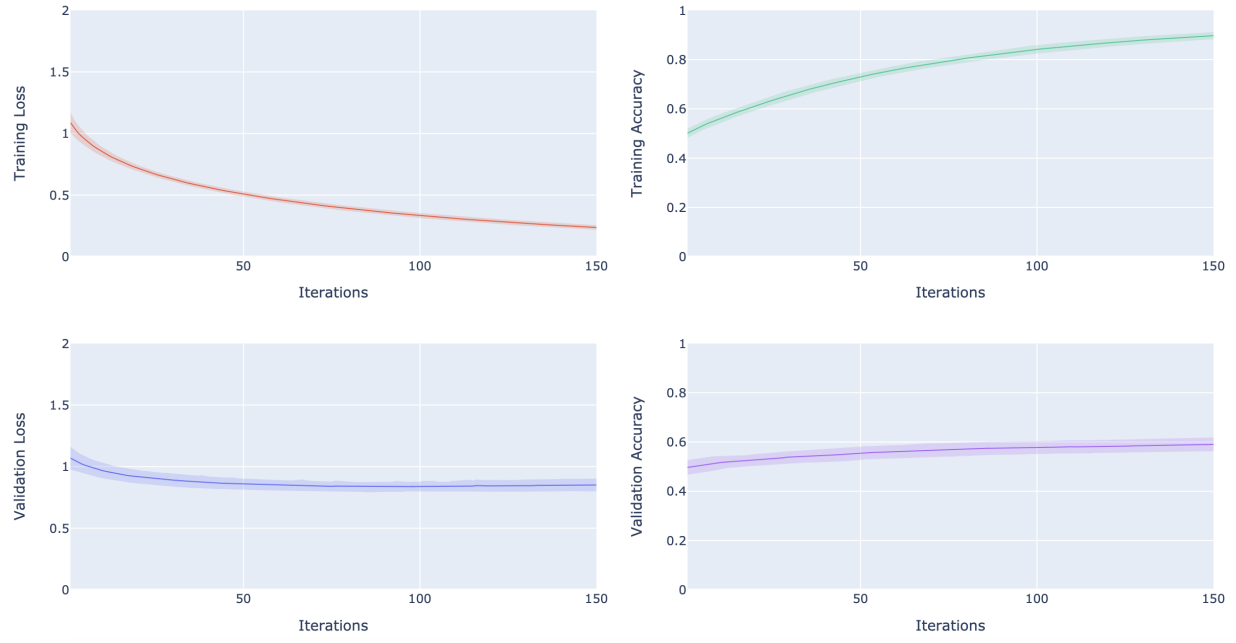

***Figure 15**: Pre-Alpha 2 - experiment 1*
dataset B, 30 runs, CPU

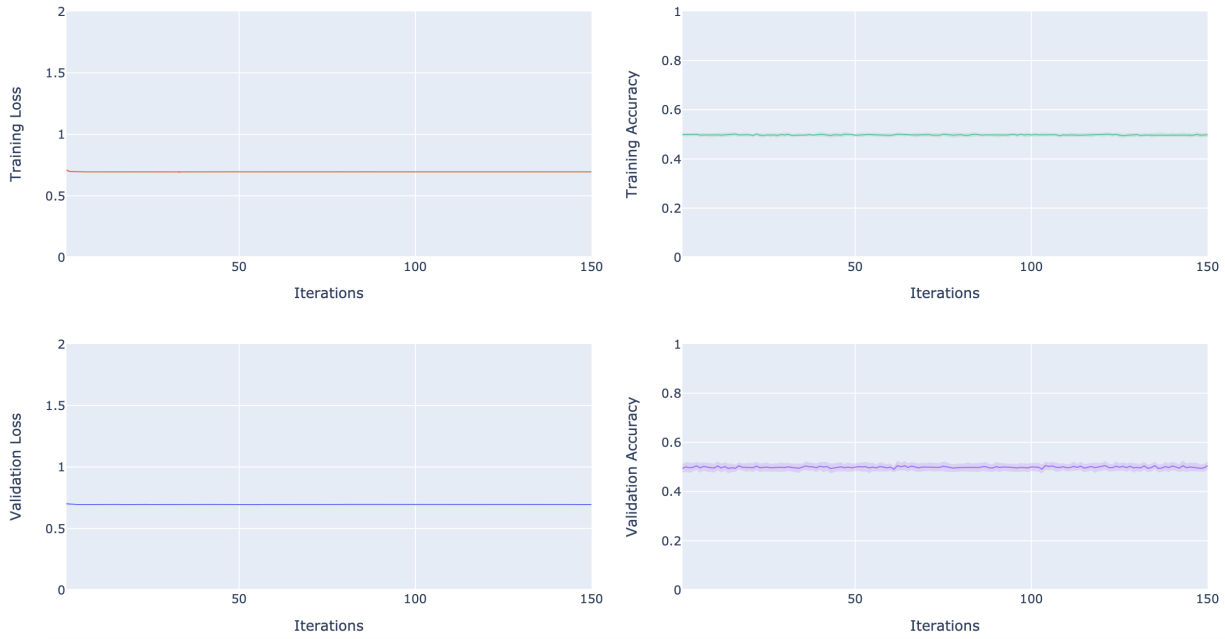

***Figure 16**: Alpha 1 - experiment 1*
dataset B, 30 runs, CPU

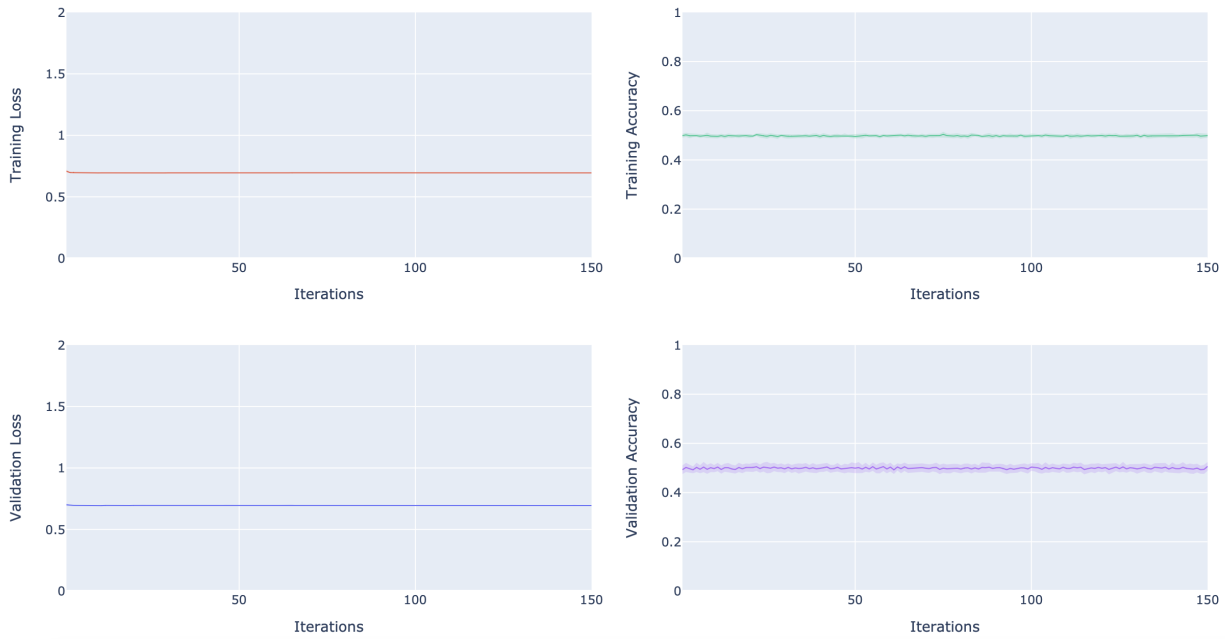

***Figure 17**: Alpha 2 - experiment 1*
dataset B, 30 runs, CPU

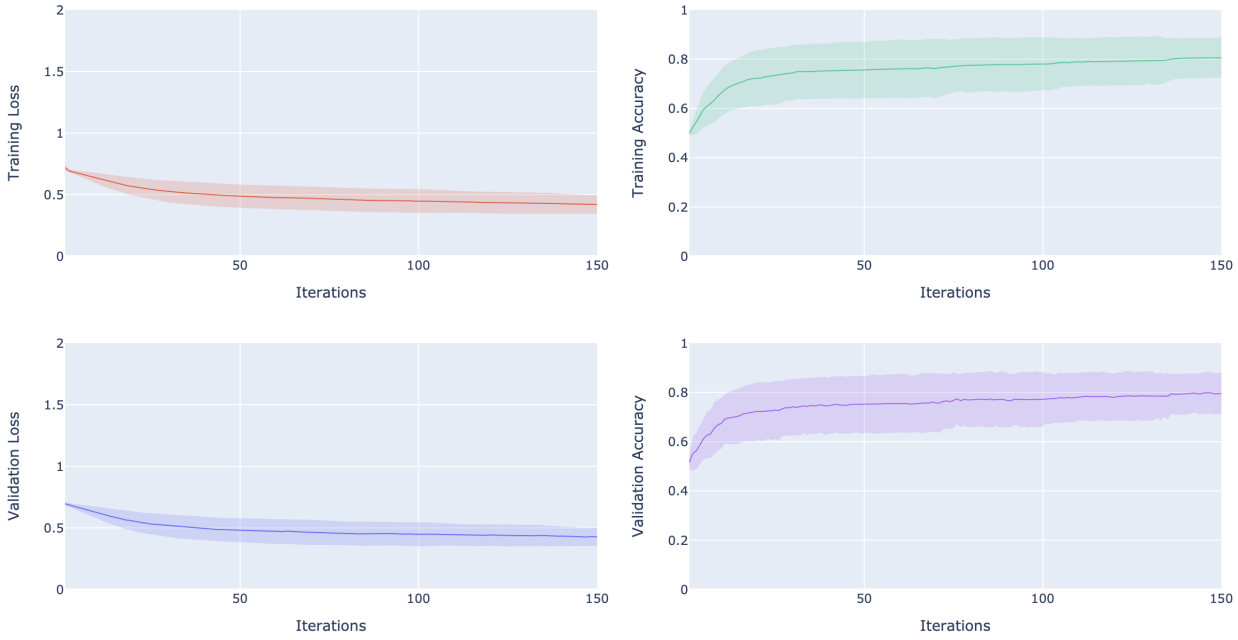

***Figure 18**: Alpha 3 - experiment 1*
dataset B, 30 runs, CPU

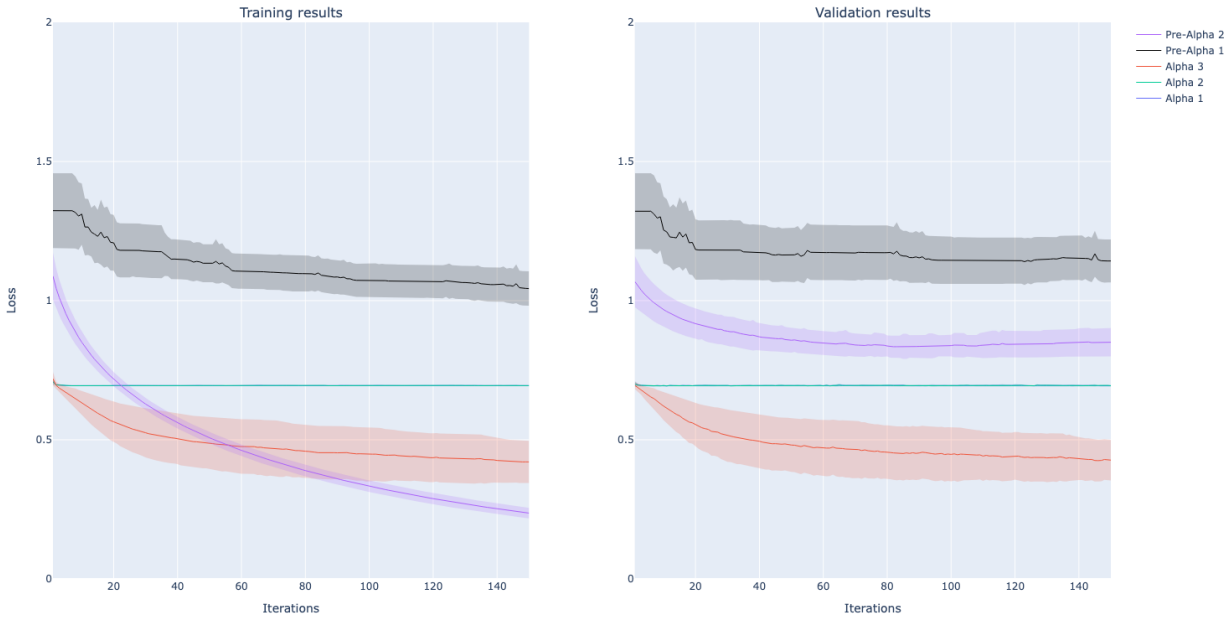

***Figure 19**: All model losses - experiment 1* [1]
dataset B, 30 runs, CPU

Note: Alpha 1 & 2 plots are superposed

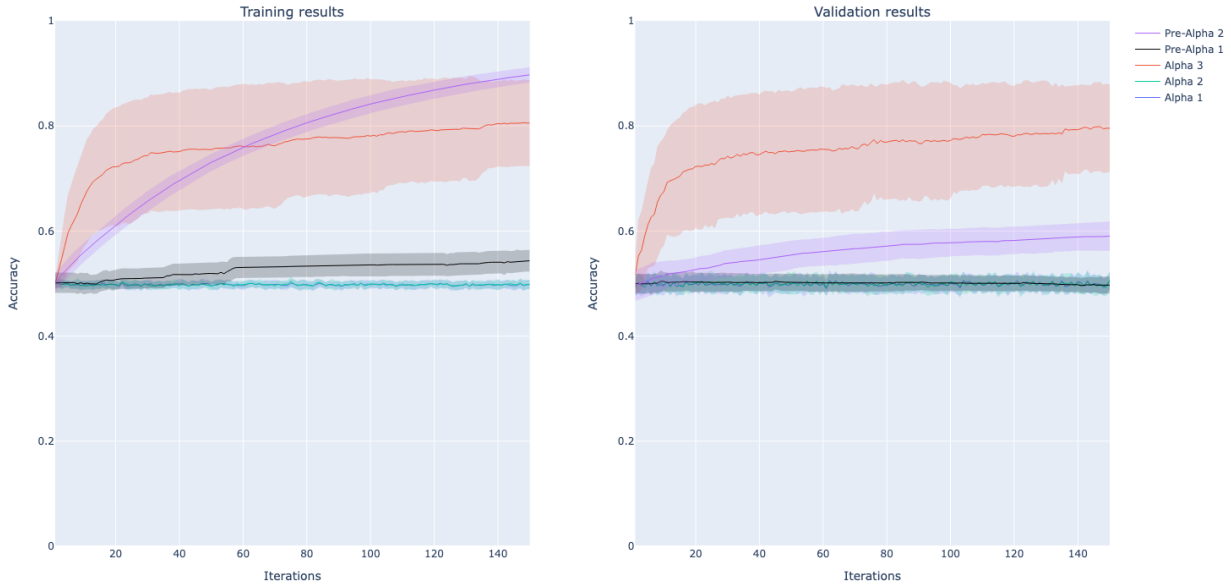

***Figure 20**: All model accuracies - experiment 1*
dataset B, 30 runs, CPU

5.3. Experiment 2

The goal of experiment 2 is to determine whether our models can generalize effectively on a larger dataset. Again it takes into account the constraints of every model, which are either constraints related to processing units or constraints related to runtime. Therefore this experiment is made of 10 runs on a CPU for each model, using the larger dataset A.

model	experiment 2						
	runtime (min)	train loss	valid loss	test loss	train acc	valid acc	test acc
pre-Alpha 2	1357.9 ± 24.68	0.27 ± 0.01	0.68 ± 0.05	0.34 ± 0.01	0.88 ± 0.01	0.69 ± 0.02	0.68 ± 0.03
Alpha 1	1219.4 ± 12.83	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.50 ± 0.00	0.53 ± 0.04	0.49 ± 0.02
Alpha 2	1206.33 ± 11.41	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	0.50 ± 0.00	0.53 ± 0.04	0.49 ± 0.02
Alpha 3	11.17 ± 0.26	0.45 ± 0.08	0.41 ± 0.09	0.37 ± 0.11	0.78 ± 0.09	0.81 ± 0.08	0.83 ± 0.11

***Table 10**: Experiment 2 results*
dataset A, 10 runs, CPU, pre-Alpha 2 and Alpha models

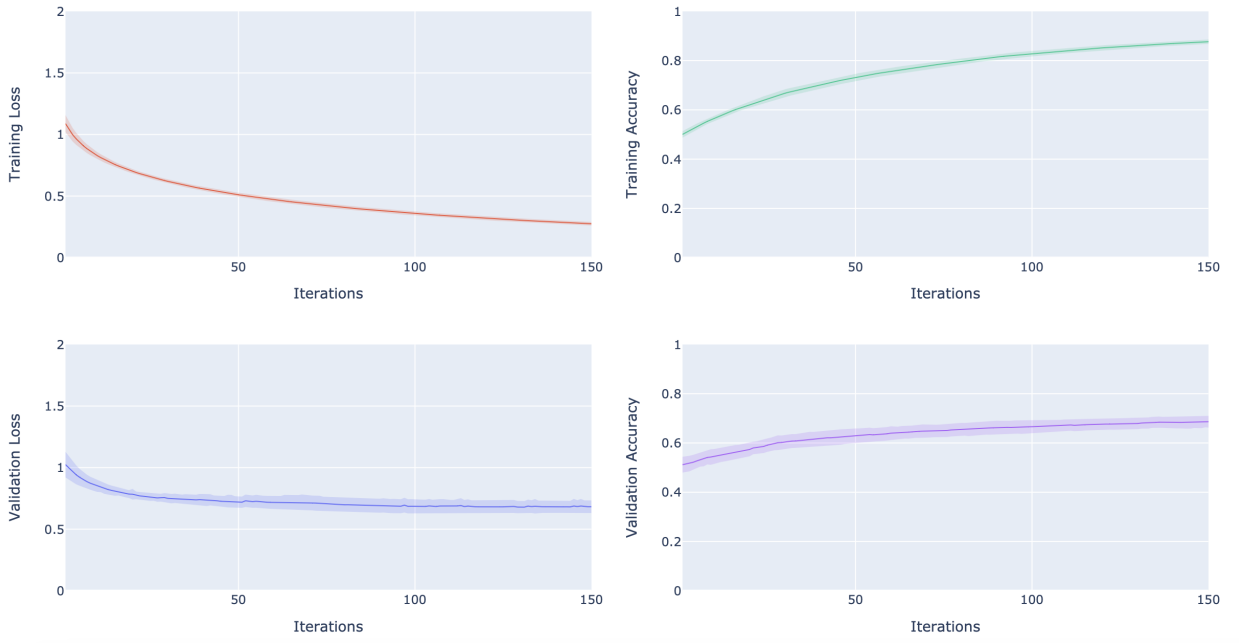

***Figure 21**: Pre-Alpha 2 train/validation loss and accuracy*
dataset A, 10 runs, CPU

Similarly to experiment 1, Alpha 1 and Alpha 2 maintain almost the same loss and accuracy throughout the training, displaying a random binary classification behaviour with the training-test accuracies hovering around 0.5 and the training-test losses hovering around 0.7. Both models have extremely similar performances as can be seen in fig. 25 & 26.

The behaviour of pre-Alpha 2 is similar to the one the same model displays in experiment 1 with low test and validation accuracies around 0.7 while train accuracy was at 0.88.

Finally the Alpha 3 model reaches a test accuracy of ≈ 0.87 with a low variance.

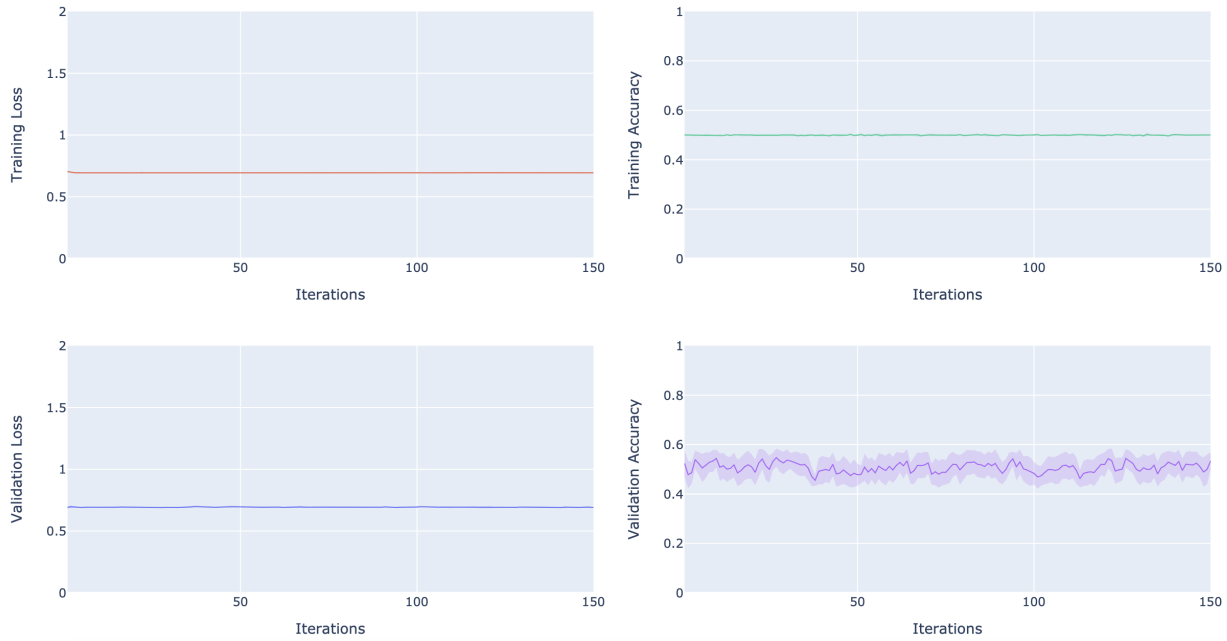

***Figure 22**: Alpha 1 - experiment 2*
dataset A, 10 runs, CPU

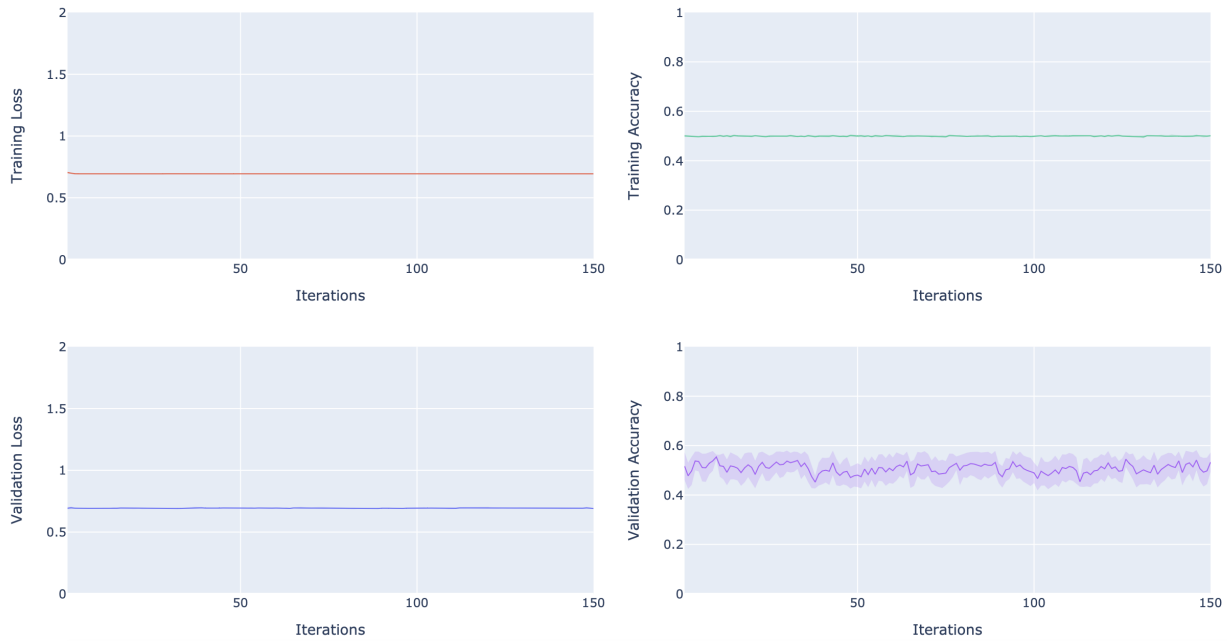

***Figure 23**: Alpha 2 - experiment 2*
dataset A, 10 runs, CPU

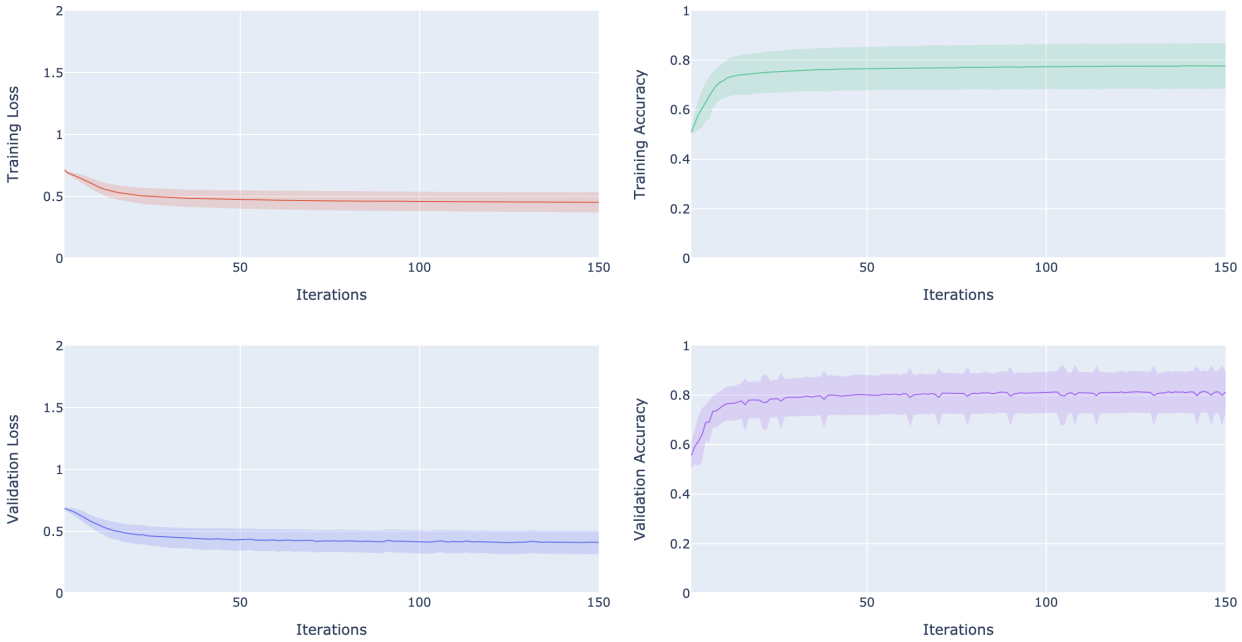

***Figure 24**: Alpha 3 - experiment 2*
dataset A, 10 runs, CPU

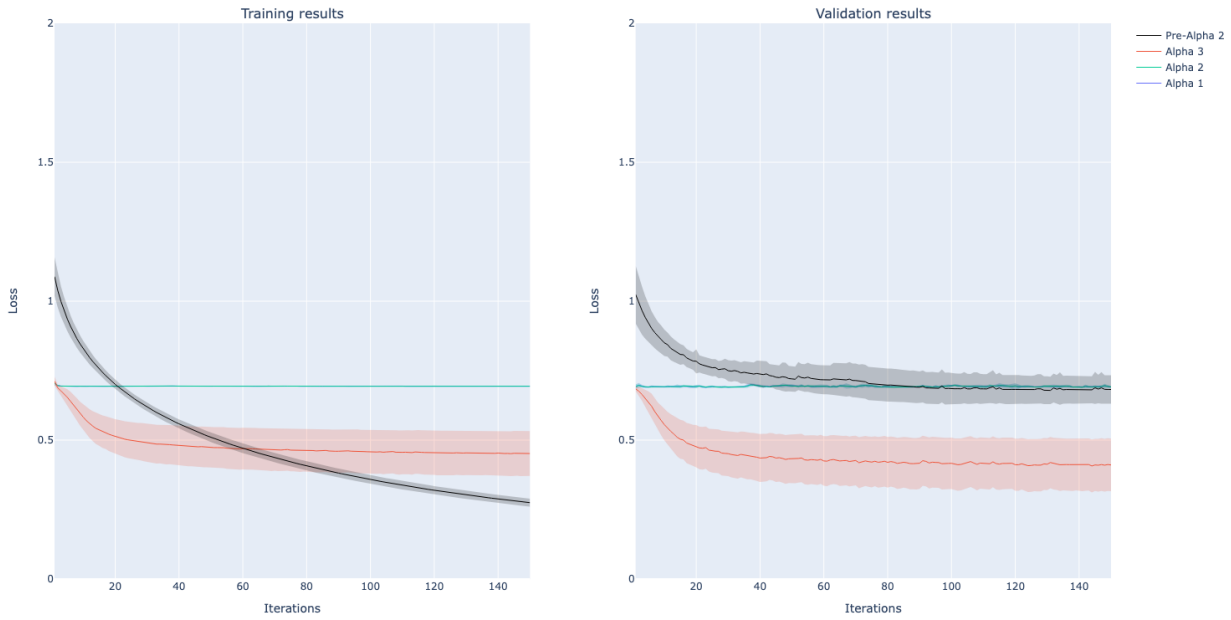

***Figure 25**: All model losses - experiment 2* [2]
dataset A, 10 runs, CPU

Note: Alpha 1 & 2 plots are superposed

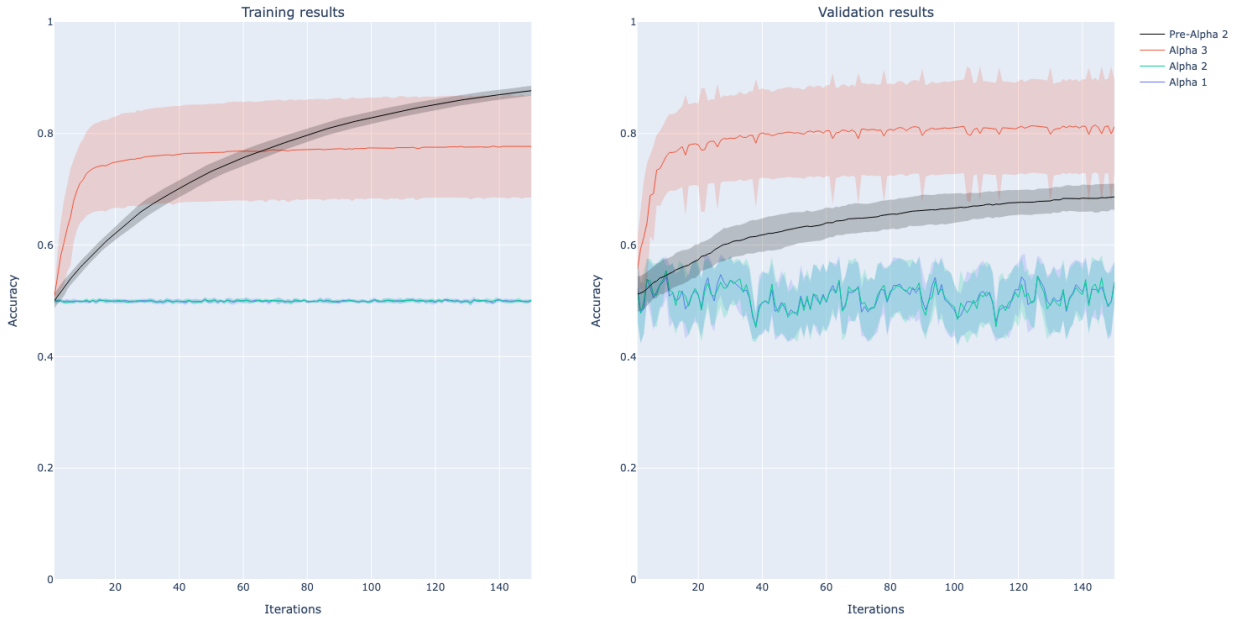

***Figure 26**: All model accuracies - experiment 2*
dataset A, 10 runs, CPU

5.4. Experiment 3

The goal of experiment 3 is to evaluate the performances of the Alpha 3 model without any constraints. Given that this model is the only one which can run on GPU efficiently, this experiment was designed solely for Alpha 3, with 30 runs on the larger dataset A.

model	experiment 3						
	runtime	train loss	valid loss	test loss	train acc	valid acc	test acc
Alpha 3	8.23 ± 0.03	0.44 ± 0.07	0.4 ± 0.08	0.36 ± 0.09	0.79 ± 0.08	0.82 ± 0.07	0.84 ± 0.1

***Table 11**: Experiment 3 results*
dataset A, 30 runs, GPU, Alpha 3 model

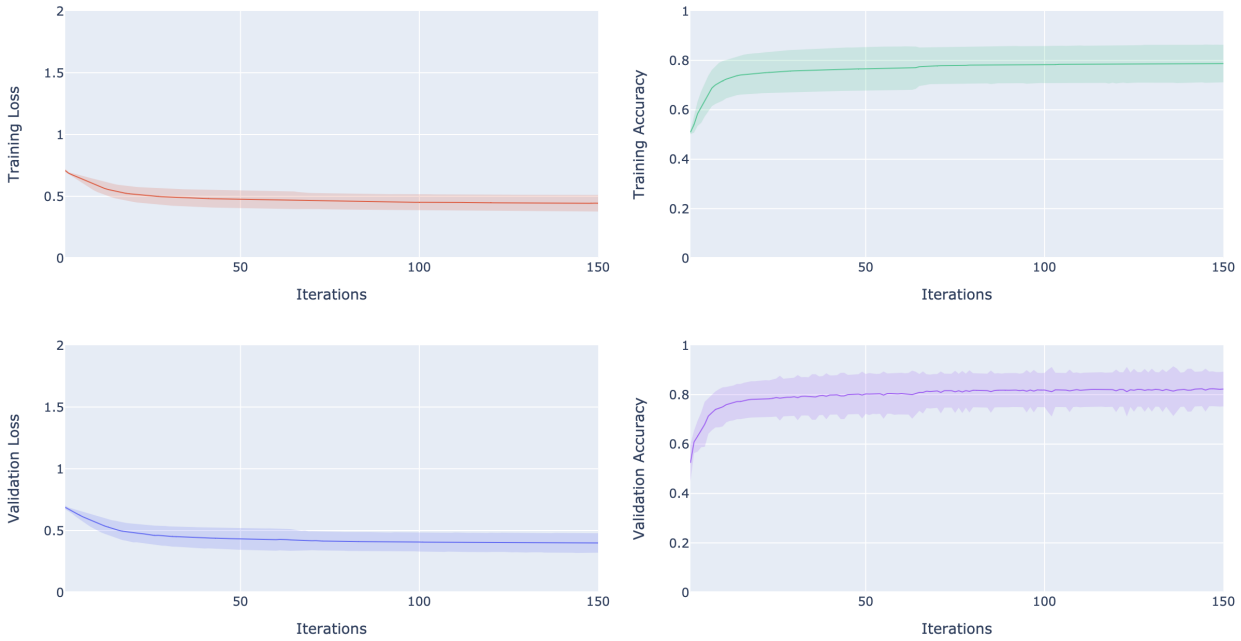

***Figure 27**: Alpha 3 - experiment 3*
dataset A, 30 runs, GPU

In this experiment, the Alpha 3 model reaches a test accuracy of ≈ 0.87 with a very low variance as can be seen in fig. 27. While its runtime is reduced by almost 30% compared to the version running on the CPU.

Note that due to the size of dataset A, one must use a suitable batch size. Using too large a batch size could lead to over-fitting, while one too small could lead to slower training times. Thus, tests were carried out and a balanced batch size was chose, the value of which is shown along other parameter values in tab. 19.

5.5. Experiment 4

Experiment 4 evaluates the Beta 1 model. It involves using different size datasets and also different values of k, which corresponds to the number of nearest neighbors that are considered when making a prediction.

In the Beta model, we are using the training dataset to build a reference or "neighborhood". Meanwhile, the test dataset is used for making predictions and evaluating the model. Indeed for each vector in the test dataset, we compare it to its nearest k neighbors in the training dataset.

To speed up the experiment, the predictions for all k values are computed simultaneously. Rather than executing quantum circuits and computing vector distances separately for each k value, this is done once. Afterwards, these results are used to make predictions based on the specific k value. Thus the execution runtimes for each k value are identical for a specific dataset.

To speed up the experiment, we decided to compute the predictions for all k values simultaneously. Rather than executing quantum circuits and computing vector distances separately for each k, we do it once. Subsequently, we use these results to make predictions based on the specific value of k. Thus, the execution runtimes for each k are identical for a specific dataset.

In order to give a comparative baseline, we also plot the results of a classical KNN algorithm being fed the same input vector (dimensionality-reduced BERT embedding). More on this classical baseline can be found in sec. 8.3.2.

Model	Experiment 4			
	dataset A		dataset B	
	runtime (min)	test acc	runtime (min)	test acc
Beta k=1	2251.31	0.50	270.13	0.53
Beta k=3	2251.31	0.51	270.13	0.54
Beta k=5	2251.31	0.51	270.13	0.53
Beta k=7	2251.31	0.51	270.13	0.55
Beta k=9	2251.31	0.49	270.13	0.55

***Table 12**: Experiment results for the Experiment 4*
dataset A&B, 1 run, CPU

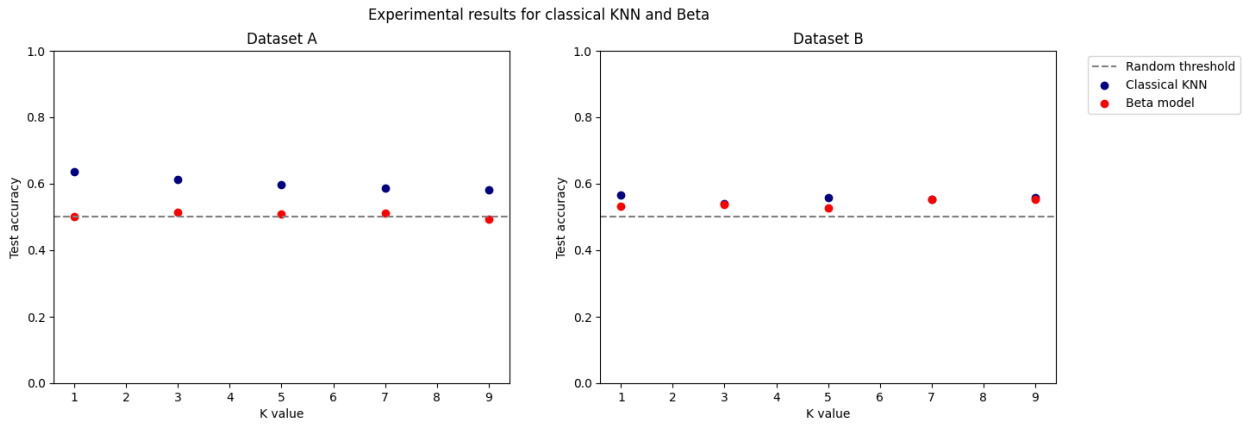

***Figure 28**: Classical KNN vs Beta 1, dataset A&B*

For each tested value of k (1, 3, 5, 7, 9), the Beta 1 model achieves an accuracy in the range of $[0.49, 0.55]$ when using datasets A and B as input, with better accuracy for the smaller dataset B.

5.6. Classical Model Results

For the classical NLP models, we trained one model per embedding model (see Section 4.3 for more details on the embedding models). We trained each model for a maximum number of epochs or till early stopping was reached. System training was stopped early if the validation loss did not improve three consecutive epochs. Convolutional neural network models with word embeddings were trained for a maximum of 30 epochs. Feed-forward networks with sentence embeddings were trained for a maximum of 100 epochs. The LSTM neural network without pre-trained embeddings trained for 58 epochs till early stopping was reached. Training progress graphs for the various models are depicted in:

- For the convolutional neural network based on fastText word embeddings *cc.en.300.bin*, see Figure 29.
- For the shallow feed-forward neural network based on Transformer sentence embeddings *all-distilroberta-v1*, see Figure 30.
- For the shallow feed-forward neural network based on BERT sentence embeddings *bert-base-uncased*, see Figure 31.

- For the convolutional neural network based on BERT word embeddings *bert-base-uncased*, see Figure 32.
- For the Bidirectional LSTM neural network with no pre-trained embeddings, see Figure 33.

Experiments were carried out using the dataset *amazonreview_train_filtered*, specifically the larger version (*dataset A*). Results for the classical models are given in Table 13. The results show that the best validation accuracy was achieved by the model where no pre-trained embeddings were used. This is contrary to typical benchmarks in NLP where models that rely (in some form) on pre-trained embeddings are state-of-the-art. One explanation, which, however, would require further investigations, is that the task of short text classification where the short texts are only up to 6 tokens long poses a challenge for pre-trained embedding models, which are typically trained on typical texts (e.g., for news articles a natural average sentence length is 20 tokens; far from a maximum of 6 tokens). I.e., the models may not be capable of providing good contextualised word or sentence embeddings for such short text segments. Another possible explanation could be that since the dataset features titles of reviews, the titles may already be highly polarised (rather positive or rather negative) and feature strong sentiment-informing words, which in turn may make the sentiment classification task rather simple also for models that do not require pre-trained embeddings. This possible explanation would also fit considering the relatively high sentiment classification performance (and accuracy of up to 90%) of all models.

Embedding model	Model	Valid/test acc.
fastText word emb. *cc.en.300.bin*	Convolutional NN (max sent. len. 6)	0.8718 / 0.9003
Transformer sent. emb. *all-distilroberta-v1*	Shallow feed-forward NN	0.8651 / 0.8775
BERT sent. emb. *bert-base-uncased*	Shallow feed-forward NN	0.8575 / 0.8642
BERT word emb. *bert-base-uncased*	Convolutional NN (max sent. len. 6)	0.8471 / 0.8888
No pre-trained embeddings	Bidirectional LSTM NN (max sent. len. 6)	**0.8727** / 0.9031

***Table 13**: Results of classical model training*

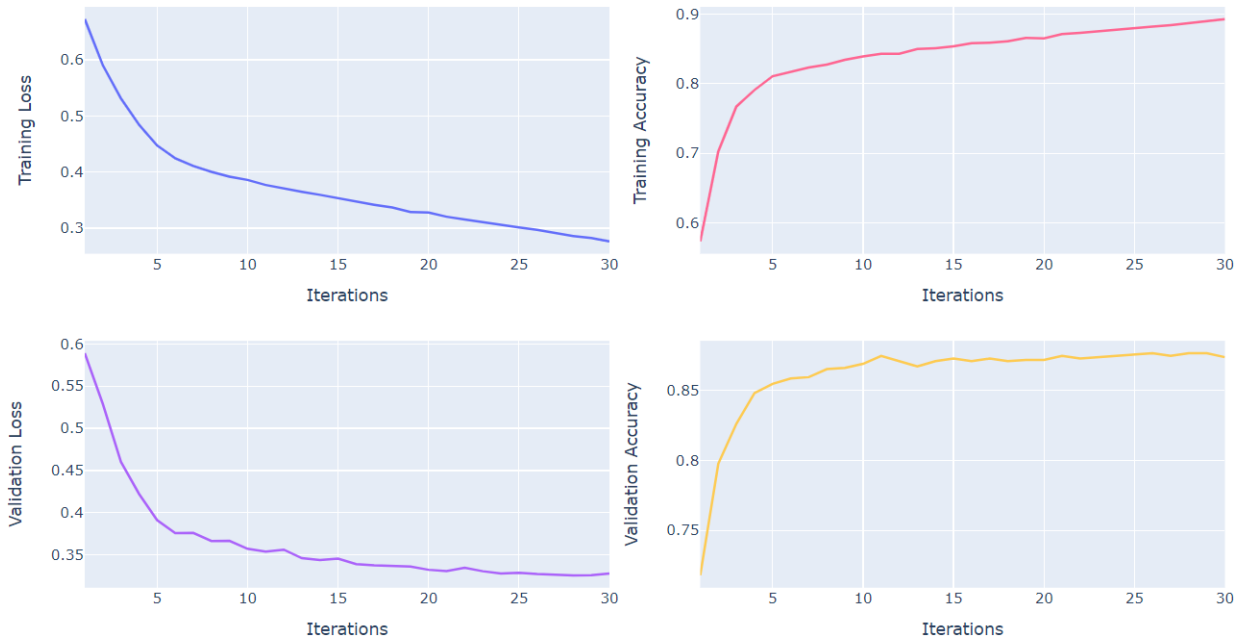

***Figure 29**: Convolutional network with FastText word embeddings*

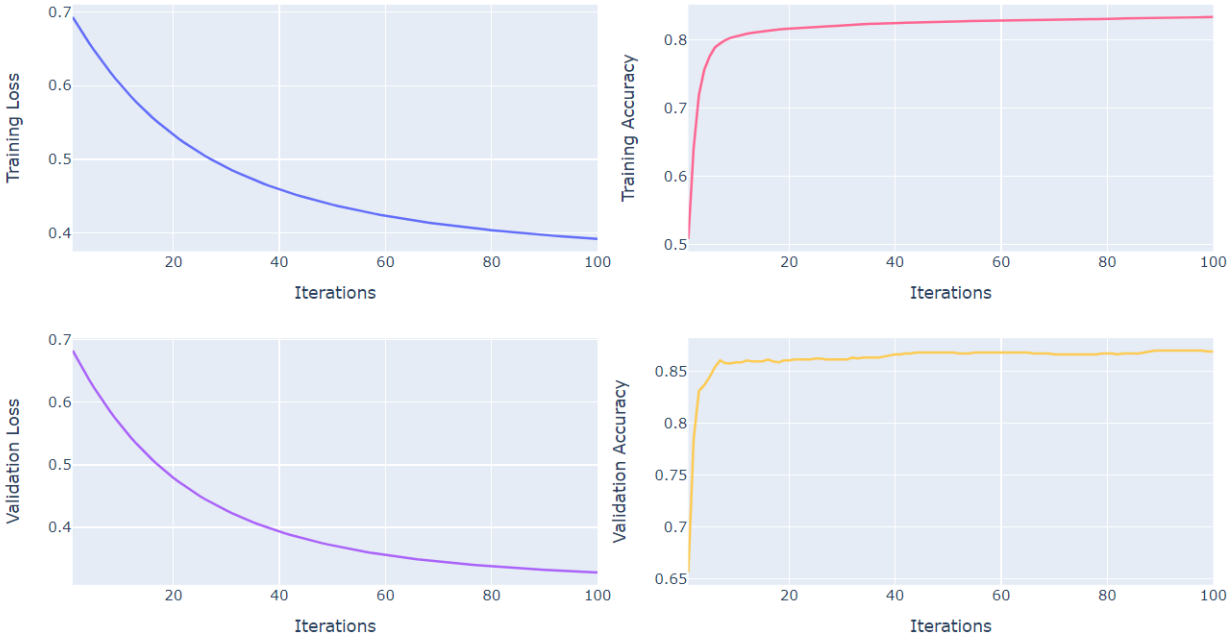

***Figure 30**: Shallow feed-forward neural network with Transformer sentence embeddings*

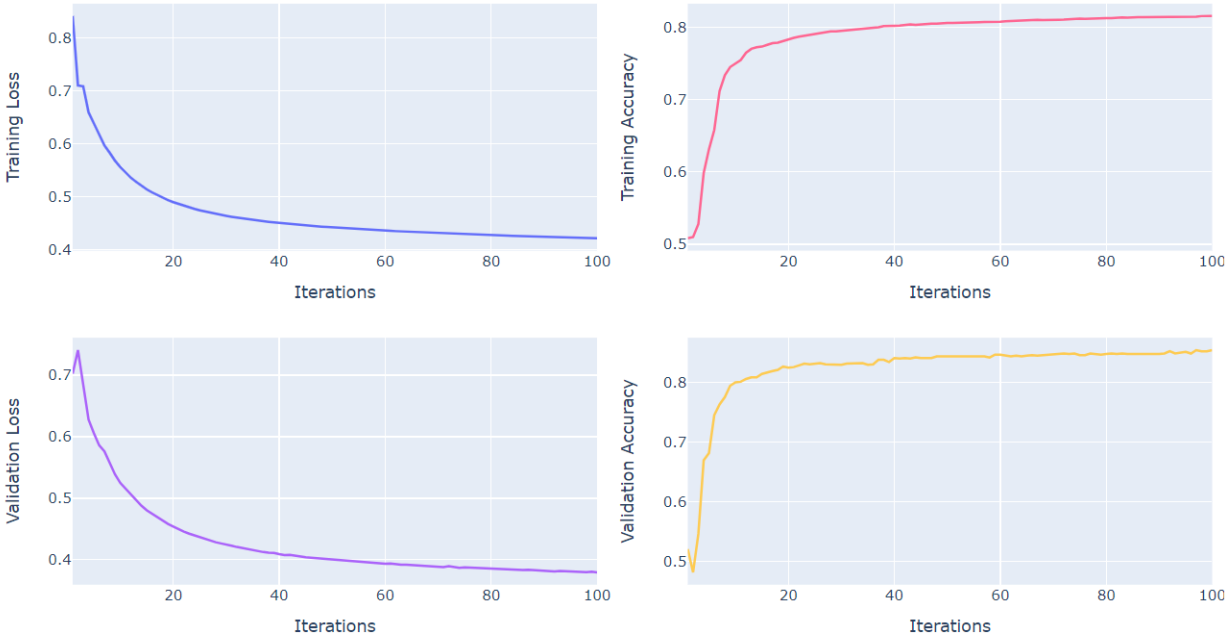

***Figure 31**: Shallow feed-forward neural network with BERT sentence embeddings*

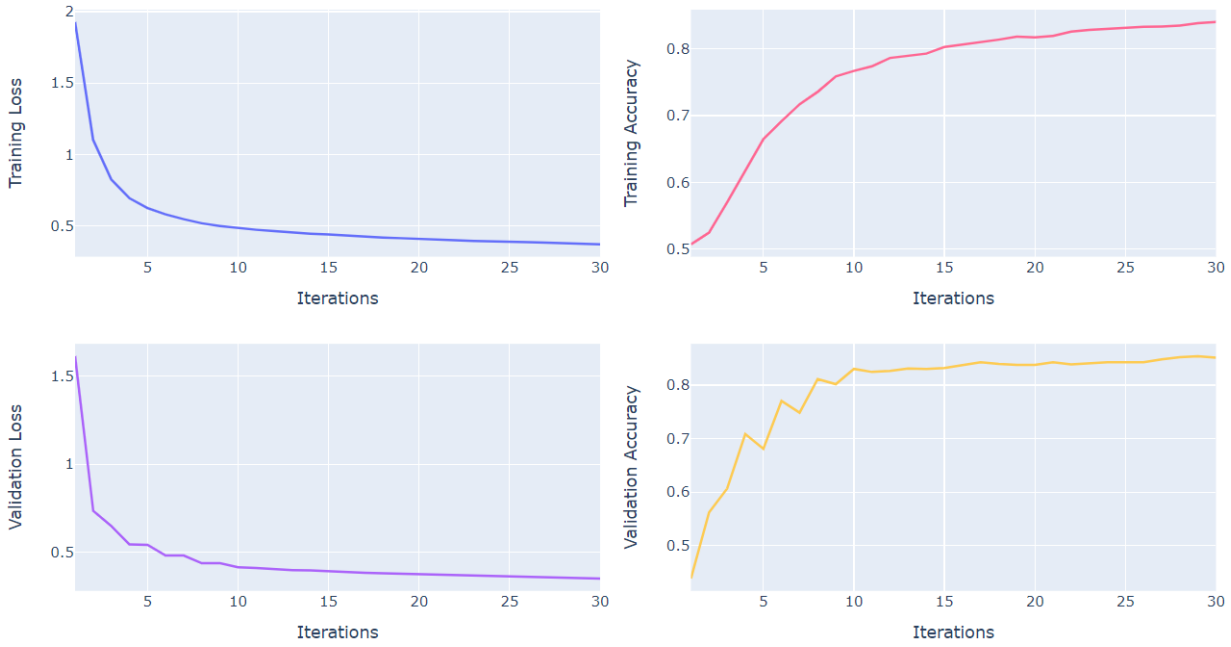

***Figure 32**: Convolutional network with BERT word embeddings*

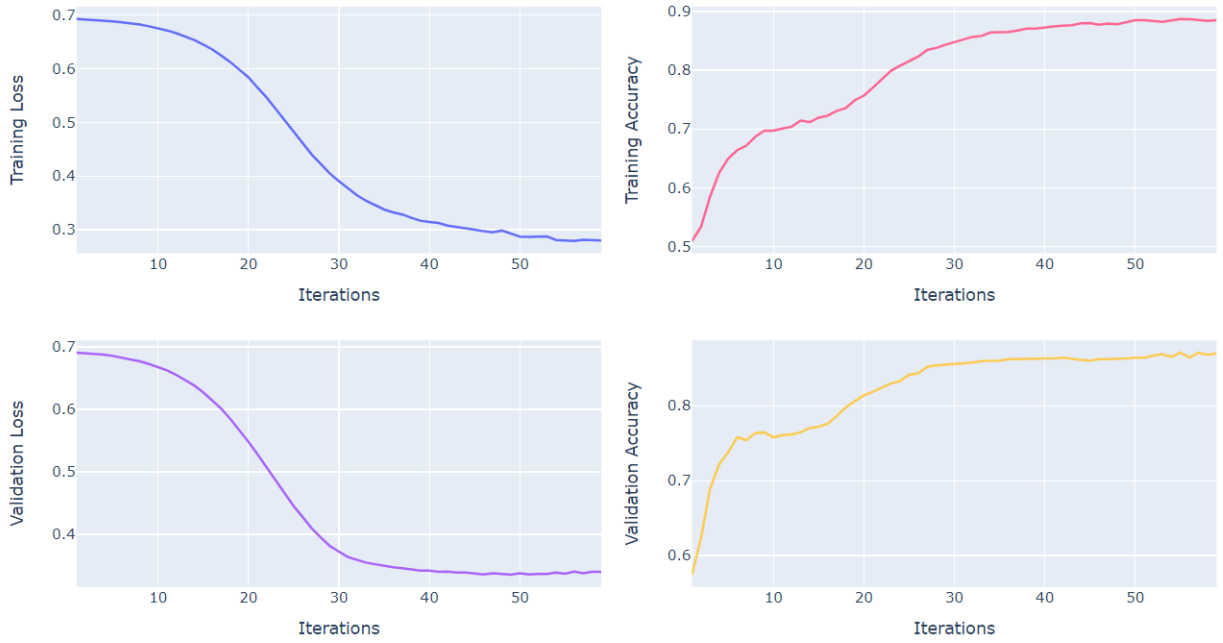

***Figure 33**: Bidirectional LSTM network without pre-trained embeddings*

6. Analysis and discussion

6.1. Losses and accuracies

Since loss and accuracy capture different quantities, it can be interesting to analyse their discrepancies when those occur. However in the experiments for pre-Alphas and Alphas, losses and accuracies hint at no notable divergences in the results. The only noteworthy occurrence that could be investigated is the stable loss for Alpha 1 and 2 while the accuracy oscillates in experiment 1 and 2, most noticeable in experiment 2 because of the lower number of runs the results were averaged over. One noteworthy piece of information is the loss of pre-Alpha models.

In terms of accuracy, it seems indisputable at this stage that Alpha 3 over-performs all other models, irrelevant of the dataset. With accuracies around 0.8 across train, validation and test on the smaller dataset B and accuracies ranging from 0.81 to 0.87 in the larger dataset A, it is the only hybrid candidate which can rival with classical models which results oscillate between 0.81 and 0.90. Note that with the larger dataset A, accuracy over the test set surpassed that of both train and validation sets while training accuracy under-performed both validation and test. This gap which could hint at a possible under-fitting and under-exploitation of the information available is however not large enough to be of concern in this case and we treat it as a positive signal that the model is not over-fitting the training data. Moreover, classical models all display the same behaviour with higher accuracies for test sets than training sets, indicating that it might possibly be linked to the dataset itself rather than the models. Last but not least, the accuracy attained for the large dataset (A) surpasses that of the smaller dataset (B), which suggest that the model has the potential to perform well with larger datasets.

As for other models' accuracies, Alpha 1 and 2 seem to display random behaviours with accuracies in the vicinity of 50%, be it over the training or validation data. Pre-Alpha 1 at least seems capable of some learning since its accuracy increases over time in the training set, however the model doesn't generalise and its performance drops down to the level of randomness of Alpha 1 and 2 for the validation set. On test data, both Alpha 1 and 2 perform completely randomly with an accuracy of 0.5 in both cases. Slight improvements can be noted for pre-Alpha 1 and 2 but given their poor performance on the validation set, the slightly improved performance over the test set should be interpreted with a critical eye.

The Alpha 1 & 2 experimental results could be explained by their intrinsic architecture. Indeed as it can be seen fig. 6&7, an embedding is fed to the quantum circuit as parameters. However, this approach can lead to several issues. First, the input embedding is split based on the number of parameters, potentially resulting in a significant loss of information, potentially rendering it useless. Moreover BERT embeddings do not take into account the order of the words in a sentence. Meaning that the sentences *"Bob loves Alice"* and *"Alice is loved by Bob"* will have very similar BERT embeddings. However, the generated circuits for these two sentences will differ significantly, both in terms of architecture and the number of parameters. These aspects of the Alpha 1 and 2 models could potentially explain why they are unable to effectively learn from a training dataset.

6.2. To DisCoCat or not to DisCoCat

One - and perhaps the most - interesting takeaway from these experiments is that the DisCoCat model of computation doesn't necessarily improve a model's performance. Alpha 3, the only of our 5 models besides Beta 1 which doesn't rely on DisCoCat clearly outperforms all other models, which should at first be unintuitive: why would more information deteriorate results? The DisCoCat paradigm is meant to encode extra grammatical information, which should enhance model performances. However several drawbacks should be noted which could adversely impact the DisCoCat framework. First and foremost, DisCoCat was formulated to support strictly grammatical sentences. Because the model was developed based on Lambek's pre-group grammar (Lambek, 1999), it relies heavily on grammaticality to build meaning. However, the datasets used to benchmark the models here included some sentence structures which are not strictly grammatical. Relaxed rules of classical NLP are able to accommodate for such minor deviations and so can support non-strict grammaticality well enough. However given the way DisCoCat was designed one can wonder how well it can accommodate non-grammaticality. This result prompts us to list some items for future work in order to investigate the question of whether DisCoCat-based models would have performed

better had the dataset been made solely of strictly grammatical sentences. However, irrelevant of how well DisCoCat can perform on grammatical sentences, the current results raise the question of whether this framework is well-adapted to natural language which is commonly accepted to be full of valid yet ungrammatical instances. A model requiring strict grammaticality to be informative might prove too restrictive for use on natural language.

Another possible interpretation of the degraded results of DisCoCat-based models could be obtained by comparing it with the performances of classical models. Results in table 13 show that the best performer on test set is the bidirectional LSTM fed with no pre-trained embeddings. Instead of relying on informative word or sentence embeddings such as BERT and FastText, this model builds its representation of the sentences from scratch. How then, can it outperform its informed counterparts? A possible explanation hints at the short length of the sentences. As opposed to standard natural language sentences which have an average count of 20 words ± 5 depending on the language- the sentences studied here are limited to 6 words. Moreover, given the nature of the texts - review titles - the vocabulary in use only spans a small portion of the existing words in a language. Both of these facts could lead the model to need less information that what is provided by classical embeddings such as BERT or FastText, which would in that case hamper results by creating noise. In this case, the same behaviour would not be observed when scaling to longer sentences. This could suggest that the shortness of sentences combined with to the potential repetitivity of the vocabulary of the dataset could cause information to behave as noise, and it might be the same reason that would underly the worsening of results when injecting DisCoCat information. In which case the reason for DisCoCat-based models to under-perform would be how short the sentences are and how repetitive the vocabulary is, leaving room for improved performance in the presence of more natural sentences (longer and more varied).

6.3. Simple vs. complex

Architectural minimalism taught us that *less is more* and the current results seem to go in this direction with Alpha 3 being a rather pared-down version and best performer. However, more experiments are needed before concluding that a simpler model performs better. For instance, given that no specific method has been used for hyper-parameter search, it is possible that the more complex model's performance relies more heavily on those while the simplest model can do away despite suboptimal hyperparameters. Another possibility could be that the simpler model is well-suited to a *simple* dataset made of short sentences built out of high-frequency words but would drastically under-perform when scaled to larger and more complex sentences. At this experimental stage, we therefore advocate for a conservative outlook and prefer to withhold too general conclusions regarding relative model performance gaps.

6.4. The specific case of Beta 1

The current implementation of the Beta 1 model is in its early stages and we consider it to be currently insufficient for meaningful conclusions to be drawn. In its current state, Beta 1 can only take vectors of dimension 2 as input. Given that inputs are dimensionality-reduced word-embeddings of several hundred dimensions initially, reducing them to 2 dimensions seems unlikely to yield encouraging results. In order to study the minimal number of qubits needed for this approach to start making sense, we used the same dimensionality-reduction technique and inputted the resulting reduced vectors into a classical KNN, sec. 8.3.2. This side experiment revealed encouraging results since a phase transition seem to happen around dimension 16. As shown in the KNN experiments on both the large and smaller datasets, results improve drastically upon reaching dimension 16 for the input vector. What is more, further increasing the dimension of the input vector doesn't seem to improve the results by much. On the large dataset, a KNN with 16-dimensional reduced BERT word embeddings yields accuracies between 0.75 and 0.76 for the binary classification while a 512-dimensional reduced BERT vector only yields results between 0.79 and 0.81, thus displaying at most a 0.06 improvement for about 500 more dimensions of the input vector. This hints at the possibility to scale up the model to dimension 16 on NISQ devices and potentially obtain viable results.

Beta 1's extensions are of particular interest because they circumvent the need for long training that other models can have.

6.5. Runtimes

Given the differences in supported hardware, the runtime of Alpha 3 is clearly superior to others. This is however mostly due to the model relying on GPUs. Other models showcase similar runtimes of approximately 20 hours for the larger dataset A.

6.6. Vocabulary flexibility

One of the core improvements of Alpha models over pre-Alpha ones was to remove the limitations on the words included in the validation and test datasets. By relying on embeddings which conveyed semantic and grammatical information, the Alpha models eliminated the prerequisite pre-Alpha had to have access to the full vocabulary during the training phase.

6.7. CPU, GPU and parallelism

Most models aren't designed to take advantage of multi-core parallelism or GPU acceleration, giving an edge to those which are equipped with this capability. Both pre-Alpha 2 and Alpha 3 are the exceptions to this rule.

Based on lambeq's training module, pre-Alpha 2 should be able to advantage of GPU acceleration. However, experimental data suggested this functionality of the library is currently under-performing by using only a single GPU and no more than 10% of said GPU's power. Moreover, experiments run on CPU showed a better performance than those on GPU. This motivated the choice to exclude pre-Alpha 2 from the GPU-based experiment 3.

As for Alpha 3, the only model capable of harnessing both GPU acceleration and multi-core parallelism, it only managed to take advantage of a single GPU. However, because the exploitation of the GPU was more efficient than for pre-Alpha 2, the model displayed a better time performance. Moreover, running on CPUs, Alpha 3 takes advantage of the whole 40 CPUs in the cluster node, making it highly efficient time-wise.

6.8. Trade-offs between models and best overall model

At this stage, with no hyper-parameter optimisation and for this specific dataset that we have benchmarked our models on, Alpha 3 undeniably outperforms other models. Insufficient experimental data doesn't allow us for now to draw generalised conclusion as to which model could perform better in general over any sort of dataset or even whether the current trend would persist should hyper-parameter optimisation be pursued.

7. Conclusion

7.1. Limitations

7.1.1. Taking advantage of hardware

The current models do not take full advantage of multi-CPU parallelism or GPU acceleration, which can speed up a number of processes such as parameter optimisation, sentence parsing/vectorisation, parallel execution of circuits, etc. Either due to design flaws or the libraries used, these issues mean increased computational times. For instance in the case of pre-Alpha 2, though the model uses the lambeq library which should be taking advantage of GPUs, only 10% of a single GPU is currently being exploited, which is suboptimal. On a similar topic, the number of CPU cores used should be increased.

7.1.2. Constrained datasets

Despite being natural language datasets, the filtering applied on them to constrain both the sentence length to a maximum of 6 words and only 5 different sentence structures results in a highly engineered dataset. One should wonder how representative results obtained on such specifically-engineered dataset would be able to generalise to different/longer sentence structures. The constraint on sentence structures stems from the way the pre-Alpha 1 model is built, since it requires Ansätze to be hand-crafted, this entails the need to limit the number of supported sentences.

7.1.3. Lambeq-based limitations

Despite providing a useful framework for QNLP tasks, the lambeq library also comes with an array of limitations that it introduces in the work, making it worthwhile to wonder whether the advantages it confers outweigh the drawbacks it entails. The major limitation introduced by lambeq-based models is the necessity for models to be initialised with *all* the words present in the vocabulary. This means that, though of course the 23 words in the validation and test set do not need to be *trained*, they however still need to be made available to the model during initialisation. Failure to do so results in impossibility of the model to tackle any of the previously unseen words. Another limitation is that In lambeq-based models, the number of trainable parameters is directly correlated to the input vocabulary and scales linearly with it. This raises the question of scalability of models when faced with larger and larger vocabularies.

7.1.4. Scaling to multi-class classification

Existing versions of pre-Alpha can not be straightforwardly adapted to multi-class classification. The same is true for Beta 1.

7.1.5. Current state of Beta 1

In its current state, the Beta 1 model can only support dimension-2 vectors as input, making it impractical for use.

7.2. Future work

7.2.1. Scaling up Beta 1 to support higher input dimensions

The current version of Beta 1 supporting only a 2-dimensional input vector was a first step towards an improved implementation which will cater to larger input dimensions. The minimal goal is to implement a version with input size 8 since this is the lower threshold for results to start showing non-random performance in the case of classical KNNs and we infer that the same should apply for the quantum version. The reach goal would be to scale up our input dimension to 16 that it becomes possible to analyse whether the

same phase transition that we observe in the classical case occurs in the quantum case too. If this go as planned based on the classical model, a visible improvement in accuracy should be reached at that stage.

7.2.2. Upgrading to multi-class classification

One of the core areas of improvement that will be pursued in the next steps will be to move away from simple binary classification and dig into multi-class classification. We expect challenges in intent identification and sentiment analysis as the border between different classes becomes more blurred. The first step will be to look at 3-class classification with positive, neutral and negative. The reach goal would be to look at 5 classes thus including very positive, positive, neutral, negative, very negative. However those fine-grained tasks being occasionally challenging even for classical models depending on how they are phrased could prove difficult to tackle with hybrid models.

7.2.3. Analysing model's precision and recall

Improving performance monitoring is another direction that will be explored in the next steps of the project. Precision quantifies the proportion of correctly predicted positive instances out of all instances that the model predicted as positive $\frac{TP}{TP+FP}$. Recall quantifies the proportion of all positive instances out of the total actual positive instances $\frac{TP}{TP+FN}$. Computing precision and recall will complement the existing loss and accuracy, allowing for a more detailed overview of the model's performance.

7.2.4. Modifying data encoding

One of the potentially key components impacting the performance of our models is the data encoding. The vectors which we feed into the different models should contain sufficient information to allow models to leverage it while avoiding containing too much information which could act as noise and hinder a model's performance. A first component of the data encoding pipeline is of course the choice of embedding. A wide range of embeddings exist and so far BERT was selected because it performed best for the dataset at hand on classical models. However, one could wonder whether being best suited to classical models necessarily equates to being best suited for any hybrid model. Given the fact that some of the hybrid models rely on the DisCoCat paradigm which is quite different from the existing classical approaches, it would be worth investigating if changing the input embeddings might not prove beneficial to the lambeq-based models for instance. Similarly, since the hybrid models require dimensionality reduction techniques, it will be beneficial to look into whether some embeddings better resist dimensionality reduction than others. Though BERT proved to outperform other classical embeddings available in its full length, it might be possible for other embeddings to yield better performances when submitted to hard dimensionality cuts imposed by the NISQ regime on hybrid models. Finally, we also aim to look into different ways of encoding information into the quantum system (Chen et al., 2020; Liang et al., 2020; Mahmud et al., 2020; Mancilla & Pere, 2022).

7.2.5. Software

As part of future deliverables, the developed library will be made available as a python package on PyPI.

7.2.6. Hyper-parameter optimisation

Hyper-parameters search and tuning are essential steps in optimizing deep learning models. Determining the most effective way to configure individual hyper-parameters and their interactions for a specific dataset is a complex and time consuming task. Multiple search methods exist to discover hyper-parameters that lead to optimal model performance with a given dataset. Part of the future work will be focused on finding better hyper-parameters. The first and simplest approach will be a grid search, which we will later upgrade to more efficient techniques such as random search (Bergstra & Bengio, 2012), Bayesian optimisation (Kandasamy et al., 2018) and genetic algorithms (Aszemi & Dominic, 2019).

Bibliography

Abramsky, S., & Coecke, B. (2004). A categorical semantics of quantum protocols. *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, 415–425.

Abramsky, S., & Coecke, B. (2008). Handbook of quantum logic and quantum structures, volume 2, chapter categorical quantum mechanics.

Al-Ghuribi, S. M., & Noah, S. A. M. (2021). A comprehensive overview of recommender system and sentiment analysis. *arXiv preprint arXiv:2109.08794*.

Alsmadi, I., & Gan, K. H. (2019). Review of short-text classification. *International Journal of Web Information Systems*, *15*(2), 155–182.

Aszemi, N. M., & Dominic, P. (2019). Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, *10*(6).

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M. S., Alonso-Linaje, G., Akash-Narayanan, B., Asadi, A., Arrazola, J. M., Azad, U., Banning, S., Blank, C., Bromley, T. R., Cordier, B. A., Ceroni, J., Delgado, A., Matteo, O. D., . . . Killoran, N. (2022). Pennylane: Automatic differentiation of hybrid quantum-classical computations.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: Analyzing text with the natural language toolkit*. " O'Reilly Media, Inc."

Blagec, K., Dorffner, G., Moradi, M., & Samwald, M. (2021). A critical analysis of metrics used for measuring progress in artificial intelligence.

Burchell, L., Birch, A., Bogoychev, N., & Heafield, K. (2023). An open dataset and model for language identification. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 865–879. https://doi.org/10.18653/v1/2023.acl-short.75

Chen, S. Y.-C., Huang, C.-M., Hsing, C.-W., & Kao, Y.-J. (2020). Hybrid quantum-classical classifier based on tensor network and variational quantum circuit. *arXiv preprint arXiv:2011.14651*.

Coecke, B., & Kissinger, A. (2018). Picturing quantum processes: A first course on quantum theory and diagrammatic reasoning. *Diagrammatic Representation and Inference: 10th International Conference, Diagrams 2018, Edinburgh, UK, June 18-22, 2018, Proceedings 10*, 28–31.

Coecke, B., Sadrzadeh, M., & Clark, S. (2010). Mathematical foundations for a compositional distributional model of meaning. *arXiv preprint arXiv:1003.4394v1*.

de Felice, G., Toumi, A., & Coecke, B. (2021). DisCoPy: Monoidal categories in python. *Electronic Proceedings in Theoretical Computer Science*, *333*, 183–197. https://doi.org/10.4204/eptcs.333.13

Demner-Fushman, D., Chapman, W. W., & McDonald, C. J. (2009). What can natural language processing do for clinical decision support? *Journal of biomedical informatics*, *42*(5), 760–772.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186.

Feng, F., Yang, Y., Cer, D., Arivazhagan, N., & Wang, W. (2022). Language-agnostic bert sentence embedding. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 878–891.

Fix, E., & Hodges, J. L. (1989). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, *57*(3), 238–247.

Gasparetto, A., Marcuzzo, M., Zangari, A., & Albarelli, A. (2022). A survey on text classification algorithms: From text to predictions. *Information*, *13*(2), 83.

Giovannetti, V., Lloyd, S., & Maccone, L. (2008). Quantum random access memory. *Physical review letters*, *100*(16), 160501.

Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., & Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. *Nature*, *567*(7747), 209–212.

Honnibal, M., & Johnson, M. (2015). An improved non-monotonic transition system for dependency parsing. *Proceedings of the 2015 conference on empirical methods in natural language processing*, 1373–1378.

Hutto, C., & Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the international AAAI conference on web and social media*, *8*(1), 216–225.

Jin, Z., & Mihalcea, R. (2022). Natural language processing for policymaking. In *Handbook of computational social science for policy* (pp. 141–162). Springer International Publishing Cham.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Kalra, S., & Prasad, J. S. (2019). Efficacy of news sentiment for stock market prediction. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 491–496.

Kandasamy, K., Krishnamurthy, A., Schneider, J., & Póczos, B. (2018). Parallelised bayesian optimisation via thompson sampling. *International Conference on Artificial Intelligence and Statistics*, 133–142.

Kang, M.-S., Heo, J., Choi, S. G., Moon, S., & Han, S.-W. (2019). Implementation of swap test for two unknown states in photons via cross-kerr nonlinearities under decoherence effect. *Scientific Reports*, *9*. https://api.semanticscholar.org/CorpusID:117725177

Kartsaklis, D., Fan, I., Yeung, R., Pearson, A., Lorenz, R., Toumi, A., de Felice, G., Meichanetzidis, K., Clark, S., & Coecke, B. (2021). Lambeq: An efficient high-level python library for quantum nlp.

Kenton, J. D. M.-W. C., & Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 4171–4186.

Kerenidis, I., Landman, J., Luongo, A., & Prakash, A. (2018). Q-means: A quantum algorithm for unsupervised machine learning.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, 177–180.

Lambek, J. (1999). Type grammar revisited. *Logical Aspects of Computational Linguistics: Second International Conference, LACL'97 Nancy, France, September 22-24, 1997 Selected Papers 2*, 1–27.

Lambek, J. (2008). *From word to sentence: A computational algebraic approach to grammar*. Polimetrica sas.

Liang, J.-M., Shen, S.-Q., Li, M., & Li, L. (2020). Variational quantum algorithms for dimensionality reduction and classification. *Phys. Rev. A*, *101*, 032323. https://doi.org/10.1103/PhysRevA.101.032323

Lloyd, S., Mohseni, M., & Rebentrost, P. (2013). Quantum algorithms for supervised and unsupervised machine learning.

Lorenz, R., Pearson, A., Meichanetzidis, K., Kartsaklis, D., & Coecke, B. (2023). Qnlp in practice: Running compositional models of meaning on a quantum computer. *Journal of Artificial Intelligence Research*, *76*, 1305–1342.

Maćkiewicz, A., & Ratajczak, W. (1993). Principal components analysis (pca). *Computers Geosciences*, *19*(3), 303–342. https://doi.org/https://doi.org/10.1016/0098-3004(93)90090-R

Mahmud, N., Haase-Divine, B., MacGillivray, A., & El-Araby, E. (2020). Quantum dimension reduction for pattern recognition in high-resolution spatio-spectral data. *IEEE Transactions on Computers*, *71*(1), 1–12.

Mancilla, J., & Pere, C. (2022). A preprocessing perspective for quantum machine learning classification advantage in finance using NISQ algorithms. *Entropy*, *24*(11), 1656. https://doi.org/10.3390/e24111656

Mari, A., Bromley, T. R., Izaac, J., Schuld, M., & Killoran, N. (2020). Transfer learning in hybrid classical-quantum neural networks. https://quantum-journal.org/papers/q-2020-10-09-340/

McAuley, J. J., & Leskovec, J. (2013). From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. *Proceedings of the 22nd international conference on World Wide Web*, 897–908.

Meichanetzidis, K., Gogioso, S., De Felice, G., Chiappori, N., Toumi, A., & Coecke, B. (2020). Quantum natural language processing on near-term quantum computers. *arXiv preprint arXiv:2005.04147*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, *2*(5), 183–197. https://doi.org/https://doi.org/10.1016/0925-2312(91)90023-5

NewsCatcher. (2020). Topic Labeled News Dataset. https://www.kaggle.com/datasets/kotartemiy/topic-labeled-news-dataset

Orús, R. (2014). A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, *349*, 117–158.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S.,

Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.
Rich, E. (1979). User modeling via stereotypes. *Cognitive science*, *3*(4), 329–354.
Sarma, A., Chatterjee, R., Gili, K., & Yu, T. (2019). Quantum unsupervised and supervised learning on superconducting processors. *arXiv preprint arXiv:1909.04226*.
Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics*, *24*(1), 97–123.
Schwenk, H., & Douze, M. (2017). Learning joint multilingual sentence representations with neural machine translation. *Proceedings of the 2nd Workshop on Representation Learning for NLP*, 157–167.
Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*, 379–423. Retrieved April 22, 2003, from http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf
Shepherd, D., & Bremner, M. J. (2008). Instantaneous quantum computation. *arXiv preprint arXiv:0809.0847*.
Sim, S., Johnson, P. D., & Aspuru-Guzik, A. (2019). Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, *2*(12), 1900070. https://doi.org/https://doi.org/10.1002/qute.201900070
Song, G., Ye, Y., Du, X., Huang, X., & Bie, S. (2014). Short text classification: A survey. *Journal of Multimedia*, *9*(5).
Thakkar, G., & Pinnis, M. (2020). Pretraining and fine-tuning strategies for sentiment analysis of latvian tweets. In *Human language technologies–the baltic perspective* (pp. 55–61). IOS Press.
Villalpando, A., Balodis, K., Krišlauks, R., & Kannan, V. (2021). NEASQC D6.3 QNLP pre-alpha prototype. https://www.neasqc.eu/wp-content/uploads/2021/09/NEASQC_D6.3_WP6_QNLP_PreAlpha_final.pdf
Wankhade, M., Rao, A. C. S., & Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, *55*(7), 5731–5780.
Weld, H., Huang, X., Long, S., Poon, J., & Han, S. C. (2022). A survey of joint intent detection and slot filling models in natural language understanding. *ACM Computing Surveys*, *55*(8), 1–38.
Wiebe, N., Kapoor, A., & Svore, K. (2014). Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv preprint arXiv:1401.2142*.
Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.
Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., & Mann, G. (2023). Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.
Zeng, W., & Coecke, B. (2016). Quantum algorithms for compositional natural language processing. *arXiv preprint arXiv:1608.01406*.
Zhang, W., Deng, Y., Liu, B., Pan, S. J., & Bing, L. (2023). Sentiment analysis in the era of large language models: A reality check. *arXiv preprint arXiv:2305.15005*.
Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. *Advances in neural information processing systems*, *28*.

8. Appendix

8.1. Instructions to Use

8.1.1. Quantum Models

1. First, follow installation and environment set-up and activation instructions here.
2. Change directory to the root of the repository via:

   ```
   cd /path/to/repo/WP6_QNLP/neasqc_wp61
   ```

   where `/path/to/repo` is the path to the cloned repository in your local machine.
3. Now from this directory you may run any quantum model you wish. Below are sample commands for running each of these models. Please note that all the parameters in these commands are set to some value for illustrative purposes, but you may change these values as you wish. Please refer to sec. 8.2 for a description of these parameters.

   pre-Alpha 1

   ```
   bash 6_Classify_With_Quantum_Model.sh -m pre_alpha_1
   -t ./data/datasets/reduced_amazonreview_train.tsv
   -v ./data/datasets/reduced_amazonreview_test.tsv
   -j ./data/datasets/reduced_amazonreview_val.tsv
   -s 13 -r 10 -i 150 -p COBYLA
   -o ./benchmarking/results/raw/
   ```

   pre-Alpha 2

   ```
   bash 6_Classify_With_Quantum_Model.sh -m pre_alpha_2
   -t ./data/datasets/reduced_amazonreview_train.tsv
   -v ./data/datasets/reduced_amazonreview_test.tsv
   -j ./data/datasets/reduced_amazonreview_val.tsv
   -s 13 -p Adam -i 150 -r 10 -a IQP -q 1 -n 1 -x 3 -b 512
   -o ./benchmarking/results/raw/
   ```

   Alpha 1

   ```
   bash 6_Classify_With_Quantum_Model.sh -m alpha_1_2
   -t ./data/datasets/reduced_amazonreview_train_word.csv
   -j ./data/datasets/reduced_amazonreview_dev_word.csv
   -v ./data/datasets/reduced_amazonreview_test_word.csv
   -s 13 -r 10 -i 150 -b 512 -l 2e-1 -w 0 -z 150 -g 1
   ```

```
-y alpha_1 -c 22 -e 1 -a IQP

-q 1 -n 1 -x 3 -o ./benchmarking/results/raw/
```

Alpha 2

```
bash 6_Classify_With_Quantum_Model.sh -m alpha_1_2

-t ./data/datasets/reduced_amazonreview_train_sentence.csv

-j ./data/datasets/reduced_amazonreview_dev_sentence.csv

-v ./data/datasets/reduced_amazonreview_test_sentence.csv

-s 13 -r 2 -i 150 -b 512 -l 2e-1 -w 0 -z 150 -g 1

-y alpha_2 -c 22 -e 1 -a IQP -q 1 -n 1 -x 3

-o ./benchmarking/results/raw/
```

Alpha 3

```
bash 6_Classify_With_Quantum_Model.sh -m alpha_3

-t ./data/dataset/reduced_amazonreview_train_sentence.csv

-j ./data/dataset/reduced_amazonreview_dev_sentence.csv

-v ./data/dataset/reduced_amazonreview_test_sentence.csv

-s 13 -r 10 -i 150 -u 3 -d 0.01 -b 512 -l 2e-1 -w 0 -z 150

-g 1 -o ./benchmarking/results/raw/
```

Beta

```
bash 6_Beta.sh

-t ./data/datasets/reduced_amazonreview_train_sentence.csv

-v ./data/datasets/reduced_amazonreview_test_sentence.csv

-k "1 3 5 7 9" -o ./benchmarking/results/raw/
```

4. Running any of the above will produce a JSON file, which we will generically refer to as `results.json`. Assume this file is located in `/output/path/`. To plot the results stored in this file use the following:

   pre-Alpha and Alpha models

   ```
   python3.10
   /benchmarking/results_processing/plot_single_experiments_results.py

   /output/path/results.json
   ```

   Beta 1 model

   ```
   python3.10
   ```

```
/benchmarking/results_processing/plot_beta_experiment.py

/output/path/results.json
```

This will open a browser window with the plot of the results, which one can download in `.png` format by clicking the small camera icon in the top-right corner of the browser window.

8.1.2. Classical Models

To set up the environment, follow the instructions provided here.

The full workflow for data preparation, model training, and evaluation is divided into the following eight steps:

- Step 0 - Data download.
- Step 1 - Tokenizing and parsing datasets using the Bobcat parser.
- Step 2 - Selecting data with the pre-defined syntactical structures.
- Step 3 - Splitting data into train/dev/test parts.
- Step 4 - Acquiring embedding vectors using a selected pre-trained embedding model.
- Step 5 - Training the model.
- Step 6 - Using the classifier (inferencing).
- Step 7 - Evaluating results.

Bash script files for each step are located in the root of the repo. The detailed description of each step is included in the doc subdirectory.

Based on the Amazon reviews dataset, two datasets are created (each dataset is split into train/development/test parts):

- **amazonreview_train_filtered** with 18961 examples for training, 1053 examples for development, and 1053 examples for testing.
- **reduced_amazonreview_pre_alpha** with 3300 examples for training, 700 examples for development, and 700 examples for testing.

Prepared datasets are located here.

Step 4 of the workflow involves acquiring sentence or word embeddings. The script *4_GetEmbeddings.sh* has the following parameters:

- -i *(input file)*: Input file with text examples
- -c *(column)*: '3' – if 3-column input file containing class, text and parse tree columns; '0' – if the whole line is a text example
- -m *(embedding name)*: Name of the embedding model
- -t *(embedding model type)*: Type of the embedding model - 'fasttext', 'transformer' or 'bert'
- -e *(embedding unit)*: Embedding unit: 'sentence' or 'word'
- -g *(gpu use)*: Index of the GPU to use (from 0 to available GPUs - 1), -1 if only CPU should be used (default is -1)

The FastText model works only on the CPU. Embedding unit is 'word' or 'sentence' for the 'bert' models, 'word' for the 'fasttext' model, and 'sentence' for the 'transformer' models. Run this step for all 3 dataset parts - train, dev and test.

Example:

```
bash 4_GetEmbeddings.sh \
-i ./data/datasets/amazonreview_train_filtered_train.tsv \
-c '3' -m 'bert-base-uncased' -t 'bert' -e 'sentence' -g '0'
```

Model training is performed in **Step 5** by running run the script *5_TrainNNModel.sh* with the following parameters:

- -t *(train data file)*: JSON data file for classifier training (with embeddings)
- -d *(dev data file)*: JSON data file for classifier validation (with embeddings)
- -f *(field)*: Classify by field
- -e *(embedding unit)*: Embedding unit: 'sentence', 'word', or '-' (if not using pre-trained embeddings)
- -m *(model directory)*: Directory where to save the trained model
- -g *(gpu use)*: Index of the GPU to use (from 0 to available GPUs-1), -1 if only CPU should be used (default is -1)

Example:

```
5_TrainNNModel.sh \
-t ./data/datasets/amazonreview_train_filtered_train_bert-base-cased.json \
-d ./data/datasets/amazonreview_train_filtered_dev_bert-base-cased.json \
-f 'class' -e 'sentence' \
-m ./models/classical/amazonreview_train_bert-cased -g '0'
```

Model training is performed in **Step 6** by running run the script *6_ClassifyWithNNModel.sh* with the following parameters:

- -i *(input file)*: JSON data file for classifier testing (with embeddings)
- -o *(output file)*: Result file with predicted classes
- -e *(embedding unit)*: Embedding unit: 'sentence', 'word', or '-' (if not using pre-trained embeddings)
- -m *(model directory)*: Directory of the pre-trained classifier model
- -g *(gpu use)*: Index of GPU to use (from 0 to available GPUs-2), -1 if CPU should be used (default is -1)

8.2. Parameters and hyper-parameters

Below we detail the different parameters and hyper-parameters associated with each of the models, as well as the values that these were assigned in the experiments that we ran. Parameters with no command line tag are fixed in the code to the stated value, and each seed corresponds to a set of n runs, where $n =$ total number of runs of the experiment $\div$ number of seeds for the experiment.

Parameter Name	Command Line Tag	Values for Exp 1
Seed	s	13, 137, 541
Optimiser	p	COBYLA
Iterations of the optimiser	i	150
Runs of the model	r	30

***Table 14**: Pre-Alpha 1 parameters for experiment 1*

Parameter Name	Command Line Tag	Values for Exp 1	Values for Exp 2
Seed	s	13, 137, 541	29, 73
Optimiser	p	Adam	Adam
Iterations of the optimiser	i	150	150
Runs of the model	r	30	10
Batch size	b	512	2048
Ansatz	a	IQP	IQP
Qubits per Noun Type	n	1	1
Qubits per Sentence Type	-	1	1
Single qubit parameters	x	3	3
Circuit Layers	n	1	1

***Table 15**: Pre-Alpha 2 parameters for experiments 1 and 2*

Parameter Name	Command Line Tag	Values for Exp 1	Values for Exp 2
Seed	s	13, 137, 541	29, 73
Optimiser	-	Adam	Adam
Iterations of the optimiser	i	150	150
Runs of the model	r	30	10
Ansatz	a	IQP	IQP
Qubits per Noun Type	n	1	1
Qubits per Sentence Type	e	1	1
Single qubit parameters	x	3	3
Circuit Layers	n	1	1
Batch size	b	512	2048
Learning rate	l	0.2	0.2
Weight Decay	w	0	0
Step size for the learning rate scheduler	z	150	150
Gamma for the learning rate scheduler	g	1	1
Reduced dimension for the word embeddings	c	22	22
Version (choice of Alpha 1 or 2)	y	1*	1*

***Table 16**: Alpha 1 parameters for experiments 1 and 2. * Note that to choose Alpha 1 as the desired version, one must type alpha_pennylane_lambeq_original in the command line*

Parameter Name	Command Line Tag	Values for Exp 1	Values for Exp 2
Seed	s	13, 137, 541	29, 73
Optimiser	-	Adam	Adam
Iterations of the optimiser	i	150	150
Runs of the model	r	30	10
Ansatz	a	IQP	IQP
Qubits per Noun Type	n	1	1
Qubits per Sentence Type	e	1	1
Single qubit parameters	x	3	3
Circuit Layers	n	1	1
Batch size	b	512	2048
Learning rate	l	0.2	0.2
Weight Decay	w	0	0
Step size for the learning rate scheduler	z	150	150
Gamma for the learning rate scheduler	g	1	1
Reduced dimension for the word embeddings	c	22	22
Version (choice of Alpha 1 or 2)	y	2*	2*

***Table 17**: Alpha 2 parameters for experiments 1 and 2. * Note that to choose Alpha 2 as the desired version, one must type alpha_pennylane_lambeq in the command line*

Parameter Name	Command Line Tag	Values for Exp 1	Values for Exp 2
Seed	s	13, 137, 541	29, 73
Optimiser	-	Adam	Adam
Iterations of the optimiser	i	150	150
Runs of the model	r	30	10
Batch size	b	512	2048
Number of qubits in the circuit	u	3	3
Initial spread of the parameters	d	0.01	0.01
Learning rate	l	0.002	0.002
Weight Decay	w	0	0
Step size for the learning rate scheduler	z	150	150
Gamma for the learning rate scheduler	g	1	1

***Table 18**: Alpha 3 parameters for experiments 1 and 2*

Parameter Name	Command Line Tag	Values for Exp 3
Seed	s	61
Optimiser	-	Adam
Iterations of the optimiser	i	150
Runs of the model	r	10
Batch size	b	2048
Number of qubits in the circuit	u	3
Initial spread of the parameters	d	0.01
Learning rate	l	0.002
Weight Decay	w	0
Step size for the learning rate scheduler	z	150
Gamma for the learning rate scheduler	g	1

***Table 19**: Alpha 3 parameters for experiment 3*

8.3. Sentence grammar and parameterised quantum circuits

While we do build alternative NLP models in this project, which will be discussed later, the main method we employ is the already discussed DisCoCat framework. This framework aims to encode the meaning of sentences into a so-called string diagram. These string diagrams can then be mapped onto a parameterised quantum circuit using an Ansatz. The resulting PQC has a structure that corresponds to the grammar of the sentence, and the variable and hence trainable parameters correspond to the words that make up the sentence. This allows sentences with the same grammatical structure to be distinguished according to the different words they may contain.

8.3.1. The Lambeq pipeline

The lambeq software package, gives an easy to use pipeline for mapping sentences onto parameterised quantum circuits. Lets give an example here with the sentence *'John walks in the park'*.

Firstly we input the sentence into a parser, in this case the BobCat parser from lambeq. We can then generate a string diagram via the DisCoCat framework using lambeq. This is shown in fig.34.

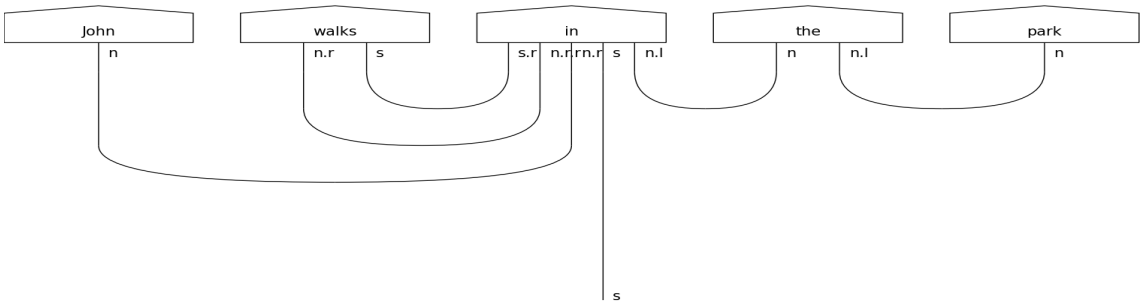

***Figure 34**: DisCoCat string diagram*

A diagram rewriting process is then implemented, which serves to reduce the complexity and therefore hardware resource requirements of the model. The diagram is then normalised. The resulting string diagram is shown in fig.35.

To convert the resulting string diagram to a quantum circuit we use the IQP Ansatz. This requires specification of Ansatz parameters which determine the complexity of the resulting circuit. For example, the number of qubits for atomic types must be specified. The resulting circuit is in DisCoPy (de Felice et al., 2021) form, which is another Python package, which is shown in fig.36.

This DisCoPy circuit can then be converted to various other quantum computing frameworks such as Pytket, Qiskit, and Pennylane.

Lambeq also enables model training by implementing this pipeline for an array of sentences in a class-based structure. This can be combined with other quantum computing frameworks and classical machine learning libraries in the design of new models.

8.3.2. Classical KNN experiment

To better understand the legitimacy of the Beta model and compare it to its classical counterpart, we decided to build a very simple KNN model. The goal here is to have a pipeline as similar as the Beta model. Thus,

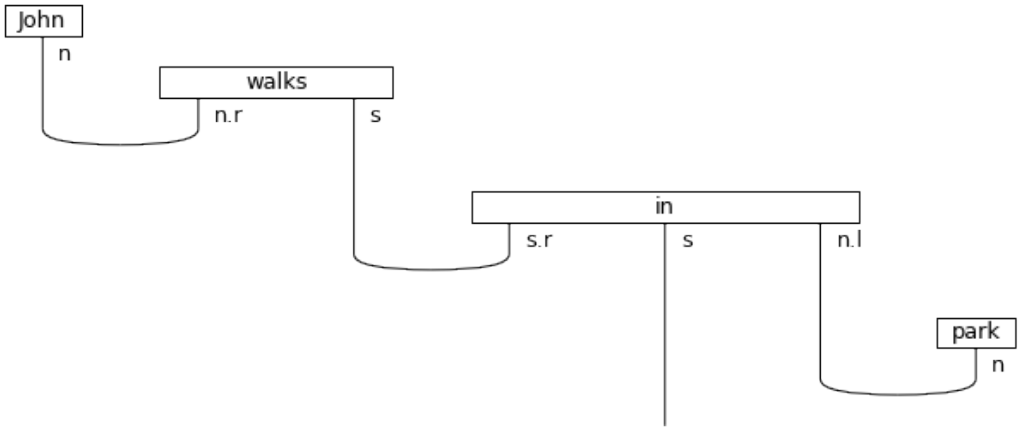

***Figure 35**: DisCoCat rewritten string diagram*

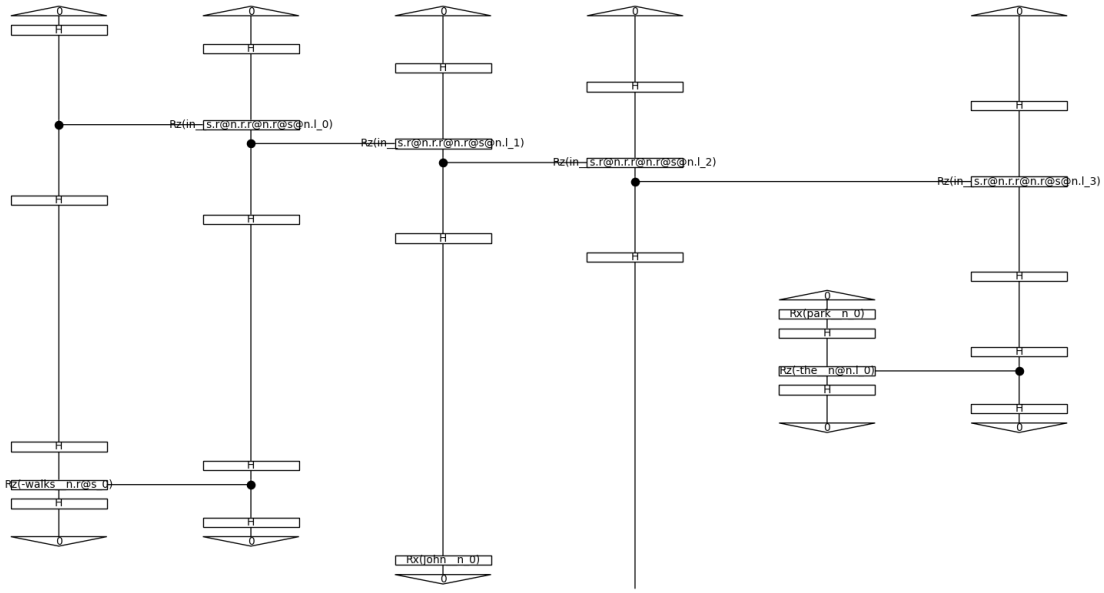

***Figure 36**: DisCoCat quantum circuit*

we have BERT sentence embeddings as input, followed by PCA dimensionality reduction, and finally, a KNN model.

In this experiment, we vary the number of PCA components between 2 and 512. By doing so, we aim to better understand at which point the PCA dimensionality reduction causes our vectors to lose too much information. We also vary the value of k between 1 and 9. Moreover, this experiment has been conducted with datasets A and B to assess if the dataset size has an impact on KNN classification accuracy.

As we can see in tab.20, the larger the number of PCA components, the higher the model accuracy. For n_components = 2 (the same as the one used in Experiment 4), we observe a test accuracy ranging between 0.58 and 0.63 for dataset A and between 0.54 and 0.57 for dataset B. Thus, for a low number of PCA components, the classical KNN performs poorly in binary classification tasks.

PCA	Classical Knn									
	dataset A					dataset B				
	k=1	k=3	k=5	k=7	k=9	k=1	k=3	k=5	k=7	k=9
n_components=2	0.63	0.61	0.6	0.59	0.58	0.57	0.54	0.56	0.55	0.56
n_components=4	0.65	0.65	0.66	0.64	0.63	0.57	0.58	0.59	0.57	0.57
n_components=8	0.71	0.71	0.69	0.69	0.69	0.6	0.61	0.62	0.61	0.61
n_components=16	0.78	0.78	0.76	0.76	0.75	0.68	0.69	0.7	0.68	0.68
n_components=32	0.79	0.79	0.79	0.79	0.77	0.69	0.7	0.7	0.7	0.69
n_components=64	0.8	0.8	0.78	0.79	0.79	0.7	0.71	0.71	0.72	0.69
n_components=128	0.8	0.8	0.79	0.78	0.78	0.72	0.71	0.7	0.72	0.7
n_components=256	0.81	0.8	0.8	0.79	0.78	0.73	0.72	0.71	0.71	0.71
n_components=512	0.81	0.8	0.8	0.79	0.79	0.73	0.72	0.71	0.72	0.71

***Table 20**: Experiment results for classical KNNs*
dataset A&B, 1 run, CPU

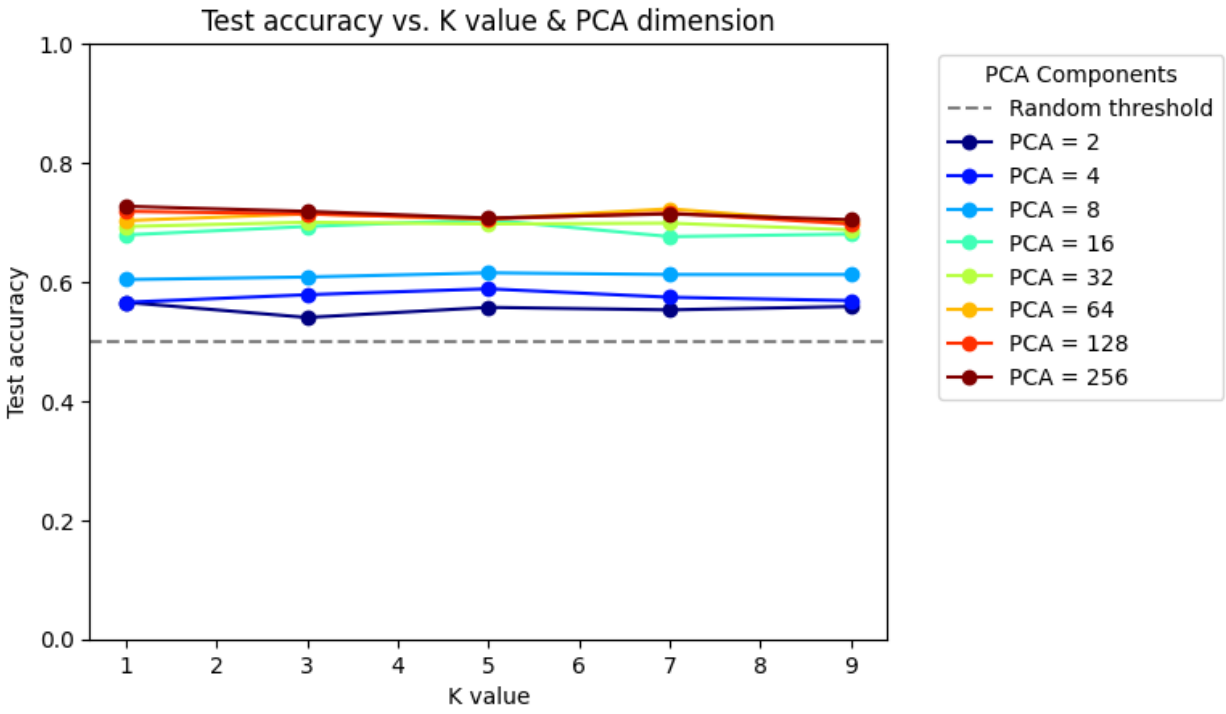

***Figure 37**: KNN experiment Dataset B*
dataset B, 1 run, CPU

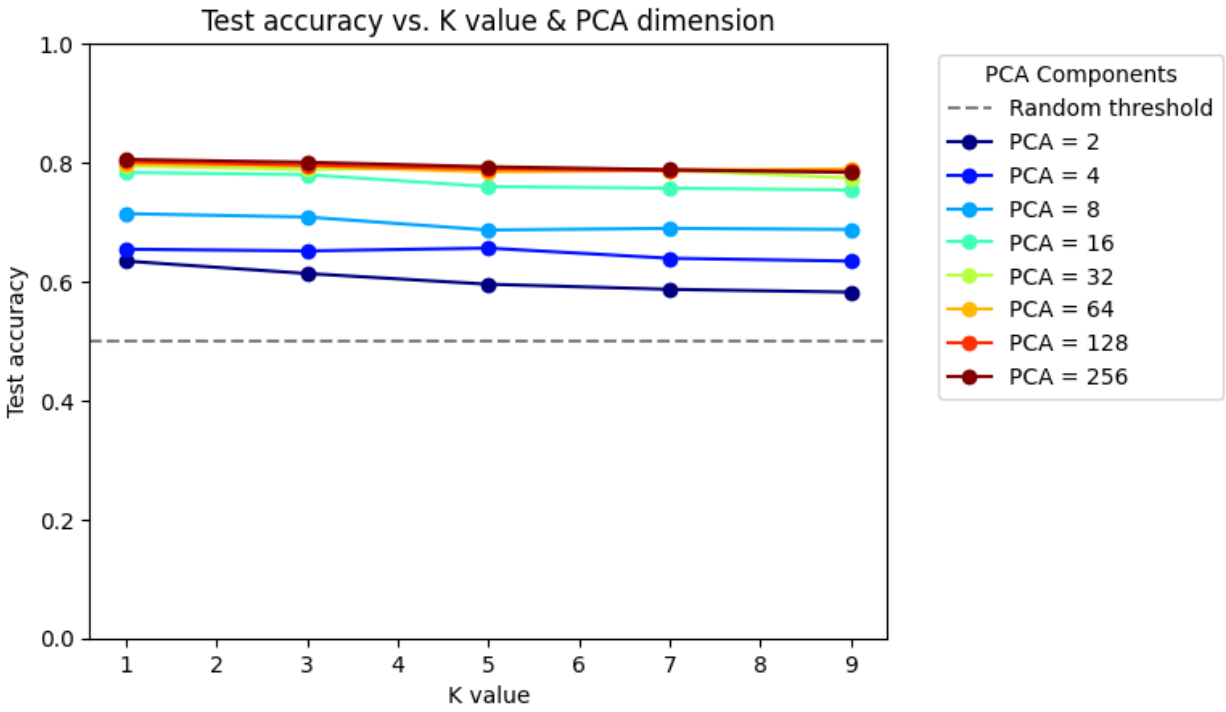

***Figure 38**: KNN experiment Dataset A*
dataset A, 1 run, CPU

List of Acronyms

Term	Definition
BCE	Binary cross entropy
BERT	Bidirectional Encoder Representations from Transformers
CLS	Classification
CPU	Central processing unit
D	Deliverable
GPU	Graphic processing unit
GHz	Gigahertz
GiB	Gibibyte
KNN	K-nearest neighbours
HPC	High-performance computing
IQP	Instantaneous quantum polynomial
NISQ	Noisy intermediate-scale quantum
NLP	Natural language processing
PCA	Principal component analysis
PQC	Parameterised quantum circuit
QC	Quantum computing
QNLP	Quantum natural language processing
WP	Work package

***Table 21**: Acronyms and Abbreviations*