1 2 9 0

UNIVERSIDADE D
COIMBRA

Paulo Miguel Teixeira Cortesão

Towards Generalisation in Tabular Models with LLM-Learned Concepts

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Professor Pedro Henriques da Cunha Abreu and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2024

1 2 9 0

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA

FACULDADE DE
CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE Ð
COIMBRA

Paulo Miguel Teixeira Cortesão

Towards Generalisation in Tabular Models with LLM-Learned Concepts

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Professor Pedro Henriques da Cunha Abreu and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Paulo Miguel Teixeira Cortesão

Contributos para a Generalização em Modelos Tabulares com Conceitos Aprendidos por LLMs

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes, orientada pelo Professor Pedro Henriques da Cunha Abreu e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2024

Abstract

Nowadays, human activities are taking advantage of the developments in Artificial Intelligence (AI) in its various fields. Typically, this is materialized by the use of data-driven approaches, where model parameters are learned by training with a dataset. However, this setup presents some issues, with the lack of generalisation being one of the most severe. With the development of Large Language Models (LLMs), trained on millions of texts across a variety of fields, real-world knowledge can be extracted and used to attenuate this issue.

This work addresses the generalisation issue by extracting relevant information from LLMs and incorporating it in the latent space of Machine Learning (ML) models. To reach this goal, we retrieved formulas from LLMs and applied contrastive and multitask learning to make ML models sensitive to those formulas. We conducted further experiments to evaluate the possibility of increasing the quality of the formulas provided by LLMs both iteratively and by finetuning. After applying these methods across 12 tabular classification datasets, we concluded that the proposed approach increases generalisation in comparison to the standard Multi-Layer Perceptron (MLP), with LLM knowledge producing an impact, especially in cases where there is concept drift. Experiments with iteration and fine-tuning processes revealed that iteration can produce improvements in some cases and that a dedicated finetuning process increases the quality of the retrieved formulas, showing gains in the proposed methods.

Keywords

Machine Learning, Large Language Models, Generalisation, Classifier Performance

Resumo

Hoje em dia, as atividades humanas tiram partido dos desenvolvimentos na área da inteligência artificial nos seus vários ramos. Tipicamente, isto é feito com abordagens baseadas em dados, em que os parâmetros associados a um modelo são aprendidos pelo treino com um *dataset*. No entanto, este paradigma apresenta alguns reveses, sendo um dos mais severos a falta de generalização. Com o desenvolvimento de LLMs (Grandes Modelos de Linguagem – Large Language Models), treinados com milhões de textos a cobrir várias áreas do saber, é possível extrair conhecimento do mundo real para atenuar os efeitos deste problema.

O objetivo do presente trabalho é melhorar a generalização de modelos de Machine Learning (ML) extraindo informação relevante de LLMs e incorporando-a no espaço latente destes. Para alcançar este objetivo, obtiveram-se fórmulas a partir de LLMs e aplicaram-se técnicas de aprendizagem multi-tarefa e por contraste para tornar os modelos de ML sensíveis a estas fórmulas. No mesmo sentido, realizaram-se experiências para a avaliar a possibilidade de melhorar a qualidade das fórmulas obtidas tanto através de iteração como de *finetuning*. Depois da aplicação destes métodos em 12 *datasets* tabulares de classificação, concluiu-se que a abordagem proposta melhora a generalização em comparação com o MLP, com o conhecimento obtido a partir das LLMs a produzir um impacto positivo especialmente em casos com *concept drift*. As experiências com processos de iteração e de *finetuning* revelaram que a iteração produz melhorias em alguns casos, e que um processo de *finetuning* dedicado melhora a qualidade das fórmulas extraídas, demonstrando ganhos nos métodos propostos.

Palavras-Chave

Aprendizagem Computacional, Grandes Modelos de Linguagem (LLMs), Generalização, Desempenho de Classificadores

Agradecimentos

Nesta pequena página, gostaria de agradecer ao meu orientador, o Professor Pedro Abreu, pela disponibilidade que sempre demonstrou ao longo do trabalho para tirar dúvidas, discutir ideias e puxar ao máximo o meu potencial, depositando uma confiança inabalável nas minhas capacidades. Espero continuar a embrenhar-me no mundo da investigação científica com todos estes ensinamentos! Também não poderia deixar de agradecer à Professora Mihaela van der Schaar por me ter recebido em Cambridge e ao Professor Fergus Imrie pelas discussões ricas que moldaram o decurso do trabalho de investigação aqui desenvolvido.

Do mesmo modo, quero deixar uma enorme palavra de gratidão à minha família: à minha mãe, ao meu pai e à minha irmã, pela companhia que foram, tanto cá como lá fora, e por serem o meu porto seguro. Este agradecimento estende-se às minhas avós, pelo apoio e compreensão ao longo destes anos de estudo; ao meu avô, a título póstumo, por me ter incutido valores que têm perdurado até hoje; e ao meu tio, que sempre me motivou para a tecnologia e para a ciência.

Os amigos que fiz neste percurso que já conta com 5 anos merecem um especial obrigado por todos os momentos que partilhámos. Estas vivências enriqueceram-me e deram-me força nos momentos mais difíceis, permitindo-me levar este projeto a bom porto. Por isso, espero poder continuar a viver convosco os anos que se avizinham, mesmo depois do início da vida profissional!

Passando para o plano musical, queria dar uma calorosa saudação aos Perpetuum, que me mostraram um estilo de música diferente e me ajudaram a viver Coimbra de alma e coração, aquém e além-fronteiras! Faço votos para que continuemos a aquecer corações e a levar a Canção de Coimbra aos quatro cantos do mundo!

Por fim, um sonoro FRA para a Quantunna, que me viu crescer desde a minha entrada definitiva em 2022, e fomentou em mim capacidades que nunca seria capaz de prever, mas que se revelaram indispensáveis. Todas as atividades que desenvolvemos têm sido para mim um ótimo escape da pressão da vida académica; e a música que criamos e partilhamos cria um espírito de união que espero levar para a vida!

A todos um sentido obrigado!

Contents

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

BERT Bidirectional Encoded Representations from Transformer.

BPE Byte-Pair Encoding.

CNN Convolutional Neural Network.

CoT Chain of Thoughts.

CPU Central Processing Unit.

DL Deep Learning.

DPO Direct Preference Optimization.

GPT Generative Pre-Trained Transformer.

GPU Graphical Processing Unit.

GQA Grouped Query Attention.

LLM Large Language Model.

LoRA Low-Rank Adaptation.

LSTM Long Short-Term Memory.

MHA Multi-Head Attention.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MQA Multi-Query Attention.

NLP Natural Language Processing.

ORPO Odds Ratio Preference Optimization.

PEFT Parameter Efficient Finetuning.

QLoRA Quantized Low-Rank Adaptation.

RLHF Reinforcement Learning from Human Feedback.

RNN Recurrent Neural Network.

TURL Table Understanding through Representation Learning.

List of Figures

List of Tables

Chapter 1

Introduction

Since the Dartmouth workshop, which marked the official birth of the field of Artificial Intelligence (AI) in 1956 [Russell et al., 2022], there have been several efforts to aid human life with resort to it.

In fact, the applications of AI span across many human activities: identifying and classifying objects in images, predicting the outcome of economic cycles and pandemics, detecting anomalies in medical images, producing computer code, summarising text, and even generating images from prompts [Becker et al., 2023; Day and Chen, 2018; Peng et al., 2022; Ting et al., 2018].

An important subfield of AI that allows for this is Machine Learning (ML). Typically, in this context, two main tasks are addressed: supervised and unsupervised learning. In the former, the data is labeled, while in the latter, the data is unlabeled. In supervised learning, ML models learn from training data, which can lead to several issues that hinder their performance in deployment. One important issue to consider in this setting is generalisation: when a model goes from the simulation phase, where it is trained and tested, to the production phase, where it is applied in the real world, its accuracy can significantly decay.

As such, it becomes imperative to research alternative ways of improving the generalisation of ML models.

1.1 Motivation

Despite the development of AI to its current standards, generalisation is a constant concern in the development of any data mining pipeline. Various factors can lead to poor generalisation, which may be related to the training data or the training setup:

- The dataset might not represent its underlying context correctly. In this case, relevant features or feature values might be missing, which results in an incomplete characterisation of the context. The same can happen if some data groups are missing or underrepresented. With this missing information, the ML model struggles to generalise to certain regions of the context and performs poorly;
- Even though the dataset might accurately describe its context, the training setup can be built with flaws, such as dataset shift or an incorrect model complexity. In dataset shift, the train-test split can leave out relevant information from the training set if strategies such as class stratification are not applied. An incorrect complexity means that the model to train can be too complex, fitting its parameters to noise and overfitting the training data; or too simple, displaying poor performance as it cannot represent the context with enough detail, underfitting the data. ML common practices, such as cross-validation, stratification, hyperparameter tuning and regularisation help to minimise these problems, by ensuring a correct train-test split and fostering an adequate fitting of the data with a model of the right complexity.

Some strategies can be adopted to attenuate these problems, but the ideal scenario for the application of ML models would require an unlimited amount of data to represent every detail within the context, which implies that generalisation is an issue inherent to ML.

The field of Natural Language Processing (NLP), with its recent advancements, provides an alternative way to incorporate knowledge into ML models: the Large Language Model (LLM) – an ML model that is trained on massive amounts of text of various sources, which makes it knowledgeable regarding a great variety of fields. With the correct prompt engineering techniques, it is possible to extract its knowledge for use [Arora et al., 2023; Wei et al., 2022].

Considering the millions of texts these models were trained on, they can provide relevant insights that could be incorporated into the smaller ML models in their various domains. After feeding LLMs with a description of the training data, they can be asked to generate relevant concepts for use in model training.

1.2 Main Goals

The present work aims to increase the generalisation capability of ML models with a resort to the information present in LLMs. In order to achieve this, two main research questions were posed, defining two different directions for exploration:

1. Can we increase an ML model's generalisation by embedding the information extracted from LLMs into the model?
2. Can we improve the quality of the information provided by an LLM, training it for the information retrieval task that is taking place?

To answer Question 1, an experimental setup was created allowing for the extraction of information from LLMs and subsequent incorporation into the learning process of Multi-Layer Perceptrons (MLPs). The extracted information took the form of concepts, which are formulas derived from the features in a given dataset. These concepts were then incorporated into the training process of MLPs using various combinations of contrastive and multi-task learning techniques. Generalisation was measured by determining model AUROC on the test set across 12 tabular classification datasets, using a stratified split, using a feature-based split and using only a portion of the training set for training. More details on this experimental setup are presented in Section 4.1.

The results, developed in Section 5.1, show a positive effect on classifier AUROC in various settings where generalisation could be tested, and demonstrate the importance of the knowledge embedded in LLMs, especially in cases where concept drift is present.

Question 2 was tackled by studying the impact of iteration and finetuning on the quality of the retrieved concepts. For this, after determining the impact of each of the concepts provided by an LLM on model performance, this information would be provided to the LLM as feedback to obtain better concepts in a subsequent iteration, and the process was repeated 4 times across 8 of the initial datasets. To understand the impact of finetuning, we determined the quality of some concepts and used this data for the compilation of a preference dataset. Following this, we finetuned *LLaMa3-8B-Instruct* on this dataset and extracted concepts from both the pretrained and finetuned model. Finally, we assessed the impact on model performance by comparing the test-time AUROC in the methods that used the concepts provided by both models. This experiment is further detailed in Section 4.2.

The outcomes of this experiment are discussed in Section 5.2. We conclude that an iterative process can be beneficial in some cases, and that finetuning further improves generalisation.

1.3 Document Structure

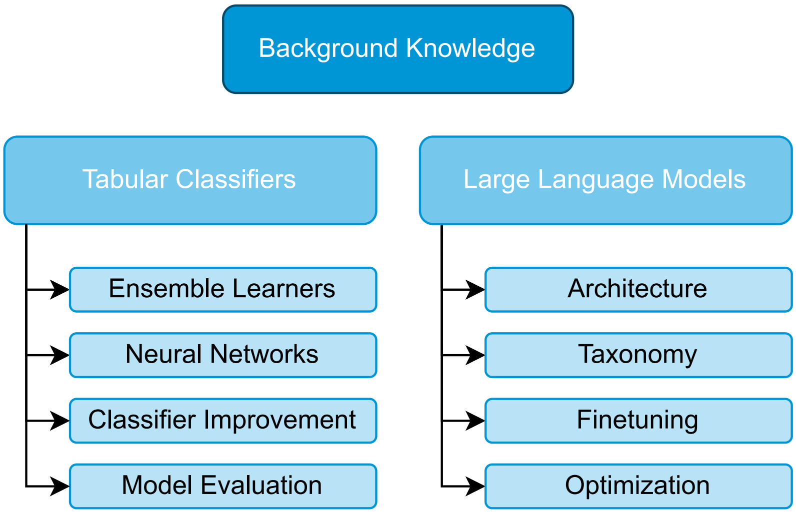
This document is structured as follows:

- Background knowledge will be presented in Chapter 2. Rudiments on tabular classifiers, including ensemble learners and neural networks, will be presented; and a discussion on LLMs will be conducted;
- A literature review will be provided in Chapter 3, encompassing the various works that aim to use LLMs as model teachers for various aims and common benchmarks for these models. Relevant conclusions about prompt engineering will also be mentioned, and a brief state of the art regarding LLMs and their performances will be included;
- A description of the experimental setups involved in the exploration of the research questions is provided in Chapter 4, including the used datasets and LLMs, proposed methods and gathered performance metrics;
- Results are provided and discussed in Chapter 5, showcasing baselines, graphs and tables that clarify the role of the various independent variables in the performance of the analysed models;

- The conclusions of the present work are provided in Chapter 6.

Chapter 2

Background Knowledge



For a good understanding of the present work, several important concepts shall be illustrated in this chapter. Tabular classifiers will be handled in Section 2.1, including ensemble learners, which will serve as baselines; and Artificial Neural Networks (ANNs), the main type of model in use. Other aspects such as the improvement of these models and common evaluation metrics will also be mentioned. Following this, in Section 2.2 we will discuss LLMs, presenting their architecture and taxonomy along with some finetuning and optimisation techniques applied to these models.

2.1 Tabular Classifiers

In the context of this work, tabular classifiers are ML models that handle tabular data, conducting classification tasks from it. Within this scope, ensemble classifiers will be used as baselines for comparison. They are described in Subsection 2.1.1. Neural networks can also work with tabular data, learning a suitable representation that captures the necessary details for the task at hand. These models shall be discussed in Subsection 2.1.2. Then, the possibility of LLM knowledge incorporation in these models shall be discussed in Subsection 2.1.3, motivating the choice of neural networks as the main object of attention in the present work. Finally, some evaluation metrics that allow to compare ML model performance will be discussed in Subsection 2.1.4.

2.1.1 Ensemble Learners

Ensemble learners are tabular classifiers that aim to achieve superior performance by combining the outputs of multiple smaller tabular classifiers.

The type of combination of outputs determines the category of ensemble learner [Géron, 2017]:

- **Voting classifiers** consist of voting systems where each different model casts a vote on a given class, and the resulting output is the class with the most votes. Their performance requires some level of statistical independence between the models, which is achieved by using different algorithms for predictions;
- **Bagging models** are also voting systems, but instead of applying various algorithms, they use models that are all of the same type but are trained on different subsets of the training data. Random forests are common examples of these models, where the base models are decision trees;
- **Boosting models** combine the outputs by implementing a chain where each model tries to correct the output issued by the previous one. This can be done by assigning a greater weight to mispredicted instances in the subsequent models, as is done by *AdaBoost*, or by fitting the subsequent models on the residual errors resulting from the previous models, which is what happens in *XGBoost* [Chen and Guestrin, 2016] and *LightGBM* [Ke et al., 2017].

These models are common choices for handling tabular data, so their perfor-

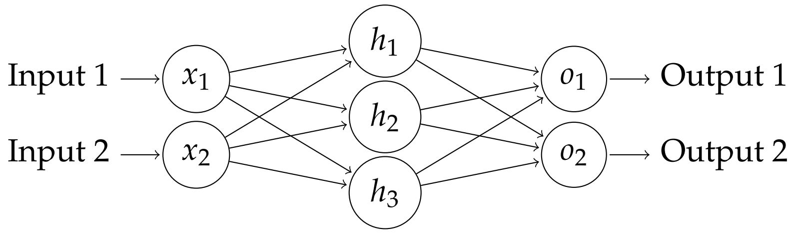


Figure 2.1: Neural Network Diagram

mance in the datasets considered in the experimental setup will serve as a baseline.

2.1.2 Artificial Neural Networks

In this section, a broad explanation of the functioning of neural networks will be provided, since they will be the main kind of tabular classifiers to work with throughout the present project. As such, their functioning will be presented, along with their training process and a summary of their applications.

Definition

ANNs are ML algorithms that aim to mimic the connections between neurons in the human brain. Because of this, they can be inserted in the connectionist view of AI – the mimicking of the human brain, as opposed to the symbolic view, based on pure logic, and the bio-inspired view, based on biological processes [Russell et al., 2022].

This human brain metaphor stems from the connections that exist among computing units, called neurons: in the human brain, neurons are connected among each other through synapses, where the axon terminals of a neuron connect to the dendrites of another. These cells work with electrical current flowing through them, and electrical stimulation of a neuron only occurs if the stimuli it gets in the dendrites from its synapses surpasses a given threshold potential.

Likewise, an ANN can be thought of as a process with several layers of neurons, where a neuron connects to neurons in the following layers; and computations can be made in parallel, just like in the human brain.

The circles in Figure 2.1 represent numerical values; the arrows represent a multiplication operation involving the values in the left tip of the arrow and a trainable scalar parameter called weight. The values in the right tip of the arrows are the result of the application of an activation function to the weighted sum of the values in the left tip of the arrow (in Figure 2.1, each arrow has a weight w associated with it). The activation function is usually non-linear, allowing the network to separate non-linearly separable data – for example, hyperbolic tangent, arc tangent, or ReLU: $ReLU(x) = max(0, x)$. In the case of classification problems, the final activation function is commonly softmax, which turns a vector of real numbers into a vector of values in $[0, 1]$ where each value can be seen as the probability that an instance belongs to a given class:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}} \tag{2.1}$$

where $\mathbf{z}$ is the n-element output vector representing a given instance before softmax activation.

The neural network structure presented can be replicated throughout several layers (instead of the three layers shown in the figure, it can have an arbitrary number of layers). There are also some specialized layers for image recognition (convolutional layers, which apply filters to the image, whose output is forwarded), and time series (recurrent connections where the output in a given instant influences the computations in later instants).

Unlike other ML methods, where the learning process is based on the representation of raw data or with very little processing (such as feature normalization, selection and reduction), in neural networks, the representation that leads to the classification or regression is learned by the network itself – representation learning. This more complex representation corresponds to the output vector of the second-to-last layer, as the linear layer that follows it learns the weight to assign each of the values of the vector for determining each class. It is said that this is the latent space of the neural network.

The training process for these models is data-driven. This means that we need specific inputs and their respective expected outputs to be able to teach the network to classify a given category or determine a given output value. The update of the various parameters that constitute the network are updated according to an elementary calculus concept: the minimisation of a loss function that represents the error produced by the network.

Loss functions Depending on the task to perform, certain loss functions are more applicable. For example, for classification, it is common to use cross-entropy – in this case, given n classes, the output is a vector of length n of values in the interval $[0, 1]$ that sum to one (class probabilities) where the determined class is the one in the position of the maximum output value. The loss is given by -log y_{target}, so it measures the uncertainty associated with the target class (if this probability is close to zero, the loss is very large; if it is close to 1, the loss is near 0).

Likewise, for reconstruction problems (such as compressing images in a compact vector, so they can be adequately decompressed with little error), the loss to minimise is the sum of squared differences between the pixels in the input image and the decompressed image. Regression problems can apply the same loss function.

Gradient Descent In order to train the network, the gradient descent algorithm aims to minimise these losses for the training examples. For a given number of epochs, the whole training dataset (D_{train}) passes by the network (*net*) in batches and, for each batch, the weights are updated using the chain rule. A learning rate ν governs the magnitude of the update and can be scheduled to gradually decrease as training proceeds.

Algorithm 1 Stochastic Gradient Descent in Neural Networks

Input: ν, D_{train}, *epochs*, *net*, *batch_size*
Output: Trained network *net*
  $no\_epochs \leftarrow 0$
  while $no\_epochs < epochs$ **do**
    for random batch $(inputs, targets)$ of size *batch_size* in D_{train} **do**
      $outputs \leftarrow net(inputs)$
      $loss \leftarrow loss_{func}(targets, outputs)$
      for *weight* `in` *net* **do**
        $weight \leftarrow weight - \nu \times \frac{\partial loss_{func}(outputs)}{\partial weight}$
      end for
    end for
    $no\_epochs \leftarrow no\_epochs + 1$
  end while

According to Algorithm 1, the update is governed by the partial derivative of the loss function with respect to each weight; and the weight is updated in the opposite direction, thus countering the growth of the loss function (gradient descent).

Using symbolic calculus for determining each partial derivative from scratch would lead to an impractical time complexity; however, considering the chain rule, it is possible to divide a gradient calculation among multiple isolated terms; and these terms repeat themselves – the derivative of the error with respect to a given weight only depends on the derivative of terms in the following layer. Because of this, the results of derivative calculations can be stored, speeding up training in a process called backpropagation, where these partial derivatives are calculated in a direction contrary to the inference process: starting in the output layer and ending in the input layer, with the values calculated in the layers to the front being used to aid derivative calculation in the layers to the back.

As well as this, the determination of the output of a given layer can be defined as in Equation 2.2.

$$y_{i+1} = f(\mathbf{W}y_i) \tag{2.2}$$

The representation of the forward pass in this way makes it clear that matrix multiplications are present, and they can be parallelized using Graphical Processing Units (GPUs) to speed up both the inference process and the update rule for backpropagation.

Neural Network Applications

Throughout time, the development of neural networks has expanded their applicability to various computational tasks. Novel architectures have been created, allowing for the processing of tabular data, images, time series and text. The text below aims to summarise some relevant ANN architectures and respective applications.

Multi-Layer Perceptrons MLPs [Ivakhnenko and Lapa, 1967] are simple neural networks where every layer is fully-connected, i.e., every output element of one layer is connected to every neuron of the folowing layer, as described in Figure 2.1. These networks are feed-forward, meaning that the output of a layer is only fed to layers after it. Despite the simplicity of these networks, the Universal Approximation Theorem applies to them: given a hidden layer with enough neurons and nonlinear activation functions, an MLP can approximate any function with any given precision [Hornik et al., 1989]. As such, these neural networks can be used as classifiers of tabular data, since they can nonlinearly map an input with f features to an output with o elements. Considering this, MLPs are the main focus of the present work.

Convolutional Neural Networks CNNs are specialized architectures that are primarily used for image data. For this, they possess layers where the input is convolved with various kernels. This allows for the extraction of patterns in a shift-invariant fashion, which is useful for classification purposes. Usually, the architecture of CNNs consists of several convolutional layers with downsampling, allowing for the extraction of patterns in progressively larger scales; followed by fully connected layers where feature vectors are extracted to handle the end task. This architecture also allows for backpropagation and GPU optimization [Chellapilla et al., 2006; Cireşan et al., 2010].

Recurrent Neural Networks and Long Short-Term Networks RNNs are neural networks that handle time series [Elman, 1990]. For this, they display recurrent connections, where the output of layers to the front is an input for layers to the back. Because of this architecture, they model the relationship between inputs of different moments in time. Still, this modelling is limited to the recurrence depth, the maximum number of steps behind considered at a given time. To tackle this, the LSTM [Hochreiter and Schmidhuber, 1997] was created: in this architecture, in every step, there is a context vector retaining the relevant information from past steps. Both of these architectures are trained by backpropagation through time, where the output from more recent steps is also backpropagated to the layers in the calculation of the previous steps.

Neural Networks for Tabular Data Efforts have been made to improve the way neural networks consider interactions among features in various abstraction levels. This was achieved with the invention of new kinds of layers, or by taking advantage of the attention mechanisms that had been developed [Vaswani et al., 2017]. For instance, DeepFM [Guo et al., 2017] uses both Deep Learning (DL) and factorisation machines to extract high- and low-level feature interactions from a dataset. High-level interactions come from the DL process; whereas low-level interactions are calculated in factorisation machines, using dot-products between the representations of sparse features in the latent space of early layers of the same neural network – a similar process occurs in the transformer architecture behind LLMs. The resulting vectors from both components are then summed and passed by a sigmoid for binary classification purposes.

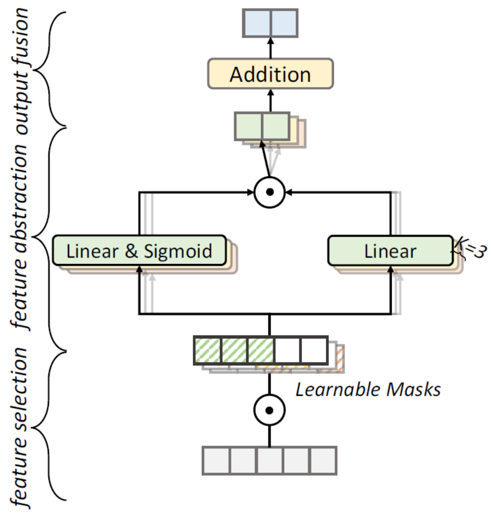


Figure 2.2: DANet architecture, proposed in [Chen et al., 2022]: On the left, the architecture of an Abstract Layer; on the right, a DANet basic block

Another example is the abstract layer [Chen et al., 2022]: a layer that applies attention-based feature selection aiming to group features into meaningful abstract groups. These groups are then used for the calculation of higher-level features, which are finally fused into the layer's output. This block can be stacked, with each layer always receiving the input features along with the outputs produced by the previous layer, forming a DANet architecture. With 20 stacked abstract layers, the performance achieved by this ANN was similar to or better than the baseline ensemble learners, across the various classification and regression tasks used as benchmarks.

2.1.3 Tabular Classifier Improvement

Regarding tabular classifiers, several methods could be developed to incorporate more relevant information into these models. Assuming the extra information to include is numeric, the following strategies can be applied:

- The extra information could be included in the training phase as an extra feature. This has the advantage of applying to all the tabular classifiers discussed. However, in the deployment phase, this numeric information would have to be calculated for all the instances being tested, which could be too complex;

- The classifier could be used to determine both the extra information and its end task output, gaining sensitivity to this information – multitask learning. This would only be of use if the part of the model that determines the output for the end task is influenced by the part that determines the extra information. Considering the functioning of ensemble learners, this is not the case. It is only possible in neural networks, where all the outputs influence the latent space, which in turn impacts the end task output;
- The classifier could be made sensitive to discretized information by the application of contrastive learning: if the extra information could be divided into classes, the model could distinguish instances corresponding to equal and different classes by making its intermediate representations of instances of the same class close to each other while pulling instances of different classes apart. This method is also limited to neural networks since it depends on the existence of a latent space.

Bearing in mind the various alternatives for classifier improvement, a brief introduction on multitask and contrastive learning is provided below.

Multitask Learning

Multitask Learning is a mechanism where various tasks are learned in parallel based on the same machine representation of data. For example, in the case of neural networks, there are outputs associated with each task, and the network's latent space is used to determine all the outputs. This procedure has been applied to NLP [Liu et al., 2019], as well as reinforcement learning and in medical diagnosis of pneumonia [Caruana, 1997].

This method is based on the assumption that an inductive bias produced by training on a task can affect the performance on other correlated tasks, since the trained representation is common. Several reasons can be responsible for this. On one hand, as there can be unrelated tasks, simultaneous training adds noise to both, which can be beneficial for the generalisation of the network. On the other hand, the information in the extra tasks is also embedded in the learning algorithm, leading to an inductive bias that it would otherwise not have and that can positively impact performance.

Contrastive Learning

Contrastive learning is another method affecting the latent space. It is a form of self-supervised learning, in the sense that it does not explicitly require class labels: the only goal is to bring instances of the same class closer to each other in the latent space, while distancing instances of different classes from each other. Because of this, knowing if two instances belong to the same class is enough for the application of the algorithm.

Contrastive learning is achieved by the application of a loss function to the latent representations of a given number of instances. If a contrastive loss is used, only two instances x_i, x_j are considered at a time [Le-Khac et al., 2020], and the loss aims to bring them closer if they are of the same class ($y_i = y_j$), and vice-versa (Equation 2.3):

$$L_{contrastive}(z_i, z_j) = \begin{cases} ||z_i - z_j|| & y_i = y_j \\ -||z_i - z_j|| & \text{otherwise} \end{cases} \tag{2.3}$$

If a triplet loss is used, three instances are considered at a time: one anchor x_a, one positive x_+ and one negative x_-, such that $y_a = y_+$ and $y_a \neq y_-$. In this case, the loss function aims to simultaneously bring the latent representation of x_a closer to the one of x_+ and further away from the one of x_-, while accounting for a margin $m > 0$, the minimum difference between the distances from the anchor to the positive and the negative instances (Equation 2.4):

$$L_{triplet}(z_a, z_+, z_-) = max(||z_a - z_+|| - ||z_a - z_-|| + m, 0) \tag{2.4}$$

Contrastive learning has been applied to fields such as NLP, for sentiment analysis [Luo et al., 2022], computer vision tasks [Chen et al., 2020], and in few-shot learning contexts [Gidaris et al., 2019].

2.1.4 Model Evaluation

Given that tabular classifier generalisation will be evaluated in the present work, it is necessary to define the metrics by which this evaluation will be conducted. To this aim, we will define the notation and provide some metrics that allow us to evaluate classification tasks.

Notation

The notation to be used throughout this section shall be the following: given a dataset $\mathcal{D} = (\mathcal{X}, y)$ with n instances, with feature matrix $\mathcal{X}$ and targets y, the model will produce an output $o = f(\mathcal{X})$, one for each instance ($o_i = f(x_i)$). For classification tasks, where there are C classes that the model will choose from, counts such as false positives (FP), true positives (TP), false negatives (FN), and true negatives (TN) are determined with respect to each class (in the case that the problem is multiclass; otherwise these counts are determined regarding the positive class). The overall number of correct predictions is defined as TC. Mathematically, these counts can be defined in the following way:

- $FP_c = \#(\{x_i \in \mathcal{X} : y_i \neq c \wedge o_i = c\})$
- $TP_c = \#(\{x_i \in \mathcal{X} : y_i = c \wedge o_i = c\})$
- $FN_c = \#(\{x_i \in \mathcal{X} : y_i = c \wedge o_i \neq c\})$
- $TN_c = \#(\{x_i \in \mathcal{X} : y_i \neq c \wedge o_i \neq c\})$
- $TC = \#(\{x_i \in \mathcal{X} : y_i = o_i\})$

Metrics for classification tasks

Common metrics that evaluate model performance in classification tasks are the following:

- Accuracy, the ratio of correct predictions:

$$acc = \frac{TC}{n} \tag{2.5}$$

- Recall, also named true positive rate (TPR), is the ratio between correct positive predictions and total positives, giving an estimate of the power of the classifier with respect to a given class:

$$recall_c = \frac{TP_c}{TP_c + FN_c} \tag{2.6}$$

The counterpart of this metric is the false positive rate:

$$FPR_c = \frac{FP_c}{FP_c + TN_c} \tag{2.7}$$

- Precision, also named positive predictive value (PPV), is the ratio between correct positive predictions and total positive predictions, constituting an estimate of how much a positive prediction should be trusted:

$$precision_c = \frac{TP_c}{TP_c + FP_c} \tag{2.8}$$

- F-β score, a combination of both precision and recall that aims to capture the performance of the classifier with respect to a given class as a whole. β can be adjusted to give more or less importance to recall over precision. Commonly, equal importance is considered with $\beta = 1$, yielding a harmonic mean between them:

$$F_{\beta,c} = (1 + \beta^2) \cdot \frac{precision_c \cdot recall_c}{\beta^2 \cdot precision_c + recall_c} \tag{2.9}$$

- AUROC (Area Under Receiving Operator Characteristic curve) score – in binary classification, the distinction between classes can be performed based on an estimate between 0 (negative class) and 1 (positive class). As such, the boundary between classes can be cut in any number between 0 and 1. Depending on the boundary defined, the precision and recall are different. The graph that results from the variation of the boundary, mapping the true positive rate (recall) to the respective false positive rate in each boundary value between 0 and 1, can be used to evaluate the quality of the classifier: the close the area under this curve is to 1, the better. An AUC of 0.5 corresponds to random chance; and an AUC under 0.5 reflects a classifier that is worse than random (see Figure 2.3). For multiclass problems, this metric is obtained with two different strategies: one-vs-one, where only the instances of two classes are confronted; and then the average AUC over all pairs of classes is determined; or one-vs-all, which considers a given class as positive and all the others as negative, with the average AUC over all classes being determined.

For multiclass problems ($C > 2$), the class-related metrics are averaged for an

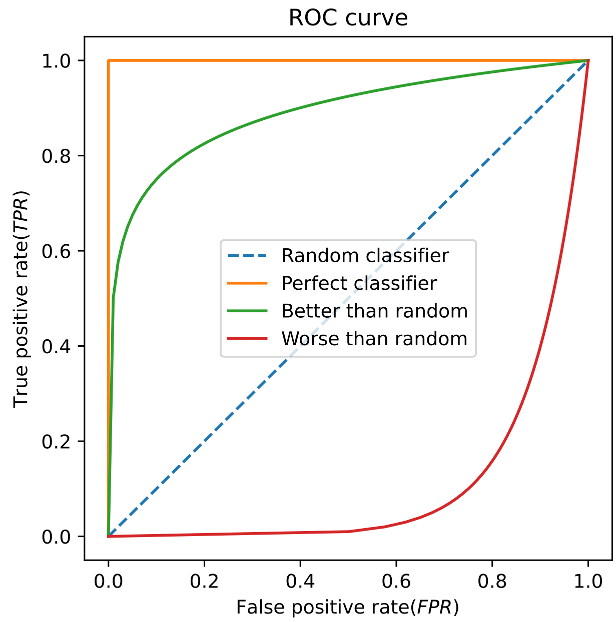


Figure 2.3: AUC examples

overall evaluation of the classifier. This average is normally determined by making each instance count equally (micro-average – takes into account class prevalence throughout the dataset) or making each class count equally (macro-average).

2.2 Large Language Models

Large Language Models are ANN models that deal with natural language. Their architecture allows them to understand text and perform tasks of various natures with it, for example text generation, sentiment classification and named entity recognition [Wang et al., 2022]. The arrangement of their basic building blocks is also adapted to their end task, and optimisation strategies can be applied to enhance the quantity of text that they can process at a time. In this section, the basic building blocks that constitute LLMs are described, as well as the categories of LLMs that are most suitable for each task, finetuning processes to train them in specific tasks and common strategies that give them faster inference and a more economical finetuning process.

2.2.1 Architecture

The architecture behind LLMs is comprised of several basic blocks. Some of them aim to represent the information from the text in a way that a computer can understand it (tokenization, embedding layers), whereas others extract the semantic connections among the various tokens present in the text (self-attention layers). The basic blocks that make up an LLM are the following:

- **Tokenisation** – When a given input is given to an LLM, the text is divided into tokens for processing. This means that the LLM sees the text as a sequence of tokens. The relationship among them will determine the meaning perceived by the LLM. One possible way of determining the list of the tokens to include is using Byte-Pair Encoding (BPE): starting with the letters of the alphabet and punctuation as tokens; and given a large corpus of text, a count of the consecutive pairs of tokens in the text is made; and the most frequent pair of tokens are concatenated and added to the list. This process is repeated until the final number of tokens k is reached. This number k determines the size of the vocabulary that the language model can handle; and the input is fed to the language model as a series of one-hot encoded vectors of size k, where the element set to 1 represents the token in each position.

- **Token Embedding Layer** – Each token has its own embedding: a representation in a real-number vector of a given dimension d. One possible way of determining word embeddings is by using a logistic regression algorithm known as skip-gram [Mikolov et al., 2013]: given a large text, a specific token window is specified (e.g. 5 tokens). As that window slides through the text, the embedding of the token in the center is trained: the neighbour tokens in the window are positive examples, and then some randomly selected tokens serve as negative examples. The training process of the embedding will adjust the linear regression weights so that the embedding of the token is close to the embeddings of its neighbors, and far from the embeddings of the randomly selected negative examples, according to a given distance metric (for example, the dot product of two embeddings). The underlying assumption of the algorithm is that words that are found in the same neighborhood have similar meanings, and vice-versa.

- **Positional Embedding Layer** – When considering a sentence/text as a whole, each word/token can have different meanings depending on the position where it appears. As such, it is necessary to embed the positional information in the token as well. For this, a positional embedding vector is added to the token embedding before it is fed to the rest of the network. The original article where positional embedding was described proposed a weighted sum of sines and cosines of different amplitudes and phases for each element of the positional embedding vector [Vaswani et al., 2017]. However, there are other possible positional embedding functions, including learnable ones.

- **(Multi-Head) Attention Layer** – This layer correlates the various tokens fed to the model among themselves, so it could be considered a semantic analyser. As the algorithm is only working with embeddings, the operations to conduct have to be mathematical. Considering a one-directional layer (where the tokens only depend on the ones before), for each token, this layer will output a weighted sum of the inputs seen so far, where the weights represent the relevance of each input for that moment. In this case, the embeddings x_i are used as inputs, but they undergo matrix multiplications that allow different representations: one as the current focus of attention (query, q_i); another as a preceding token (key, k_i); and another one as a value in the weighted sum (value, v_i). For each of these representations, there is a trainable matrix allowing for each representation: W^Q, W^K and W^V: $q_i = W^Q x_i$, $k_i = W^K x_i$ and $v_i = W^V x_i$. The importance score given to a given preceding input with the current token is given by $score(x_i, x_j) = q_i \cdot k_j$, divided by the squared root of the embedded dimension ($\sqrt{d_k}$) for numerical stability. These scores undergo a softmax transformation for the determination of the output. All these operations can be modelled as matrix multiplications, and can therefore be parallelized:

$$SelfAttention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.10}$$

$$Q = XW^Q, K = XW^K, Q = XW^K \tag{2.11}$$

where X is the matrix with the input token embeddings. This process can be applied to different sets of W^Q,W^K and W^V, for different interpretations of

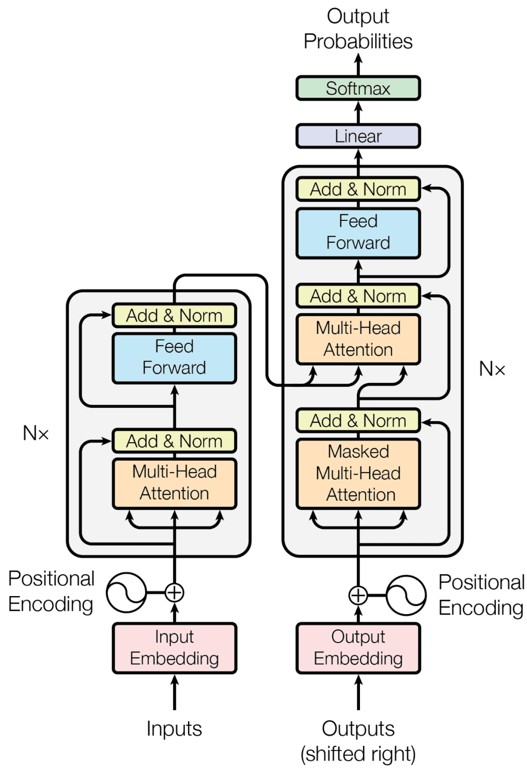


Figure 2.4: Transformer Architecture Diagram, as presented in Vaswani et al. [2017]

the relationships between inputs. A multi-head process where there are various attention heads, each with a different set of matrix parameters, allows for just that. The respective outputs are then concatenated and projected to the initial input dimension d with the learnable matrix W^O.

- **Transformer Blocks** – A transformer block is comprised of a multi-head attention layer, followed by a feedforward neural network. Both have normalisation after the output, and residual connections are present for better gradient propagation (see Figure 2.4). Multiple transformer blocks can be stacked, to allow for progressively more abstract representations of the input, leading to better language modelling. For example, one of the first transformer-based models, BERT [Devlin et al., 2019], had 12 stacked transformer blocks for processing text.

- **Softmax Layer** – For language modelling, the output of the transformer blocks is followed by a linear layer that projects it to the dimensionality of the output vocabulary. A softmax layer is then applied, so the next word predicted by the language model is determined.

The performance of these models is heavily dependent on two metrics, which dictate the amount of information considered in each inference, as well as the training and inference time:

- **Vocabulary size** – the greater the number of tokens, the greater the granularity of the semantic analysis. For example, it would be possible to have only characters as tokens, in which case the size would be 256 using ASCII; but this economy in the representation of the input and the output would make it harder to model the relationships among words. On the other hand, having an excessive amount of tokens could mean that some of them would not appear often enough in the training corpus, rendering the LLM unable to grasp their meaning.
- **Context window size** – the context window is the sequence of tokens that are considered for the computation of an answer. For a chatbot such as ChatGPT, where text is to be generated from an input, it tells how far behind the provided text is considered for computing an answer. Given that there are matrix multiplication operations in the determination of the outputs, and that the matrices have the size of the context window, the computation time of the outputs without optimizations could be quadratic with respect to that size. This limits the few-shot learning capacity of the LLM, where examples are provided for it to mimic them in a posterior answer – the only information that is considered when determining the output of the LLM apart from its general knowledge is this context window.

The arrangement of these blocks can vary depending on the nature of the task to perform. The different types of LLMs that result from this variation are described in the following subsection.

2.2.2 Taxonomy

Considering the possible tasks that can be performed with text, the information that is encoded by the LLM can vary. There are three main types of LLM architectures that allow for two different approaches:

- **Encoder-only architectures** create embeddings that contain the relevant information to complete a given task. For this, there are several transformer blocks between the initial token embeddings and the final latent vectors. The relevant information is, thereby, encoded in these latent vectors. Bidirectional Encoded Representations from Transformer (BERT) [Devlin et al., 2019] is an example of an encoder-only model. Its training involved masking part of the tokens in the text and making it predict them, considering the tokens before and after each masked token – masked language model. Thus, this model applied bidirectional attention, allowing it to have a thorough understanding of the connection among words in a text. After this pre-training stage, the model could be finetuned to other tasks, such as sentiment analysis, question answering, next-sentence prediction and named entity recognition [Wang et al., 2022].

- **Decoder-only architectures** are more specialized in text generation considering the input sequence of tokens. For this, the context window of token embeddings that precedes the current position is decoded throughout the transformer blocks to yield a probability distribution of tokens in the final softmax layer that estimates the next token in the sequence. Unlike encoder-only models, decoder-only models such as Generative Pre-Trained Transformers (GPTs) only apply unidirectional attention, as they do not consider tokens ahead of the current inference position. To train them, a process called teacher forcing is applied: input tokens are fed to the network, and the expected output is the next token in the sequence. In the next training step, it is the correct token (and not the one outputted by the LLM as being the most likely in the softmax layer) that is fed to the model as input for the determination of a subsequent output. After training, text production can be performed through autoregressive generation: the tokens in the context window are used to determine the next output token; and that token is fed to the model, which determines the subsequent token, until a termination symbol stops the generation. Similarly to the encoder-only models, these models can be further finetuned for a specific purpose. Obtaining a chatbot from these models involves creating a dataset of input-output sentence pairs that make the model go from completing text to executing an instruction - instruction finetuning.

- **Hybrid architectures** are comprised of an encoder and a decoder stage, aiming to capture complex relationships among tokens in both the input and output sequences. Hybrid models such as BART [Lewis et al., 2020] are robust to input noising transformations including masking, sentence permutation and token deletion, meaning that the input and output sequences need not be perfectly aligned. This makes them suitable for tasks such as text summarisation or translation.

In the context of the present work, only decoder-only architectures are considered, since a description of the features comprising a dataset is provided and a resulting output with relevant information is to be retrieved.

2.2.3 Finetuning

Finetuning an LLM is an important stage that can allow it to become less harmful, more truthful or more suitable to the needs of its end task. Because of this, after being trained on text of various domains, domain-specific finetuning datasets are used. Several algorithms have been developed for this stage and will be covered in this subsection, including Reinforcement Learning from Human Feedback (RLHF), Direct Preference Optimization (DPO) and Odds Ratio Preference Optimization (ORPO).

Reinforcement Learning from Human Feedback

A usual method to finetune an AI model is by using direct human feedback of some kind, as an iterative phase where this information is incorporated into the model's parameters. This was applied in models such as GPT-4 [OpenAI, 2023]: the human feedback is incorporated into the model as a reward function that has to be maximized. In the case of LLM finetuning, since the assignment of absolute scores by humans can be noisy, this information takes the form of human-made pairwise comparisons among responses, where a greater consensus is attained. Then, a Bradley-Terry model [Bradley and Terry, 1952] applies this information to estimate the ratio of the probability of generation between two different LLM responses. The goal is then to maximise the LLM-generated preferred responses over the rejected ones according to the human feedback modelled in the reward model, while not deviating excessively from the original pretrained model (not doing this could harm language modelling).

Considering decoder-only LLMs that are made to follow user instructions, after instruction-finetuning a model, where it is trained with instruction-output pairs for tasks of various kinds, a sample of model outputs is drawn and human preferences are annotated from them. The Bradley-Terry model is then trained on those samples and the pretrained model is trained to align to the preferences modelled in it after repeated sampling.

Despite the good results produced by this method, which was used to finetune the model behind ChatGPT, there are some downsides to it, including the cost associated with human annotation and the overhead associated with having two models to train: one language model that has to be optimized and another one to reflect human preferences. Other methods have been developed to minimise these issues, including DPO and ORPO, discussed below.

Direct Preference Optimization

DPO [Rafailov et al., 2023] is a possible alternative to RLHF to align LLMs to user preferences. It has the advantage of dispensing a reward model for this purpose. It is applied after an instruction-finetuning stage and requires a preference dataset, with (instruction, chosen response, rejected response) tuples. There are publicly available datasets that can be used for this purpose, but human annotation of LLM-generated answers is also a possibility.

The need for training a new reward model from pairwise comparisons is avoided by explicitly modelling the preference information in the finetuning loss function: the ratio of generation probabilities between a chosen and a rejected response is obtained based on the theory behind the Bradley-Terry models, and, similarly to what happens in RLHF, a regularisation term is added to the loss function to prevent the model from significantly drifting away from its initial state. When there is no specific instruction finetuning method prior to DPO, an initial supervised finetuning with (instruction, chosen response) pairs is a suitable strategy to obtain an initial state for the DPO stage.

The results obtained from this method showed its superiority in tasks such as text summarization, sentiment generation and dialogue when compared to the existing finetuning methods. Still, a task-specific finetuning stage is required before it can take place, which can be considered a downside. ORPO, described below, combines these two stages into one.

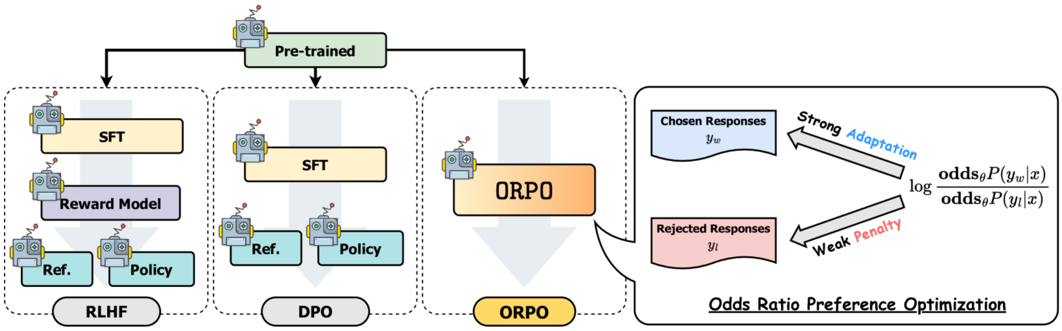


Figure 2.5: LLM Finetuning Methods: RLHF, DPO and ORPO

Odds Ratio Policy Optimization

ORPO [Hong et al., 2024] is yet another alternative for LLM finetuning that reconciles supervised finetuning with preference alignment in the same training stage through the loss function. The authors of this approach argued that applying a supervised finetuning stage preceded by a preference alignment stage was ineffective, since the crossentropy loss does nothing to penalise tokens that do not belong to a given answer, also favouring undesired styles or response formats.

To avoid this, to complement the standard finetuning loss based on the chosen responses, a new term to reward them while penalising the rejected ones is added to the loss function. It is based on the odds ratio between the generation of a chosen and a rejected response. To calculate the odds metric for a sequence, its log-probability of generation is determined by averaging the log-probabilities of each token in the sequence, considering the ones that occurred before. After this, the probability metric is obtained and the odds metric is determined by calculating the ratio between the probability of the sequence and its complementary – it tells how much more likely it is that the sequence is generated than that it is not. Therefore, the odds ratio tells how much more likely it is for the model to generate the chosen response over the rejected one. By taking the negative logarithm of this in the loss function, this quantity will be maximized throughout the finetuning process:

$$\log P_\theta(y|x) = \frac{1}{m} \sum_{t=1}^{m} P_\theta(y_t|x, y_{<t}) \tag{2.12}$$

$$odds_\theta(y|x) = \frac{P_\theta(y|x)}{1 - P_\theta(y|x)} \tag{2.13}$$

$$\mathcal{L}_{OR} = -\log \sigma(\log \frac{odds_\theta(y_w|x)}{odds_\theta(y_l|x)}) \tag{2.14}$$

$$\mathcal{L}_{SFT} = -\frac{1}{m} \sum_{k=1}^{m} \log p_{y_k} \tag{2.15}$$

$$\mathcal{L}_{ORPO} = \mathcal{L}_{SFT} + \alpha \mathcal{L}_{OR} \tag{2.16}$$

2.2.4 Optimisation Strategies

Given the high computational requirements of LLMs, some optimisation strategies have been implemented and applied to state-of-the-art LLMs. Some of them were applied to model finetuning – Low-Rank Adaptation (LoRA) and Quantized Low-Rank Adaptation (QLoRA) –, whereas others changed the models' architectures – Grouped Query Attention (GQA).

LoRA and QLoRA Finetuning

The finetuning process of an LLM can involve changing all its weights, meaning that they have to be stored in their totality. This can make finetuning impeditive in contexts with storage constraints. However, it was also found that the difference in the parameter matrices between the finetuned and pretrained model has a low rank. Because of this, a training scheme where this difference is trained as a product of two low-rank matrices, called LoRA [Hu et al., 2022], was developed. The rank can be as little as 8, instead of 1000 to 10000. This makes the trainable parameters fit in a small fraction of the original parameters – Parameter Efficient Finetuning (PEFT) –, allowing to store significantly more finetuned models in the same space, while avoiding overhead in inference time, since model parameters can be precomputed and stored in before deployment. The application of this process to the attention heads present in the transformer architectures of LLMs such as RoBERTa, GPT-2 and GPT-3 yielded very similar results to the ones obtained without this optimisation strategy at a fraction of the trainable parameters.

The process of storage economy can be further extended by quantising the trainable parameters. The resulting algorithm, QLoRA [Dettmers et al., 2023], allowed for the finetuning of models with 16-bit float parameters using only 4 bits per parameter. To this end, the NormalFloat4 data type was used: it is an information-theoretically optimal data type that encodes each float as a given quantile in the $N(0,1)$ distribution – dequantisation involves determining the correct number in the distribution and rescaling it by the correct standard deviation value. Parameters are quantized in a block-wise fashion and the block-specific quantization constants are themselves quantized as well, minimizing the memory footprint. Finally, QLoRA also applies a paging mechanism where the states of an optimizer are transferred to the Central Processing Unit (CPU) if the GPU runs out of memory (see Figure 2.6).

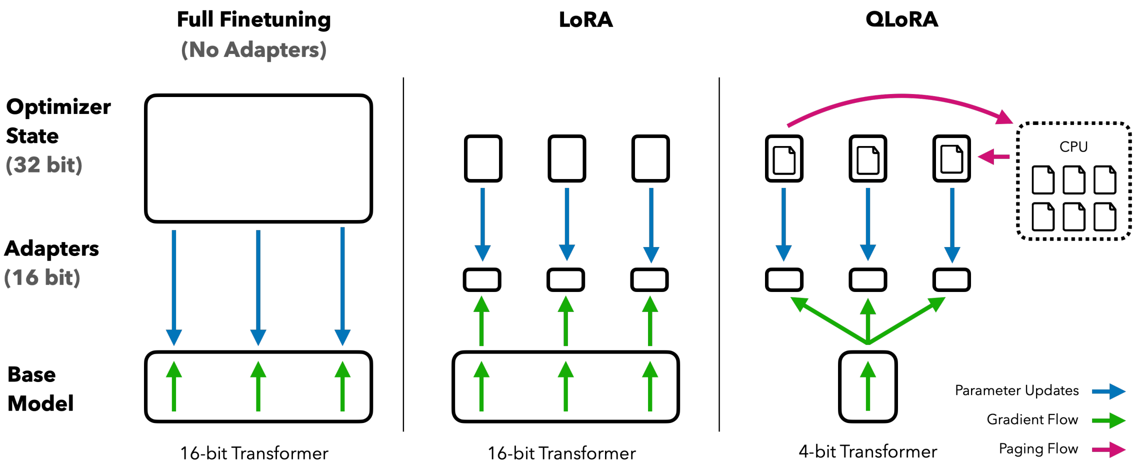


Figure 2.6: Parameter-Eficient Finetuning: LoRa and QLoRa

Grouped Query Attention

An issue with standard Multi-Head Attention (MHA) is the large number of parameters that have to be loaded into memory when performing both training and inference: there are three matrices for each attention head, which can produce significant overhead in memory access operations. In an attempt to minimise this problem, a new attention mechanism was invented: Multi-Query Attention (MQA). In this scheme, all attention heads shared the same value and key matrices and only the query matrices varied across heads. As a consequence, the number of parameters is reduced to just slightly more than one-third in the transformer blocks, and inference time is sped up.

The decrease in parameter count resulting from this led to performance degradation. An alternative to reconcile the performance of multi-head attention and the economy of multi-query attention was then created: GQA [Ainslie et al., 2023]

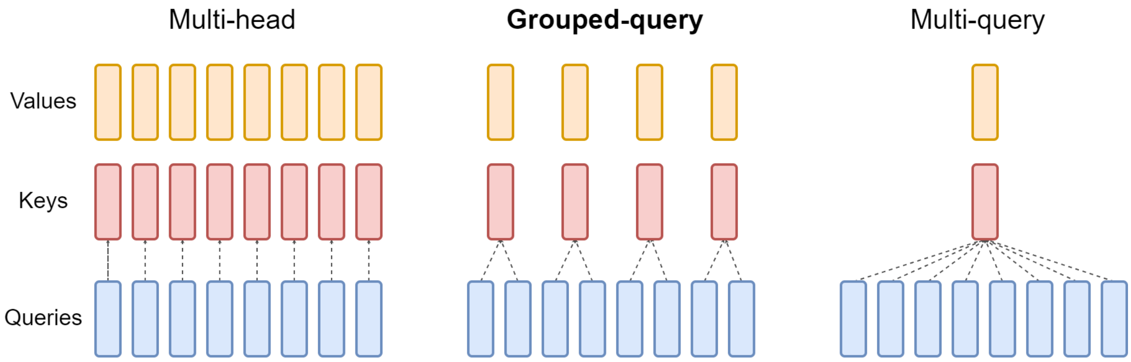
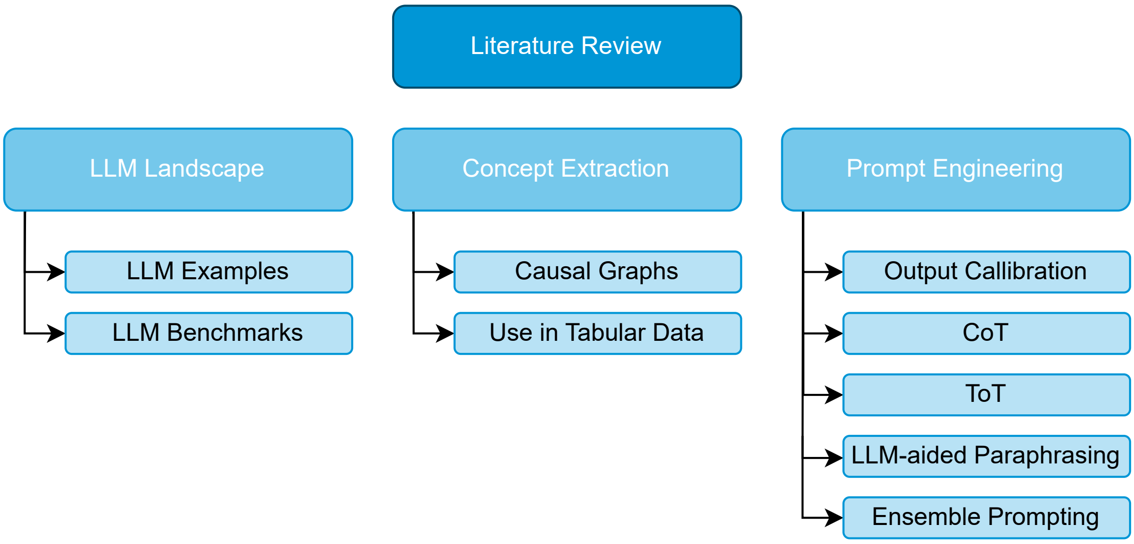


Figure 2.7: Attention Modes: Multi-Head, Grouped-Query and Multi-Query

creates groups among attention heads (where each group has 8 attention heads, for example) where key and value matrices are shared (see Figure 2.7). Thus, there is still a significant economy in parameter count. Adaptation from models that implemented MHA involved obtaining the average key and value matrices for each head group and performing an uptraining stage with the same dataset and a small portion of the compute. Experiments with several datasets showed that the resulting performance is very similar to the one obtained with MHA while maintaining an inference time similar to the one found in MQA. These results led to the adoption of GQA as the attention scheme in popular LLMs such as LLaMa3 [AI@Meta, 2024].

Chapter 3

Literature Review



A literature review is presented in this chapter, starting with Section 3.1, where we will provide some information regarding popular LLMs and the respective performance across standard benchmarks. Then, Section 3.2 will focus on the approaches that are most similar to the present work by working with tabular data and LLMs to improve the performance of ML models. Finally, in Section 3.3, prompt engineering will be discussed, including some practices and search algorithms for prompt optimization.

3.1 LLMs – current capabilities and features

Bearing in mind that the present work has LLMs as a central concept, the following section aims to summarise the current LLM landscape.

With the release of ChatGPT in November 2022, and other LLMs after it, it became clear that many repetitive text tasks could be performed by these models with an accuracy that makes them useful. Given that there is great variety in features and performance throughout various tasks, it is important to summarise the various LLMs that can be used for the purpose of increasing generalisation in tabular classifiers, including benchmarks, number of parameters, and context window size.

3.1.1 Popular examples

The following list shows popular LLMs in use nowadays:

- **ChatGPT** by OpenAI – this LLM was introduced in November 2022 and has been in continuous development. There are two main versions in use in ChatGPT. The smaller one is ChatGPT-3.5, with public access in the browser version. It was released in November 2022 and has incorporated updates that make it faster and more performant. The larger one is GPT4 [OpenAI, 2023], a larger, improved version with integration with images and sound and more limited access.
- **Gemini** by Google [Team et al., 2023], released in February 2024. This LLM was an evolution of its predecessor, Bard, and the most advanced version (Gemini 1.5 Pro) reports several benchmarks where it surpasses GPT-4;
- **Claude 3.5 Sonnet** by Anthropic[1], released in June 2024. This LLM also aims to set new performance boundaries on code generation, while integrating image and video processing. Text generation occurs at twice the speed of the predecessor, Claude 3.0 Sonnet;
- **LLaMa 3** by Meta [AI@Meta, 2024] – this LLM was released in April 2024 in different model sizes – 8B and 70B, with the larger version displaying performance similar to Gemini in some benchmarks. Unlike the other LLMs listed here, LLaMa3 is openly available, meaning it can be tested and fine-tuned by the community.

Throughout time, these LLMs have undergone improvements in several directions, including generation speed, context window length, and quality of gener-

[1] https://www.anthropic.com/news/claude-3-5-sonnet

ated content, including code, math questions, and general reasoning. The evaluation of LLMs is based on benchmarks that evaluate the quality of translations and reasoning in several fields of expertise. The following section presents commonly used benchmarks, which were also applied to the LLMs in this list.

3.1.2 Common benchmarks

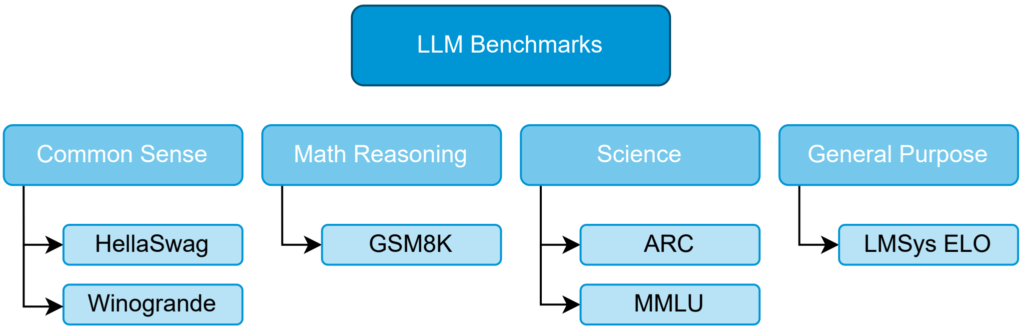


Figure 3.1: Common LLM benchmarks and respective scopes

There is a plethora of areas where LLMs can be of use; meaning that their utility across all of them should be evaluated (see Figure 3.1). The following list makes reference to some of them:

- **HellaSwag** (Harder Endings, Longer contexts, and Lowshot Activities for Situations With Adversarial Generations) [Zellers et al., 2019] is made of 70 thousand multiple-choice questions based on WikiHow[2] and ActivityNet [Caba Heilbron et al., 2015] that assess common sense reasoning: humans can achieve very high scores on these questions (superior to 95%); whereas the LLMs assessed upon the release of this benchmark had a performance inferior to 50%. The questions were based on a previous dataset, SWAG, and the choices were iteratively made harder using language model-based Adversarial Filtering: the questions easily classified as wrong are replaced by more adversarial endings, aiming to make the language model fail in them.

- **Winogrande** [Sakaguchi et al., 2021], a dataset of 44 thousand problems assessing commonsense reasoning. These questions were based on the Winograd Schema Challenge [Levesque et al., 2012] and their difficulty was made harder by minimising the unintended word association biases present in

[2] https://www.wikihow.com/Main-Page

that dataset using the AF-Lite algorithm, a lightweight version of Adversarial Filtering [Bras et al., 2020]. Human performance is also very high in this benchmark (approximately 94.9%), whereas the state-of-the-art models at the time of release were 15% to 30% worse.

- **GSM8K** [Cobbe et al., 2021], a set of 8500 grade-school math word problems, which take two to eight steps to solve using the four basic mathematical operations of addition, subtraction, multiplication and division. These problems were man-made in a strive for diversity and quality. Given the simplicity of these tasks, human performance should be perfect. LLM performance, on the other hand, required further improvement: ChatGPT-3 achieved about 55% at the time of release, based on a verification procedure that used LLM-generated answers to improve LLM performance on the run.

- The **ARC** (AI2 Reasoning Challenge) [Clark et al., 2018] is comprised of 7787 multiple-choice scientific questions with levels of difficulty from the 3rd grade to the 9th grade, with questions that span from definitions and basic facts to spacial awareness and algebra. This question dataset was partitioned into two subsets, Easy and Challenge (ARC-C), where the more challenging questions were the ones that could not be solved using word co-occurrence algorithms or information retrieval systems. A knowledge base was also published, the ARC Corpus, containing information relevant to 95% of the questions present in the dataset, to allow the community to use the benchmark.

- **MMLU** (Massive Multitask Language Understanding) [Hendrycks et al., 2021] consists of 15908 questions spanning various subject matters from social sciences to humanities and STEM in difficulty levels from Elementary to Professional. Human performance spans from 34.5% in non-experts to 89.8% in experts. Therefore, this benchmark evaluates expert knowledge in a variety of fields. Upon the release of this benchmark, GPT3, a precursor to ChatGPT-3, had achieved an average performance of 43.9%.

- The **LMSys Chatbot Arena ELO** score [Chiang et al., 2024] is a benchmark based on the pairwise comparison of LLMs performed by millions of users. This comparison is blind and collaborative: based on the prompt that the user provides and on the answers outputted by each of the LLMs, he can say which answer is better. Millions of such comparisons allow for the calculation of an ELO score, analogous to the one used in chess ratings, where the difference in scores between two LLMs maps to the proportion of times one LLM was better than the other. A categorisation of the prompts then allows for the calculation of a task-specific score.

The benchmarks mentioned in this list were applied to the analysed LLMs, and the results are presented in the next subsection.

3.1.3 Benchmark Results

The various benchmarks presented in the last subsection were applied to the LLMs for comparison. The results can be consulted in Table 3.1. The data have been gathered from [OpenAI, 2023; Team et al., 2023; Touvron et al., 2023].

Table 3.1: LLM Performance Metrics: ARC tests abstraction and reasoning, HellaSwag and Winogrande test commonsense reasoning, MMLU tests expert knowledge and multitasking, GSM8K tests math reasoning, the LMSys ELO is an overall performance score

Model	ChatGPT3	GPT4	Gemini	Claude 3.5	LLaMa3-8B
Release Date	nov/22	mar/23	mar/23	jun/24	apr/24
Context Window	8192	128K	-	200K	8K
ARC (%)	85.2	96.3	95.1	-	78.6
HellaSwag (%)	85.5	95.3	87.8	-	-
MMLU (%)	70.0	86.4	83.7	88.7	68.4
Winogrande (%)	81.6	87.5	83.0	-	76.1
GSM8K (%)	57.1	92.0	94.4	96.4	79.6
LMSys ELO	1105	1257	-	1271	1152

As can be seen from the table, performant LLMs have diverse capabilities and applications. For example, considering the relatively small size of LLaMa3 when compared to the other models, even if it performs worse than the other models in the various benchmarks, it could be a suitable choice for contexts where access to memory is more constrained.

After the main LLM performance indicators have been presented, the field of concept extraction, a task to be undertaken in the present work, will be discussed in the next section.

3.2 Concept Extraction

One possible way to improve the training of ML models is by trying to incorporate extra real-world information that is not directly present in the data fed to the model, in the training process. In the past, this was achieved by learning causal graphs and by prompting feature-level attributes from LLMs. This section aims to document some advancements in this area and introduce the methods that led to them.

3.2.1 Causal Graphs as source of knowledge

One of the advancements in this area is CASTLE [Kyono et al., 2020], a framework where the causal structure among the features and target of a dataset is discovered along with the training of a model. For this, the causal structure in the data is represented as a directed acyclic graph where some features are determined by their parents in the graph. Here, the causal graph is represented as an adjacency matrix and the loss for the model that learns its values aims to make it acyclic. The closeness between this causality and the one verified in the model is then also added as a term to the loss function to minimize.

This kind of training produced state-of-the-art AUROC when applied to several datasets when compared to other popular regularisation techniques. This implies that using relationships among features to improve model training is possible. In the present work, relationships of a similar kind are to be extracted from the LLM.

3.2.2 LLM use in handling tabular data

In the past few years, works that aim to harness the concepts learned by LLMs have been developed. Their findings and methodologies are detailed here.

LLM-based regularization

The use of LLMs to improve training with tabular data is not entirely new but is still an area of active research. In [Zhu et al., 2023] a possible LLM-based regularisation scheme is described: on one hand, for multivalue categories, the

LLM can order them in terms of their importance in determining the output – this produces an ordinal encoding that is then perfected when passed by a monotonically increasing function with learnable parameters that better approximates the importance of each category value. Likewise, the LLM can tell whether there is a positive or negative correlation between a feature that is a continuous variable.

Considering a logistic regression task, where each feature value has a coefficient, the distance between this coefficient and the prior provided by the LLM is present in the loss as a regularisation term.

The results displayed by this methodology show interesting AUROC values, especially in few-shot situations (less than 32). This shows that with very little training data and the knowledge of the world embedded in its parameters, an LLM can help solve tasks effectively. The future work of this paper referred to the possibility of further studying the inclusion of more complex priors. It can be considered that the present work develops on this, given that it extracts more complex formulas from LLMs.

LLM data generation

LLMs do not need to provide raw insights on data, as there are alternative ways of achieving performance improvement on tabular classifiers. Curated LLM [Seedat et al., 2024] is a framework that aims to tackle this problem by making LLMs generate data for training. This is especially relevant in low-to-middle-income contexts where data gathering is insufficient for robust models to be produced.

This work provides the LLM with a set of examples from the initial dataset and then prompts it to generate more examples, intending to feed them to a tabular classifier. Given that not all examples have the same quality and utility, this framework will periodically estimate the benefit of the inclusion of a feature in the dataset, using the probabilistic outputs of the network being trained to determine the average confidence (the average output of the target class throughout the checkpoints) and the respective aleatoric uncertainty (the observed variance of those outputs) – a sample will be included if its predictions are above a given threshold of confidence and a given threshold of uncertainty. The resulting selected and discarded datasets are then used to train two different models, and a held-out oracle dataset is used to evaluate their performance.

From this work, it was concluded that the data partitioning strategy resulted in a significantly better performance when training with the selected dataset when compared to the training with the discarded dataset. After testing with several medical datasets (including *covid*), it was also found that even in ultra-low data regimes ($n < 100$) GPT-4 extrapolated well to the data manifold and benefitted underrepresented demographic groups the most, showing that the context embedded in its parameters allows it to have knowledge about the problems at hand and to generate relevant examples for training.

Table learning with transformers

The use of the transformer architecture for interpreting tabular data did not start with the larger LLMs such as ChatGPT, LLaMa and PaLM, as experiments were already being conducted with the smaller BERT model. Table Understanding through Representation Learning (TURL) [Deng et al., 2022] was developed in 2020 to adapt transformers to tables.

To this aim, the embedding associated with the words in the table metadata register would include information about their location (caption or header), as well as the usual word and position embeddings. The table content would also have a specific embedding: each entity present there would have its own embedding, including the attribute name, the attribute text and a type embedding to distinguish subjects, objects and topics.

After these transformations were applied to the various elements present in the table, a sequence of embeddings would be produced and fed to a structure-aware transformer encoder. The main difference between this architecture and that of a standard transformer was the use of a visibility matrix that determines the relationship among elements: the name of a column will only relate to the tokens from that column, the table caption and topic entity are always visible; tokens in cells of the same row or column are also visible among themselves.

The training process of this transformer involved applying the usual process in the training of the BERT transformer: masking some tokens for prediction with the context around them. By making this process look at the table metadata, the variant of Masked Entity Recovery was created.

The various benchmarks in this architecture revealed advancements in the various applications of entity linking, column type annotation, relation extraction, row population, and cell filing.

This work is another example of the possibility of making transformers learn the content in tables, even if this implies slightly changing their architecture.

3.3 Prompt Engineering

Prompt engineering is an area that rose with the popularisation of LLMs: after these models had public access, it became essential to know how to extract relevant and accurate information from these models. As such, this area was created; along with the respective standard practices. This section aims to present some findings in the area, as well as some prompt optimisation algorithms.

3.3.1 LLM Output Callibration

A work [Zhao et al., 2021] revealed several kinds of biases displayed by LLMs in their answers to prompts. In the scope of few-shot learning, where a few examples and respective target outputs are fed to the LLM before a similar example is given for classification/answering, it was understood that label distribution, order and token choice greatly affect the performance displayed by the model. This was achieved by prompting some training examples and respective outputs and then retrieving the output from the LLM when the example was content-free (e.g. N/A).

The following biases were detected in the work:

- Majority label bias: in prompts with an imbalanced distribution of examples, the LLM predicted the majority class in an excessive proportion when compared to the reality;
- Recency bias: the LLM tended to predict similarly to the most recent examples, as opposed to the examples closer to the beginning. This showed the importance of the permutation of the examples fed to the model;
- Common token bias: frequent tokens used for classification in the training examples would appear more often than others.

After returning the classification probabilities from the LLM, these authors applied an affine transformation to them ($q = softmax(Wp + b)$), in an attempt to make the LLM outputs as close to uniform as possible when dealing with content-free prompts. This led to improvements in accuracy in various few-shot learning situations, reducing variance across training sets, and making calibrated smaller models outperform larger ones (GPT3-2.7B vs. GPT3-172B).

This work reveals the biases present in LLMs and proposes a way to handle them. With this, it shows the importance of iteration in the prompting process, as a way to make sure the biases are detected and that the information being extracted is truly reliable.

3.3.2 Chain of Thoughts Prompting

Bearing in mind the training process and structure of LLMs (see Section 2.2), which are trained with considerable amounts of text, they do not possess a specialized unit for mathematical operations that would be easily conducted by a CPU, meaning that the results displayed by these models are only the outcome of statistical patterns among digits and operators. As such, without optimization, they perform relatively poorly in mathematical tasks, as the paper that presented GSM8K [Cobbe et al., 2021] shows.

A strategy to improve their performance was found and documented in [Wei et al., 2022]. The simple act of asking the LLM to print the intermediate steps that lead to the answer is enough to achieve a significant improvement in its results. This way, the relationship between numbers and words can be better understood by the model. The precursor of Bard, PaLM, underwent a three-fold increase in performance with the use of this tactic (from 18% to 57% accuracy) in the GSM8K benchmark. After this technique was published, OpenAI used it for reporting its results [OpenAI, 2023] on this benchmark.

3.3.3 Tree-of-Thoughts Framework

In [Yao et al., 2023], a new framework for prompt engineering was created. It aimed to guide LLMs in the search for an answer to a question. Firstly, an external prompter inputs the statement of the problem. The intermediate steps towards achieving this are be produced by the LLM, one by one; and an external checker validates them. If an intermediate step is wrong, the LLM would be asked to return to the previous state, with an external memory module allowing for retrieving the past chain. As new solution paths to a correct output are searched, the shape of the graph where all the thoughts are connected is one of a tree. The external checker and prompter are trained using reinforcement learning that aims to make them learn which information to check and give the LLM to make it solve the problem correctly.

This framework was tested in Sudoku games with sizes $n * n, n \in \{3, 4, 5\}$; and significant improvements were obtained in all of them with comparison with the respective state of the art – all the 3-by-3 sudokus were solved, and the success rate increased 80% in 4-by-4 puzzles and 60% in 5-by-5 puzzles.

3.3.4 LLM-aided paraphrasing

In 2021, Haviv et al. proposed BERTese [Haviv et al., 2021], a framework that aimed to improve masked prompts (prompts with clozes for completions, signaled by the token *[MASK]*) to improve the query accuracy when LLMs were asked to fill in the gaps.

For this, an off-the-shelf BERT model was used as the estimator/source of knowledge, and a BERT-based rewriter took the masked prompt and was meant to produce one with the same number of tokens, such that downstream accuracy was improved. The rewriter initially had the non-pretrained BERT weights; then it was trained to yield the same tokens as it was fed (in the same order). Finally, queries to improve were fed to the rewriter and the respective result was then fed to the BERT model. The result of this prompt would then be evaluated and the resulting loss would be backpropagated to the rewriter.

The experiments performed with the rewriter revealed that it led to prompts that were better at knowledge retrieval from the BERT model, even though they had very similar meanings – the changes happened, for example, in verb tenses, or by replacing uncommon words with more frequent ones. This work showed that it is possible to improve prompt quality, given the variability of the LLM's responses to prompts of equivalent meaning but expressed in different wording.

3.3.5 Ensemble prompting strategies

If the previous methods were based on solely one prompt that could be corrected in a serialized way, the following ones aim to optimise prompts by producing multiple alternatives and converging in more performant inputs.

Ask Me Anything

A work [Arora et al., 2023] describes an alternative way by which better quality prompts can be achieved. In this experiment, it was found that open-ended questions tend to produce better results than yes/no questions or cloze questions (questions where gaps in a text have to be filled in). This can be attributed to the quantity of training data used to train the LLMs that matches the latter type – the article states it is about 1000 times greater.

As well as this, a prompting strategy was documented: creating several different prompts where a given text input is provided to the LLM for it to ask questions based on it; so that it can answer those same questions. The resulting answers are then aggregated into a single one, based on weak supervision, which aims to identify the dependencies among prompts with the aim of reaching a reasonable ensemble answer that takes prompt redundancies into account. This strategy leads to expressive improvements in performance, potentially allowing smaller models to outperform larger ones when using this kind of preprocessing.

Thereby, this work shows the importance of the format of prompts and of prompt engineering as a whole, as having well-formatted questions can be of greater use than spending computational resources training a larger model.

Boosted Prompt Ensembles

Traditional ensemble algorithms that are already widely used in other areas, such as boosting, can also be applied to prompt optimization. Pitis et al. describe an approach where that is achieved in Chain of Thought reasoning [Pitis et al., 2023] – a method that makes LLMs output their reasoning in complex problems (such as mathematical ones) in a step-by-step fashion what allows achieving more accurate responses than if using shorter outputs.

In the beginning, there is one initial example prompt and a set of questions and respective answers. These questions are fed multiple times to the LLM along with the existing example prompts so that the LLM's performance on the questions can be evaluated. For the questions that produce greater uncertainty (where the model's solutions disagree throughout the repetitions), one of the LLM's Chain of Thoughts (CoT) answers where the correct information was presented is included as a prompt example to present in future training iterations.

The comparison of this method with the single-prompt and bagged ensembles revealed a superiority of the resulting LLM performance. This method can be seen as an iterative algorithm for covering the prompt space to maximise the accuracy in few-shot learning.

3.3.6 Prompt Engineering with Evolutionary Computation

Population-based optimisation is also a possibility in prompt engineering. Various works that use evolutionary computing for this purpose are proof of this.

EvoPrompt

Guo at al. proposed a framework, *EvoPrompt* [Guo et al., 2024], that proves this: evolutionary computation is used for optimising the prompts fed to LLMs, and these models are also themselves responsible for applying the operators associated with this process.

The algorithm starts with a population of manually designed and LLM-generated prompts. Then, the process to follow is the same as the usual genetic algorithm: the crossover operator is applied by asking the LLM to analyse two prompts and to produce a mixture of the two, and the mutation operator is applied by asking the LLM to make a slight change on a given prompt. The fitness of a prompt was the performance it achieved in a given development set, and the selection process was a roulette-based one (where the probability of selecting a prompt to be used in the next generation was proportional to its fitness). The repetition of this process (selection, variation operators, evaluation and population update) throughout a given number of generations allowed for the generation of optimized prompts. Differential evolution was also tried: in that case, the differences between two prompts were analysed and inserted into a third prompt, all of this being controlled by an LLM as well.

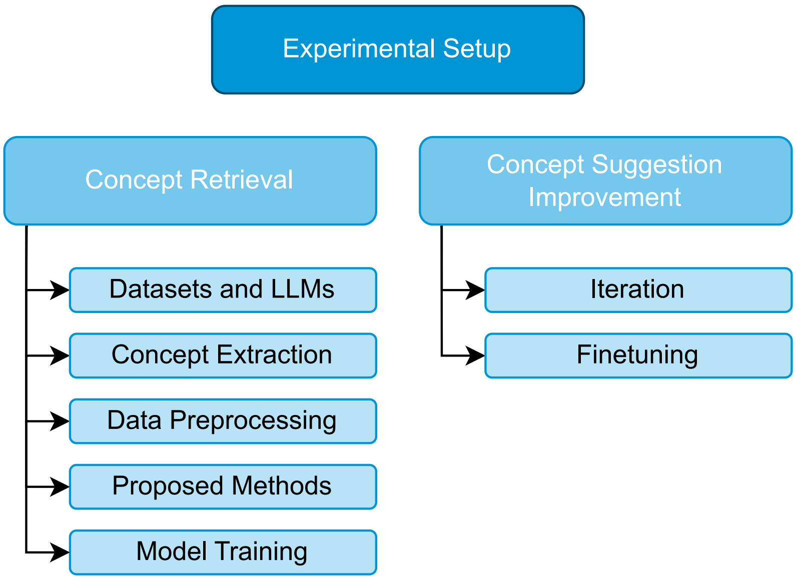
The subsequent benchmarking revealed that the prompts produced had a superior performance in tasks such as sentiment classification, topic classification, subjectivity classification, and language generation. This work revealed one of the possibilities of combining the knowledge of LLMs with conventional algorithms, where the LLMs are part of the implementation as they perform the variational operators.

Metaprompter

Text generation is not the only scope of prompt engineering, given that generative models can also generate images, for example. As such, it is also possible to find works that apply the same principles in this field. It is the case of the *MetaPrompter* [Martins et al., 2023], in which a user inputs a meta prompt, with several generic fields (e.g. animal, fruit, style of the image) whose concrete value can be chosen across predefined lists. In this case, the fitness is user-determined, in the way that it is the user that chooses the fittest prompts to produce the next generation of images. These prompts then undergo mutation (which can change, insert, or remove terms from the prompt) and crossover, where given parts of a prompt are exchanged. The results showed that the images generated were aesthetically pleasing, even though they did not necessarily represent what was intended in the prompt. Even though this experiment was applied to images, it could be possible to apply it to text generation as well, if a prompt skeleton was given; and the evolutionary process could proceed automatically.

Chapter 4

Experimental Setup



To explore the research questions posed in the introduction, an experimental setup was built. The aim of the present chapter is to describe it, contextualising the results that will be discussed in chapter 5. This chapter is divided into two main sections, one for each research question.

4.1 Concept Retrieval

The first research question tackled in this work was: can we increase an ML model's generalisation by embedding the information extracted from LLMs into the model?

More concretely, we focused on classification tasks in supervised learning set-

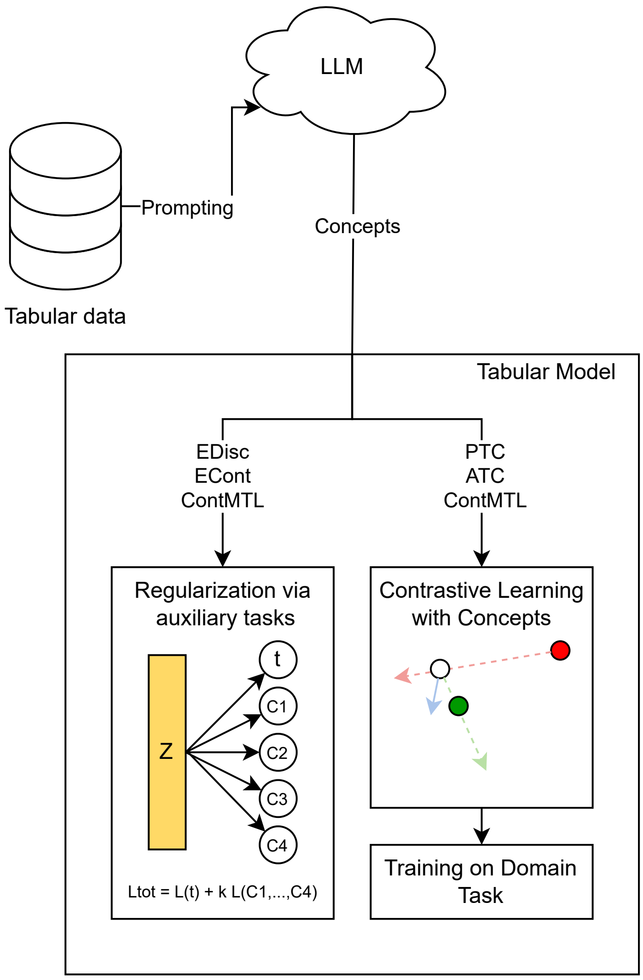


Figure 4.1: Scheme of the proposed experimental setup for the concept retrieval phase

tings using tabular data. To answer the research question, concepts for various datasets were retrieved from LLMs, and multiple directions for incorporating them in tabular classifiers were explored. In this section, the experimental setup that produced answers to the first research question (schematized in Figure 4.1), including used datasets, LLMs, methods and performance metrics, shall be described.

4.1.1 Datasets and LLMs

Several tabular classification datasets were used to validate the proposed methods. Their domain applications span from medicine to gastronomy, also including chemistry and business. They were chosen based on the number of features,

Table 4.1: Tabular Classification Benchmark Datasets. n is the number of instances, c is the number of classes, f is the number of features, f_c and f_d are the number of continuous and discrete features, respectively

Name	n	c	f	f_c	f_d	Domain
Apple Quality[1]	4000	2	7	7	0	Gastronomy
Breast[2]	286	2	9	4	5	Medicine
Breast Wisconsin[3]	569	2	30	30	0	Medicine
Cervical[4]	857	2	33	12	21	Medicine
Diabetes[5]	768	2	8	8	0	Medicine
Glass[6]	214	6	9	9	0	Engineering
HCV[7]	615	5	12	11	1	Medicine
Heart[8]	918	2	11	6	5	Medicine
Iranian Churn[9]	3150	2	13	10	3	Business
Thoracic[10]	470	2	16	3	13	Medicine
Urinalysis[11]	1436	2	14	5	9	Medicine
Wine Quality (red)[12]	1599	7	11	11	0	Gastronomy

which should not exceed 50 so that a relatively short descriptive prompt could be made. Also, the datasets should contain enough data – at least a few hundred instances – to represent the task at hand and solve it using MLPs. The list of used datasets is provided in Table 4.1. The number and diversity of datasets are meant to ensure the generalizability of the experiment's results.

To evaluate concept extraction from LLMs, two LLMs were used in their web browser version: ChatGPT versions 3.5 and 4. They were chosen due to their popularity and high scores in the LMSys Chatbot Arena[13]. They were also a suitable way to compare the performance of the proposed methods between free and paid LLM usage tiers since GPT-4 only supported paid access at the onset of the work[14].

[1] https://www.kaggle.com/datasets/nelgiriyewithana/apple-quality
[2] https://archive.ics.uci.edu/dataset/14/breast+cancer, from [Zwitter and Soklic, 1988]
[3] https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data
[4] https://archive.ics.uci.edu/dataset/383/cervical+cancer+risk+factors
[5] https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset, from [Smith et al., 1988]
[6] https://www.kaggle.com/datasets/uciml/glass
[7] https://archive.ics.uci.edu/dataset/571/hcv+data, from [Hoffmann et al., 2018]
[8] https://archive.ics.uci.edu/dataset/45/heart+disease, from [Detrano et al., 1989]
[9] https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset
[10] https://archive.ics.uci.edu/dataset/277/thoracic+surgery+data
[11] https://www.kaggle.com/datasets/avarice02/urinalysis-test-results
[12] https://archive.ics.uci.edu/dataset/186/wine+quality, from [Cortez et al., 1998]
[13] https://chat.lmsys.org/
[14] https://openai.com/index/hello-gpt-4o/

After the choice of datasets and LLMs, the prompts for concept extraction were produced. Their structure is described in the following subsection.

4.1.2 Concept Extraction

The concepts to extract from LLMs were nonlinear combinations of features, whose formulas were directly retrieved from LLM responses. A total of five concepts were obtained for each dataset, a suitable number since it is not too large but can impact performance, allowing us to test our methods. The prompt fed to the LLM started by contextualising the task at hand. This was followed by a textual description of each feature in the dataset. Finally, a request for useful high-level concepts, describing them as nonlinear combinations of features and providing examples, was included. The list of concepts would be comprised of the formula describing each concept, as well as a brief explanation of its utility. For example, for the Breast dataset, the prompt was the following:

I am building a diagnostic model that predicts whether a given set of circumstances led to the reincidence of breast cancer in a woman.

Each woman is characterised by: age, menopause stage (less than 40, more than 40 or pre-menopause), tumor size, number of inv-nodes, existence of node caps, degree of malignancy, breast side (right or left), breast quadrant and whether the zone suffered irradiation.

What are helpful higher-level concepts to help with this prediction task? Concepts should be engineered features derived from the features provided, and aim to capture a fundamental component necessary for the prediction task, based on your scientific and medical understanding. Examples of concepts include stripes in images or Gleason score grouping in prostate cancer mortality prediction. These concepts must not be simple linear combinations of features, but must be more complex and non-linear, including the product/quotient between two or more features and/or the use of other nonlinear functions.

Please suggest 5 concepts and provide explicit formulas to calculate the concept from the features specified above. You must provide explicit formulas, including all coefficients, and must not use features other than those provided. Provide a brief justification for the concepts you provided.

It can be seen that this prompt is comprised of four main parts that direct the

LLM to concept suggestion: a contextualisation of the task, a characterisation of the dataset, the question to answer with examples and the concrete instructions to follow. It follows prompt engineering practice, since the examples and the instructions provide the necessary context for an LLM to suggest concepts.

The formulas contained in the LLM's response to this prompt are used in the following stage of data preprocessing, which shall be discussed in the next subsection.

4.1.3 Data Preprocessing

After the concept retrieval phase, the data present in the datasets was preprocessed. Categorical features were encoded with the appropriate algorithm (ordinal or one-hot, depending on whether there was a logical ordering of the categories) so that all the features could have the numerical representation required to serve as MLP inputs. Following this, train-test split (stratified for the classification datasets) was conducted. Then, missing data was imputed using the median of the training dataset. This concluded the data preprocessing stage before concept formulas were applied.

After this, the five concepts retrieved for the dataset were calculated. In the case that there were ratios with divisions by zero, a constant (1) would be added to the denominator for stability. This step produced concept-labelled datasets for the subsequent tasks in the experiment's pipeline.

Some of the methods proposed in the experiments required the concepts to be discretized to binary values. Because of this, a discrete version of the concept-labelled datasets was also obtained. For this, a thresholding function was applied to each concept, such that each threshold was the one that maximized the target task information gain (minimising the entropy in the separation between the instances with a concept value of 0 and the instances with a concept value of 1):

$$\arg\min_{x} \frac{\#\{c_{ij} \in C_i : c_{ij} < x\}}{\#C_i} \times H(Y, C_i < x) + \frac{\#\{c_{ij} \in C_i : c_{ij} \geq x\}}{\#C_i} \times H(Y, C_i \geq x), x \in \mathbb{R} \quad (4.1)$$

After obtaining the datasets with discretized concepts, it was possible to test the experimental hypothesis. The underlying methods are described in the next section.

4.1.4 Proposed Methods

Considering that the present work aims to produce improvements in tabular classifier generalizability through the concepts provided by an LLM, it is necessary to develop methods that make use of those concepts and embed them in an MLP's latent space. Two possible ways of achieving this were explored: multi-task learning and contrastive learning. Different combinations using the continuous and discretized concepts were tried, leading to the various proposed methods described below:

- **PTC** (Pretraining with Triplet Contrastive learning) is a process where the discretized concepts are learned by the neural network through contrastive learning with triplet loss. This happens before the end task training stage and allows the latent space to be structured around the concepts before the supervised training process. The proposed implementation applies triplet contrastive loss [Le-Khac et al., 2020] for this purpose;
- **ATC** (Alternation with Triplet Contrastive learning) consists of applying the discretized concept contrastive learning in alternation with the end task training epochs;
- **EDisc** (Discrete Regularization) extends the output layer with m boolean outputs, one for each concept. The neural network then predicts these outputs and is trained on both the concepts and the end task. Concept training is subject to a regularisation term that regulates concept importance;
- **ECont** (Continuous Regularization) follows the same process as EDisc, except that the concepts are continuous rather than discrete;

Table 4.2: Hyperparameter intervals and default values

Hyperparameter	Interval/Set	Default
Hidden Dimension	$\{16, 32, 64, 128, 256, 512\}$	128
Learning Rate	(1e-6, 5e-3), log uniform	1e-3
Batch Size	$\{2, 4, 8, 16, 32\}$	32
Weight Decay	(1e-5, 1e-3), log uniform	1e-4
Hidden Layers	$\{2, 3, 4, 5, 6\}$	4
Concept Regularisation Term	(1e-5, 10), log uniform	1
Concept Pretraining Epochs (PTC)	(10, 300), log uniform	10
No. of Contrastive Examples (ATC)	$2^i, i \in \{4, 5, \ldots, 13\}$	1024

- **ContMTL** (Continuous concepts, Contrastive + Multitask Learning) merges ATC with ECont: for each training epoch, there is a stage with triplet contrastive learning of discretized concepts and a stage with multitask learning with the end task and the continuous concepts.

To evaluate the improvement associated with the application of a method, a control experiment where no concepts are used was also tested.

4.1.5 Model Training and Testing

After the data preprocessing stage, MLP models were trained according to the various methods described in the previous section. Before the data was fed to these models, the continuous features and concepts underwent a standard normalisation according to the corresponding training set distribution. All models were trained for 150 epochs with an early stopping after no validation loss improvement for 30 epochs. Bayesian optimisation of hyperparameters was applied for each method. The hyperparameter ranges can be consulted in Table 4.2.

This experimental process was repeated for 30 times in every method and for every dataset with different train-validation splits so that the average performance end task metrics could be calculated. The main performance metric considered in this experiment was the classification AUROC, calculated in the same test set over all repetitions.

In this part of the experimental process, the sources of knowledge were the two LLMs mentioned in subsection 4.1.1. For a better assessment of the importance of the knowledge embedded in the LLMs, the methods were also tested with random concepts: in each repetition, 5 concepts would be randomly obtained – for each concept, 2 randomly selected features would be multiplied or divided and their discretized versions would be obtained using the methods described in subsection 4.1.3. The results of these experiments can be consulted in Section 5.1.

4.2 Concept Suggestion Improvement

After the results of the first experiment, where it was concluded that LLMs could be a suitable source of knowledge to improve the generalisation capability of MLPs, the possibility of further improving concept suggestions was pondered. This improvement could occur due to an iterative process, where an LLM received feedback on the concepts it provided, or due to a finetuning process, where concept quality is represented in a preference dataset. The experimental setup for the iterative process is shown in Subsection 4.2.1 and the finetuning process is described in Subsection 4.2.2.

4.2.1 Iteration

To further improve concept quality, we developed a setup that allowed to iteratively improve concept suggestions by providing feedback to the LLM. This involved determining the individual quality of each of the concepts provided by the LLM, as well as providing to obtain new concepts; and repeating this process a number of times. The list of used datasets is smaller in this experimental setup, since MLPs already had a high performance in some of the initial datasets (Apples, Glass, BreastWisconsin, and TelephoneChurn).

Concept Quality Determination In order to determine the quality of a concept, it was necessary to discern the contribution of each concept within the set of concepts produced by an LLM. To achieve this, the ContMTL method was run on each of the subsets of that set, and the average difference between the inclusion and the exclusion of a concept was determined across all the subsets.

Iterative Process After determining individual concept impact, a prompt was compiled to provide the LLM with feedback for further iteration. It started by providing the initial prompt and the LLM's response. Then, it listed the used concepts and the respective formula. Following this, there was a list of the impact of each concept and set of concepts with maximum and minimum performance, as well as the performance resulting from using no concepts (with the Control experiment) and with all the concepts. The prompt ended with a request for concept correction.

Following concept extraction, concept quality was determined for each new set of concepts and another iteration with the LLM would take place. In these subsequent iterations, the information in the prompt only included concepts from the preceding step, i.e. the concepts from iterations before the previous one were not mentioned. However, as the web interface for GPT-4 was being used, these concepts were in the communication history and were also considered for LLM inference. A prompt example and respective LLM answer is provided in Appendix A

To determine if this process increased concept quality, the performance of the model trained with ContMTL using all concepts on a held-out test set was determined. The evolution of this performance throughout the iterations with GPT-4 was then plotted. Results are reported in Section 5.2.1.

4.2.2 Finetuning

Given the limited results of the previous experiment, where an iterative process was meant to increase concept quality, LLM finetuning was considered as an alternative path to explore. In order to conduct it, various aspects would have to be decided:

1. What LLM should be chosen so that the computational resources and time available were enough to produce results?
2. Given the virtually infinite space formed by nonlinear combinations of features, what concepts should be considered for the finetuning process, allowing for a suitable tradeoff between coverage and quantity?
3. How to determine concept quality individually for the large list of concepts to consider, knowing that each method is run with 5 concepts at a time?

4. How to produce a finetuning dataset that steers the LLM towards reasonable concept suggestions?

Regarding the choice of LLMs, it was decided that *LLaMa3-8B-Instruct* from Meta AI would be the most suitable one, due to its high score in the LmSysLLM arena (Table 3.1). Furthermore, given that it only has 8 billion (8×10^9) parameters, each taking 2 bytes of storage, the finetuning process could occur in a GPU with 16GB of memory, which was not impeditive. Finally, this LLM is already instruction-finetuned, so the only aspect to improve is concept suggestion.

Concept Enumeration and Quality Determination

Concept inclusion and enumeration was another important issue to tackle. The list of concepts to include in the finetuning process should be broad enough to capture various nonlinear relationships among features. Because of this, it was decided that the list of concepts would include all ratios among two different features, all products between two different features and all combinations of ratios and products between three different features:

$$\begin{aligned} C_{list} = \{A\ op\ B, A, B \in F, A \neq B, op \in \{\times, /\}\} \cup \\ \{A\ op_1\ B\ op_2\ C,\ A, B, C \in F, \\ A, B, C \text{ pairwise different}, op_1, op_2 \in \{\times, /\}\} \end{aligned} \tag{4.2}$$

The sets produced in equation 4.2 had a large number of elements (hundreds of concepts) and it was necessary to determine the ones that produced the most significant improvements in MLP performance, as well as the ones that led to performance degradation. As such, a search algorithm was implemented for this purpose.

Concept Search Algorithm Given the hundreds of concepts whose impact on MLP performance was to be determined (with a greater focus on the better concepts), a concept search algorithm was implemented and is described here.

The algorithm starts by generating the various concept combinations. The set is

Algorithm 2 Concept Search Algorithm

Input: r, C_{list}
Output: List of concepts and respective quality(Q)

$no\_run \leftarrow 0$
$S \leftarrow (s_i := 1 : i \in \{1, ..., |C_{list}|\})$ ▷ Score initialisation phase
$Q \leftarrow ((q_{i,0} := 0, q_{i,1} = 0) : i \in \{1, ..., |C_{list}|\})$ ▷ Initialise concept quality
while $no\_run < r$ **do**
  $C_s \leftarrow sample(C_{list}, 5, S)$ ▷ Sampling probabilities proportional to values in S
  $R \leftarrow []$ ▷ Initialise an empty array to store results
  for all $A \subseteq \mathcal{P}(S)$ **do**
    $auroc_A \leftarrow auroc(ContMTL, A)$
    $R \leftarrow [...R, (r_{A,0} = A, r_{A,1} = auroc_A)]$ ▷ Compute and save value
  end for
  for all $c \in C_s$ **do**
    $Inc_c \leftarrow \{r \in R : c \in r_0\}$ ▷ Get the sets that include c
    $Exc_c \leftarrow \{r \in R : c \notin r_0\}$ ▷ Get the sets that exclude c
    $q_{temp} \leftarrow \frac{1}{|Inc_c|} \sum_{r \in Inc_c} r_1 - \frac{1}{|Exc_c|} \sum_{r \in Exc_c} r_1$ ▷ Compute the quality of c
    $q_c \leftarrow (q_{c,0} + q_{temp}, q_{c,1} + 1)$ ▷ Store it in the total array
  end for
  $Q_{tested} \leftarrow \{q_i \in Q : q_{i,1} > 0\}$
  $avg\_improvement \leftarrow \frac{1}{|Q_{tested}|} \sum_{q_i \in Q_{tested}} q_{i,0}$
  for all $q_i \in Q_{tested}$ **do**
    $N \leftarrow \{n_i \in C_{list} : n \text{ is a neighbor of } q\}$
    if $q_{i,0} > 0$ and $q_{i,0} > avg\_improvement$ **then**
      $s_i \leftarrow s_i * 1.1$; $sgn \leftarrow 1$
    else
      $s_i \leftarrow s_i * 0.9$; $sgn \leftarrow -1$
    end if
    for all $n_i \in N$ **do**
      $s_i \leftarrow s_i * (1 + 0.05 * sgn)$ ▷ Multiply score of neighbor by 1.05 or 0.95
    end for
  end for
  $no\_run \leftarrow no\_run + 1$
end while

then pruned to ensure that every concept considered has two or more distinct values. There is also a score initialisation phase: every concept starts with a score of 1. The score of a concept s_i reflects the probability of choosing this concept in a given run.

For r runs ($r = 10000$, for example), 3 concepts are drawn with probability proportional to their scores. This represents the run's set. For every subset of this set, the ContMTL method is run using its concepts (except for the empty set, where the control method is used). Then, the performance results are used to determine the quality of each concept, given by the average difference between its inclusion and exclusion throughout the subsets. If a given concept's quality is greater than 0 and exceeds the average performance improvement, its score increases 10%; otherwise, it decreases by this amount. Given the large number of concepts in the list, a neighborhood structure was established for the algorithm, allowing for the inference of a concept's prospective quality from its neighbors.

The neighbor structure for this algorithm is the following: for concepts that were calculated using 3 features, the neighborhood would consist of the remaining combinations of operators (op_1, op_2); as well as the combinations of 2 features that would result from excluding a feature and an operator from the initial formula (Equation 4.3). For concepts calculated using only 2 features, the concept resulting from an operator swap would be included, as well as all the 3-feature concepts whose formula includes the operation in the concept (Equation 4.4).

$$\begin{aligned} N(A\ op_1\ B\ op_2\ C) = \{&A\ op_1\ B, A\ op_2\ C, B\ op_2\ C, A\ \overline{op_1}\ B\ op_2\ C, A\ op_1\ B\ \overline{op_2}\ C, \\ &A\ \overline{op_1}\ B\ \overline{op_2}\ C\} \end{aligned} \tag{4.3}$$

$$\begin{aligned} N(A\ op_1\ B) = &\{A\ \overline{op_1}\ B\} \cup \{A\ op_1\ B\ op\ F, op \in \{\times, /\}, F \in \mathcal{F} \setminus \{A, B\}\} \cup \\ &\{A\ op\ F\ op_1\ B, op \in \{\times, /\}, F \in \mathcal{F} \setminus \{A, B\}\} \cup \\ &\{F\ op\ A\ op_1\ B, op \in \{\times, /\}, F \in \mathcal{F} \setminus \{A, B\}\} \end{aligned} \tag{4.4}$$

If a given concept's score is increased in a run, then its neighbors have their score increased by 5%; and if there is a reduction the neighbors also suffer a reduction of the same magnitude.

After all the runs, a list of concepts and associated quality is produced. There can be concepts that were tested more than once, in which case the average improvement is considered. Only the concepts that were explored in the algorithm are included. Because of the algorithm's design, and assuming that neighboring concepts have similar quality, bad-quality concepts are not as thoroughly explored as good-quality ones, and good-quality concepts have their quality better determined by multiple runs.

This list is then used as a reference to create the prompts and responses that shall be fed to the LLM for finetuning. This stage shall be described in the next subsection.

Concept Suggestion Finetuning

After the obtention of a list with concept qualities, the finetuning dataset can be prepared. As can be seen in subsection 2.2.3, this process involves a supervised fine-tuning stage along with a preference optimisation stage, which can happen serially or concurrently. Considering the various choices depicted in that subsection, it was decided that ORPO would be the method of choice, given its simplicity and performance.

In order to apply ORPO, it is necessary to use a preference dataset with 3 entries for each instance: a prompt, a chosen answer and a rejected answer. Following the numbers of the finetuning datasets usually found in Huggingface[15], it was understood that the dataset should contain at least a few tens of thousands of instances, where the preference between 2 suggested lists of concepts is made clear.

This is feasible with the mapping between concepts and respective rise in performance. Even though the concept list contains less than 1000 entries, the number of possible pairwise comparisons of subsets of 5 elements is much larger than the intended number of finetuning instances. With this in mind, only the prompt and response formats have to be handled, as well as the comparison function used to choose between two responses.

[15] https://huggingface.co/datasets

Prompt Formatting

A prompt template was developed for use throughout the entire dataset. It can be divided into four parts, including a contextualisation of the task, a description of the features in a tabular dataset, and the instructions to follow, with respective examples and restrictions. The prompt shown in subsection 4.1.2 broadly follows this template, but the feature description is more elaborate in this case: each feature is accompanied by its type (continuous, binary or categorical), as well as its physical meaning. Statistical metafeatures calculated from the training set are also included in the prompt: for continuous features, the mean, standard deviation, median, minimum and maximum values, kurtosis, skewness and correlation with the target are specified. For binary and continuous features, the quantities and proportions of the various categories. The information gain of these features with respect to the target variable is also specified, as well as their average correlation with the target.

Response Formatting

Given a prompt and a list of concepts, it is necessary to elaborate a response to complete the creation of an instance. For this, it is necessary to name each concept by its formula, and justify its relevance. Both the name of the concept and the justification had to be drawn exclusively from the information present in the prompt. Thus, a naming scheme was developed and a concept justification algorithm was implemented. For three features A, B and C, the naming scheme was the following, assuming that a product can be named as an interaction and that a quotient is a ratio or a source of normalization:

- $A \times B$: A-B Interaction
- $\frac{A}{B}$: A-B Ratio
- $A \times B \times C$: A-B-C Interaction
- $A \times B/C = \frac{A \times B}{C}$: C-Normalized A-B Interaction
- $A/B/C = \frac{A}{B \times C}$: C-Normalized A-B Ratio
- $A/B * C = \frac{A \times C}{B}$: B-Normalized A-C Interaction

The justification of the relevance of a concept was made with resort to an analysis of the correlation signs: considering the concept formula as a fraction, the terms in the numerator should all have the same sign and the terms in the denominator, if present, should all have the opposite sign. In the case that the formula did not meet these restrictions, a generic justification was provided (*could be an important concept*).

Dataset Assembly and Response Comparison

The dataset was assembled by the following procedure:

- From the ordered list of concepts that had their quality determined by the concept search algorithm, the upper third was chosen for the preferred responses and the bottom third was chosen for the rejected ones;
- Three-fourths of the dataset were obtained by sampling five concepts from the upper third for the chosen response and five concepts from the bottom third;
- One fourth of the dataset was obtained by obtaining two 5-concept samples from the upper third. To determine which set was better, a comparison function was implemented, magnifying concept quality by feature diversity, according to Equation 4.5.

$$q_{list} = (\sum_{c \in list} q_c) \times (1 + \frac{no\_different\_features_{list}}{3}) \tag{4.5}$$

The division in thirds aimed to ensure that preferred responses contain qualified concepts, and the proportions for the generation of the dataset aim to promote a distinction between concepts that produce improvements and concepts that do not; while also fostering feature diversity in the lists of concepts given as response.

Given this procedure and a number of dataset instances, the finetuning dataset considering the features of one tabular dataset was created. After the Diabetes and Wine Quality datasets were processed in this way, yielding a 20000-instance dataset with 10000 instances for each tabular dataset.

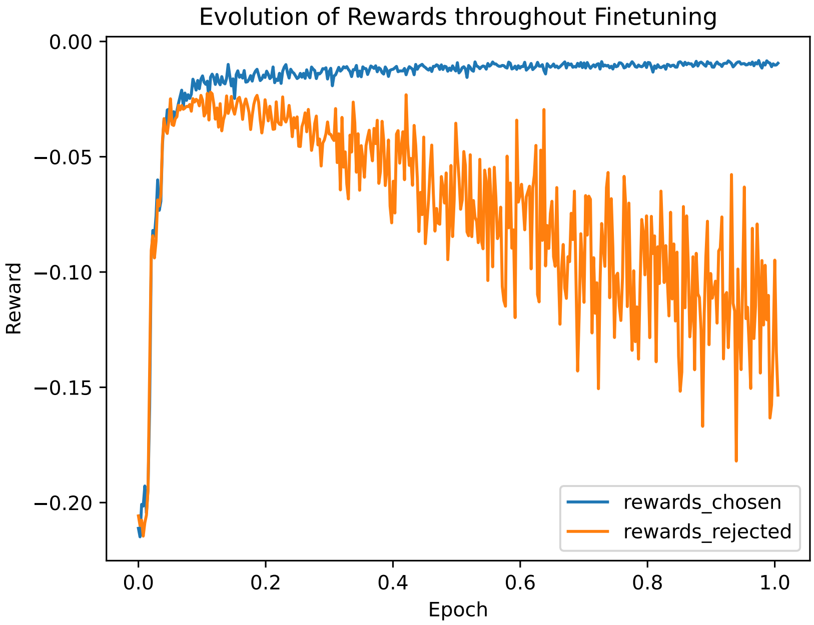


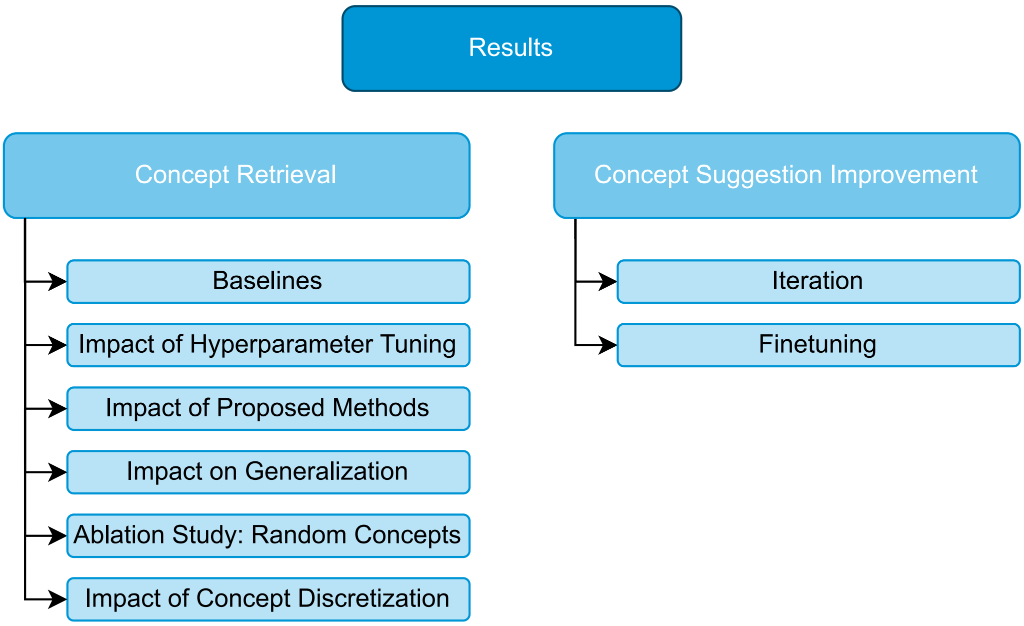
Figure 4.2: Evolution of Rewards throughout Finetuning

Finetuning Process and Parameters

To obtain a finetuned model from the dataset created in the previous stage, the *LLama3-8B-Instruct* model was obtained and quantized to 4 bits using the NF4 data type. A LoRA layer was added to it to allow for PEFT. Following this, the prompts were formatted according to this LLM's dialogue conventions. The dataset was divided into training and validation segments in a 90/10 fashion. Considering that the LLM was already instruction-finetuned and that a preliminary literature review showed that ORPO required few training epochs, the training process only consisted of one epoch with a learning rate of 8×10^{-6} and a β value (the coefficient of the Odds-Ratio loss) of 0.1. Graphs of the training process were obtained, showing that the chosen responses became progressively more likely generated than the rejected ones (Figure 4.2).

Chapter 5

Results



The experimental setups described in chapter 4 produced results that shall be analysed in this chapter. Firstly, we will look at the performance improvements produced by the methods described in section 5.1, assessing the respective generalisation capability. After this, in section 5.2 we will look at how an LLM can be trained to suggest the right concepts that lead to an increase in performance.

5.1 Concept Retrieval Results

We conducted various experiments to understand the impact of the proposed methods on the performance of ML models. Given the aim of the present work,

Table 5.1: Baseline AUROC for all datasets. Each method is tested only on the dataset's features and also using the LLM-obtained concepts as extra features.

Method	LGBM				XGB				MLP			
LLM	-	GPT4	GPT3.5	L3	-	GPT4	GPT3.5	L3	-	GPT4	GPT3.5	L3
Appl	.951	.951	.952	.947	.947	.948	.943	.947	**.977**	.966	.973	.964
Brst	.641	.623	.617	.617	.558	.549	.595	.558	**.712**	.681	.696	.700
BrWsc	.995	.991	.995	.994	.995	.994	.995	.995	.993	**.998**	.996	.997
Cerv	**.679**	.673	.615	.584	.613	.578	.603	.613	.503	.487	.466	.588
Diab	.849	.853	.844	.854	.836	.811	.797	.836	.851	.842	.851	**.857**
Glass	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	.795	.787	.784	.928
Heart	.928	.927	.928	.926	.905	.914	.898	.905	.918	.928	.917	**.935**
HCV	.984	.982	**.987**	.980	.970	.982	.973	.970	.397	.333	.393	.938
TelCh	.992	.992	**.994**	.994	.989	.988	.989	.989	.985	.987	.976	.990
Thor	.579	**.654**	.645	.594	.608	.600	.582	.608	.396	.631	.496	.554
Urin	.749	.744	.748	.783	.728	.759	**.792**	.728	.327	.769	.376	.638
WQred	.906	**.908**	.900	.898	.832	.839	.846	.832	.864	.763	.596	.686
Average	.854	**.858**	.852	.848	.832	.830	.834	.832	.727	.764	.710	.815

Table 5.2: Baseline AUROC for the datasets with age-based split

Method				LGBM				XGB				MLP
LLM	-	GPT4	GPT3.5	L3	-	GPT4	GPT3.5	L3	-	GPT4	GPT3.5	L3
Brst	.578	.676	.683	.609	.605	.593	.543	.605	.743	.623	**.755**	.674
Cerv	.602	.503	.664	.603	**.752**	.622	.661	.752	.535	.315	.293	.321
Diab	.682	.704	.687	.730	.722	.688	.704	.722	**.761**	.673	.758	.683
HCV	.974	.970	.969	.976	.973	.976	.967	.973	.699	.935	.970	**.975**
Heart	.886	.870	.878	.865	.858	.872	.852	.858	.898	.904	**.905**	.886
Urin	.828	.808	.805	.806	**.850**	.824	.814	.850	.346	.743	.373	.304
Thor	.589	.478	.530	.536	**.640**	.571	.466	.640	.339	.357	.340	.342
Average	.734	.715	.745	.732	**.772**	.735	.715	.772	.617	.650	.628	.598

we tested generalisation by using various ratios of training data, and by performing a feature-based split in some datasets. Also, we determined the impact of hyperparameter optimisation and compared it to the effect produced by the methods, having a set of default hyperparameters as control. To better contextualise the results, we explored some performance baselines.

5.1.1 Baselines

We obtained some baselines for comparison with the methods proposed, by determining the AUROC in the test set after training (Tables 5.1 and 5.2).

The performance metrics show that LightGBM and XGBoost are strong baselines with a high average test AUROC (0.83 to 0.85) and that MLPs without concepts perform significantly worse, with AUROC inferior to 0.73. Still, there are datasets where it has a performance in line with these baselines, such as Apples, Breast and Diabetes.

As far as extra features are concerned, providing the strong baselines with extra

Table 5.3: Target AUROC for classification, GPT-3.5 concepts, optimized hyperparameters

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.978	0.803	0.971	0.978	0.975	**0.979**
Breast	0.625	0.612	**0.652**	0.614	0.626	0.630
BreastWisconsin	0.992	0.996	0.996	**0.997**	0.993	0.992
Cervical	0.579	0.575	**0.632**	0.613	0.606	0.590
Diabetes	0.798	0.839	**0.853**	0.846	0.809	0.848
Glass	0.955	0.929	0.885	0.980	**0.986**	0.810
HCV	**0.803**	0.720	0.305	0.783	0.741	0.789
Heart	0.906	**0.933**	0.928	0.929	0.911	0.918
TelephoneChurn	0.986	0.987	0.987	0.987	**0.988**	0.987
Thoracic	0.642	0.665	0.662	**0.669**	0.619	0.597
Urinalysis	0.620	0.730	0.654	0.726	**0.735**	0.722
Winequality	0.527	0.513	0.545	0.562	**0.636**	0.539
Average	0.784	0.775	0.756	**0.807**	0.802	0.783

features does not affect performance significantly: XGBoost does not change at all when equipped with GPT3.5 concepts, and its performance slightly degrades with GPT-4 concepts; whereas LightGBM benefits very slightly from GPT4 concepts. This can be explained by the fact that the provided concepts are nonlinear combinations of dataset features: the nature of these baseline algorithms (ensemble and tree-based) makes them consider relevant nonlinear combinations of features for themselves.

MLPs, on the other hand, seem to benefit from the concepts provided by LLMs: when using GPT-4 concepts as extra features, the MLP architecture has an AUROC 0.04 higher than when using only the default features. The effect is further noticeable when using LLaMa-3 concepts, which make the AUROC lie within 0.02 of the performance achieved by XGBoost.

As far as generalisation with age-based split is concerned, a drop in AUROC is noticeable in all classifiers with respect to the stratified split. However, the ensemble learning methods remain largely unaffected by LLM concepts, unlike MLPs, whose performance rises upward of 0.03 with the use of GPT-4 concepts but remains far from the ensemble baselines.

5.1.2 Impact of the proposed methods

Tables 5.3 and 5.4 show the performance that was obtained by the various proposed methods, using tuned hyperparameters specific for each method and mod-

Table 5.4: Target AUROC for classification, GPT-4 concepts, optimized hyperparameters

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.975	0.969	0.965	**0.981**	0.979	0.977
Breast	0.620	0.663	**0.687**	0.660	0.603	0.670
BreastWisconsin	0.992	0.997	0.996	0.989	0.993	0.997
Cervical	0.605	0.520	0.604	**0.610**	0.580	0.558
Diabetes	0.814	0.808	0.843	0.848	0.844	**0.852**
Glass	0.875	0.903	**0.976**	0.973	0.885	0.734
HCV	0.807	**0.918**	0.705	0.800	0.843	0.265
Heart	0.908	0.911	**0.933**	0.929	0.931	0.926
TelephoneChurn	0.988	0.987	0.987	0.983	**0.989**	0.985
Thoracic	0.575	**0.668**	0.630	0.651	0.537	0.538
Urinalysis	0.624	0.721	0.725	0.665	0.694	**0.772**
Winequality	0.443	0.555	0.554	0.530	**0.595**	0.560
Average	0.769	0.802	0.800	0.802	0.789	0.736

ifying the latent space using GPT-4 and GPT-3.5 concepts.

These tables show that the proposed methods also aid performance in MLPs. This is especially clear for the PTC method, where the average AUROC is superior to 0.80 using both LLMs as sources of concepts. When using GPT-4, this threshold was also attained by ECont and EDisc, meaning that both contrastive learning and multitask learning can produce increases in performance by adapting the MLP's latent space to the concepts suggested by the LLM. When applied together, the result is reduced, as ContMTL showed little to no performance gains when compared to the control experiment in both cases.

Overall, according to these tables, ATC and PTC are the most suitable methods to improve performance with respect to the Control experiment, given that they beat this baseline about 75% of times and display an average performance close to 0.80 using both LLMs as source of knowledge. In fact, these are the only methods where the average performance is superior to Control using GPT-3.5 concepts, which is evidence of their regularisation effect, regardless of which concepts are used.

5.1.3 Impact of Hyperparameter Tuning

In order to assess the impact of hyperparameter tuning, the proposed methods were also run with default hyperparameters, which provided further insight into their behaviour. Tables 5.5 and 5.6 show the performance metrics gathered in this

Table 5.5: Target AUROC for classification, GPT-3.5 concepts, default hyperparameters

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.975	0.974	0.977	0.977	0.977	0.976
Breast	0.630	0.649	0.651	0.638	0.617	**0.654**
BreastWisconsin	0.994	0.993	0.994	0.994	**0.995**	0.994
Cervical	0.594	0.641	0.620	0.599	0.656	**0.671**
Diabetes	0.814	0.840	0.841	0.818	0.827	**0.854**
Glass	0.909	0.635	0.803	**0.939**	0.839	0.818
HCV	0.274	0.799	0.129	0.500	**0.886**	0.703
Heart	0.908	**0.918**	0.909	0.904	0.907	0.913
TelephoneChurn	0.986	0.988	0.987	0.986	0.986	**0.989**
Thoracic	0.547	**0.582**	0.572	0.558	0.550	0.565
Urinalysis	0.639	0.722	0.697	0.653	0.717	**0.738**
Winequality	0.515	0.468	0.518	0.533	**0.595**	0.575
Average	0.732	0.767	0.725	0.758	**0.796**	0.787

Table 5.6: Target AUROC for classification, GPT-4 concepts, default hyperparameters

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.977	0.976	0.975	0.977	0.975	0.975
Breast	0.623	0.644	0.644	0.639	0.663	**0.669**
BreastWisconsin	0.993	0.997	0.996	0.994	0.996	0.997
Cervical	0.597	0.635	0.617	0.594	0.617	**0.618**
Diabetes	0.815	0.838	**0.845**	0.815	0.827	0.843
Glass	0.903	0.668	0.795	**0.961**	0.825	0.820
HCV	0.187	0.681	0.157	0.484	**0.805**	0.667
Heart	0.908	0.914	0.911	0.909	0.909	**0.917**
TelephoneChurn	0.986	0.987	0.986	0.987	0.986	0.986
Thoracic	0.554	**0.604**	0.588	0.557	0.548	0.584
Urinalysis	0.633	0.708	0.702	0.631	0.689	**0.716**
Winequality	0.505	0.522	0.477	0.527	**0.602**	0.524
Average	0.723	0.764	0.724	0.756	**0.787**	0.776

experiment when using the concepts provided by GPT-3.5 and GPT-4, respectively.

When comparing the average performance between the optimized and the default networks, we can conclude that there are methods more sensitive to hyperparameter tuning than others. Control, EDisc and PTC are quite sensitive, since optimisation increases AUROC by about 0.05 in these cases. On the other hand, EDisc, ATC and ContMTL are less affected by this process, so they could be more suitable in cases where hyperparameter tuning is too expensive or time-consuming.

Looking at the results attained by the optimized Control experiment, it also becomes clear that applying the proposed methods with default hyperparameters

Table 5.7: Target classification AUROC, age split, GPT-3.5 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Breast	0.582	0.730	0.732	**0.763**	0.774	0.636
Cervical	0.479	0.600	0.523	0.465	0.612	**0.688**
Diabetes	0.668	0.674	**0.723**	0.688	0.679	0.682
HCV	0.666	**0.880**	0.160	0.732	0.492	0.526
Heart	0.824	0.869	0.887	**0.905**	0.790	0.777
Thoracic	0.453	0.609	**0.615**	0.516	0.434	0.569
Urinalysis	0.621	**0.710**	0.689	0.683	0.687	0.690
Average	0.613	**0.725**	0.618	0.679	0.638	0.653

Table 5.8: Target classification AUROC, age split, GPT-4 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Breast	0.575	0.782	**0.823**	0.524	0.677	0.747
Cervical	0.427	0.585	**0.714**	0.533	0.601	0.527
Diabetes	0.693	**0.756**	0.676	0.703	0.705	0.715
HCV	0.803	**0.845**	0.443	0.607	0.714	0.832
Heart	0.789	0.881	**0.888**	0.825	0.812	0.884
Thoracic	0.454	0.600	**0.706**	0.588	0.530	0.645
Urinalysis	0.639	0.682	0.683	0.633	0.661	**0.685**
Average	0.626	**0.733**	0.705	0.630	0.671	0.719

has a similar effect on performance while dispensing hyperparameter tuning: in both cases, using ATC or ContMTL with default hyperparameters was better than applying the Control experiment with optimized hyperparameters. This makes these methods suitable for use in cases where hyperparameter optimisation cannot be applied, as long as relevant concepts can be extracted.

Overall, we can say that hyperparameter tuning and the proposed methods have effects of similar magnitude (about 0.05 in AUROC for these datasets) and that they partially interact, leading to increases in performance that can surpass 0.07 (from 0.732 to 0.807 in the GPT-3.5 experiment, from 0.723 to 0.802 in the GPT-4 experiment) – hyperparameter optimisation changes the structure of the neural network, whereas the proposed methods alter the structure of the latent space within a network, by adding extra knowledge retrieved from LLMs.

5.1.4 Impact on Generalisation

In order to study the impact that the proposed methods have on generalisation, two main experiments were conducted: firstly, results were obtained using a feature-based train-test split approach: for datasets that included an age feature, the training set comprised the younger 80%, while the test set consisted of the

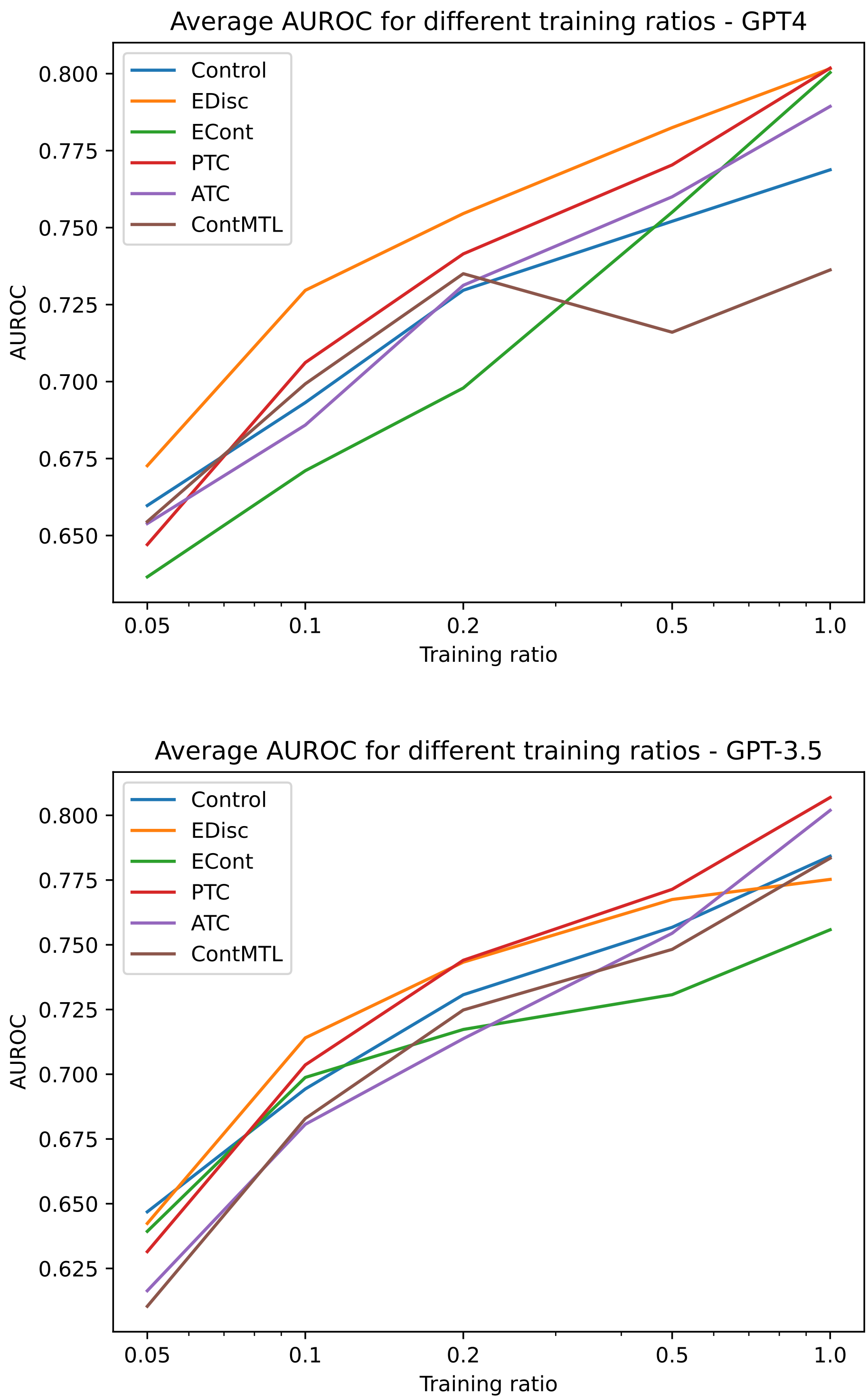


Figure 5.1: Average AUROC by method for various training ratios. GPT-4 results are on top and GPT-3.5 results are on the bottom

older 20%. Additionally, some of the proposed methods were evaluated using only a fraction of the training set: performance was measured using 5%, 10%, 20%, and 50% of the training instances to determine the data efficiency of the proposed methods. This fraction was obtained with stratification.

The results for various training set ratios across the various LLMs are shown in Figure 5.1. This graph shows that the usage of methods such as EDisc or PTC can be beneficial in cases where there is low data availability: applying these methods using only half of the training data yielded a performance that was very similar to the one verified in the control experiment using the whole training set, meaning that the incorporation of these concepts fosters generalisation. This is especially noticeable when using GPT-4 as a source of knowledge, since the hyperparameter optimisation in this stage yielded a less performant MLP.

The results for age split are shown in Tables 5.7 and 5.8. In this case, the improvements are also clear, especially in EDisc, where the AUROC score is within 0.01 of the performance achieved by LightGBM in this setting. In these tables, the superiority of the concepts provided by GPT-4 is also clear across most methods, showing that the quality of an LLM produces an impact on the generalisation capability of the MLP, which can make methods such as EDisc a possible alternative to the ensemble strategies in situations where concept drift is significant.

5.1.5 Ablation Study: Use of Random Concepts

In order to understand whether the knowledge retrieved from LLMs was responsible for the increase in performance across the various methods, we applied random concepts (obtained as ratios or products among two random features of a dataset) to the proposed methods, using the optimized hyperparameters determined for GPT-4. Tables 5.9 and 5.10 show the results obtained in this experiment.

The results show random concepts also seem to have a significant impact on model performance, both when using contrastive learning (PTC and ATC) and when using multitask learning (EDisc and ECont). This means that the proposed methods have a strong regularisation effect by themselves. This effect is much stronger than the one produced by LLM knowledge: on average, the best method is PTC using GPT-3.5 concepts, and this average is only 0.004 higher than the

Table 5.9: Target classification AUROC, random concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.977	0.969	0.968	0.981	**0.979**	0.976
Breast	0.617	**0.666**	0.656	0.644	0.570	0.621
BreastWisconsin	0.991	0.995	**0.996**	0.991	0.991	0.993
Cervical	0.611	0.586	0.581	0.608	**0.621**	0.572
Diabetes	0.809	0.788	0.838	0.841	0.849	**0.856**
Glass	0.867	0.907	**0.977**	0.968	0.900	0.787
HCV	0.834	**0.886**	0.529	0.821	0.827	0.172
Heart	0.912	0.913	**0.928**	0.925	0.925	0.923
TelephoneChurn	**0.989**	0.988	0.986	0.983	0.987	0.985
Thoracic	0.573	**0.667**	0.651	0.662	0.571	0.599
Urinalysis	0.624	0.719	0.726	0.694	0.695	**0.729**
Winequality	0.435	0.532	0.545	0.526	**0.574**	0.551
Average	0.770	0.801	0.782	**0.803**	0.791	0.730

Table 5.10: Target classification AUROC, age split, random concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Breast	0.578	0.789	**0.799**	0.536	0.679	0.666
Cervical	0.428	0.539	**0.579**	0.500	0.525	0.528
Diabetes	0.687	0.680	0.706	0.698	**0.707**	0.688
HCV	0.820	**0.849**	0.354	0.557	0.600	0.703
Heart	0.796	0.878	**0.889**	0.804	0.801	0.885
Thoracic	0.445	0.573	**0.624**	0.571	0.566	0.550
Urinalysis	0.647	**0.719**	0.682	0.633	0.706	0.718
Average	0.629	**0.718**	0.662	0.614	0.655	0.677

method that uses random concepts.

However, LLM knowledge has an important role in generalisation in cases where there is concept drift, as can be seen from the comparison between Tables 5.8 and 5.10: with age split, the AUROC achieved by GPT-4 concepts was higher by about 0.015 when applying the same method.

5.1.6 Impact of Concept Discretization

Given the nature of some of the proposed methods, where the latent space is structured around discretized concepts, it is important to understand if the process of discretisation leads to the loss of relevant information or if it is beneficial. Looking at Tables 5.3 and 5.4 and comparing ECont with EDisc directly, it can be seen that they are in line with each other: across these two tables, EDisc is superior in 10 of the cases; whereas ECont is better in 11 of the cases. However, discretisation possibilitates contrastive learning. A comparison with PTC and ATC against ECont with the same data shows that PTC is superior in 13 of the cases and ATC is better in 10 of the cases. As such, discretisation appears to increase performance in PTC. This could be due to the incorporation of training data information in the process: the entropy-based split separates the instances into positives and negatives according to a threshold determined in the training dataset.

5.2 Concept Suggestion Improvement

The results shown in the previous section led to the conclusion that LLM knowledge had little contribution to the increase in MLP performance, but we conducted further experiments to improve the quality of the concepts retrieved from LLMs. A first attempt consisted of an iterative process where after concept retrieval, we would evaluate the contribution of each concept to provide feedback on the concepts to the LLM in a subsequent iteration. Another alternative was explored: *LLaMa3-8B-Instruct* was finetuned on a concept retrieval dataset and the proposed methods were evaluated using the concepts suggested by this LLM. The results from these experiments are described in the subsections below.

5.2.1 Iteration

To evaluate the impact of an iterative process in concept suggestion, some of the datasets used in the previous steps were selected and the performance of the ContMTL method was evaluated over 5 iterations. This method was chosen since it uses both continuous and discretized concepts and has a good performance without hyperparameter tuning, allowing for a holistic evaluation. The selection process excluded some of the datasets where AUROC was too high for any improvements to be noticed, including Apples, Glass, TelephoneChurn and BreastWisconsin. The results can be consulted in Figure 5.2.

This graph shows that the impact of iteration is mixed. Even though there were 2 datasets where significant gains were attained (HCV and Urinalysis), significant losses were also found in the Breast and Cervical datasets. Still, given the improvements found in HCV and Urinalysis, performing an iterative process can be advantageous.

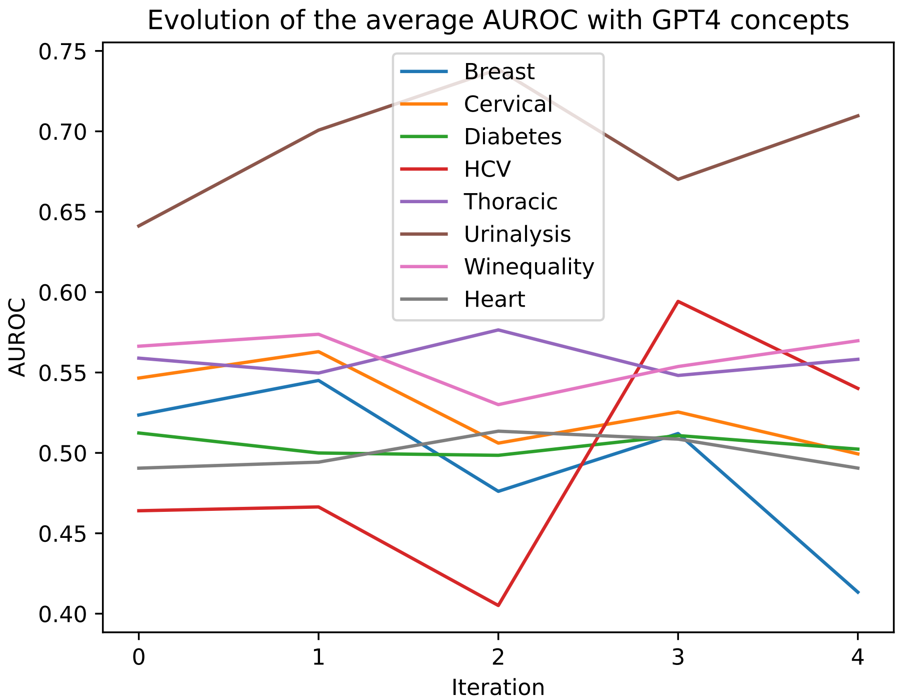


Figure 5.2: Average AUROC by method throughout GPT-4 iterations

Table 5.11: Dataset L2 complexity metrics

	Brst	Cerv	Diab	HCV	Heart	Thor	Urin	WQred
L2	0.276	0.060	0.236	0.019	0.147	0.146	0.057	0.076

In order to understand in which cases there is an improvement, several complexity metrics were extracted with the *pymfe* library and an analysis was conducted. The analysis of those metrics led to the conclusion that problems that are more linearly separable benefit more from this iterative process: the L2 metric calculated by this library, which corresponds to the average error rate of the one-vs-one linear classifiers across all the classes, is lower in HCV and Urinalysis than in the remaining datasets, as can be seen in Table 5.11. Thus, the feedback provided by these concepts can more easily be incorporated to handle corner cases in these datasets than in the remaining ones, where changes can lead to a greater error rate.

5.2.2 Finetuning

Another possible path to improve the quality of concept suggestion is task-specific LLM finetuning. We determined concept quality for the Winequality and Dia-

Table 5.12: Target classification AUROC, default LLaMa3 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	0.977	0.972	**0.981**	0.976	0.974	0.978
Breast	0.602	0.604	**0.669**	0.661	0.637	0.645
BreastWisconsin	0.989	**0.997**	0.993	0.991	0.989	0.994
Cervical	0.572	0.620	0.621	**0.660**	0.605	0.599
Diabetes	0.818	0.801	0.805	**0.842**	0.802	0.820
Glass	0.885	0.842	0.952	**0.984**	0.983	0.941
HCV	**0.836**	0.730	0.348	0.789	0.404	0.564
Heart	0.906	0.922	**0.928**	0.927	0.911	0.924
TelephoneChurn	0.987	0.988	0.987	0.988	0.988	0.985
Thoracic	**0.630**	0.625	0.560	0.586	0.598	0.569
Urinalysis	0.616	**0.777**	0.726	0.736	0.646	0.750
Winequality	0.513	0.483	0.557	0.559	0.552	**0.604**
Average	0.777	0.780	0.761	**0.808**	0.757	0.781

Table 5.13: Target classification AUROC, finetuned LLaMa3 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Apples	**0.978**	0.972	0.976	0.977	0.970	0.968
Breast	0.604	0.612	0.649	**0.666**	0.582	0.572
BreastWisconsin	0.994	0.993	**0.996**	0.988	0.989	0.992
Cervical	0.629	0.620	0.617	**0.634**	0.610	0.613
Diabetes	0.796	**0.856**	0.834	0.815	0.828	0.791
Glass	0.929	0.858	0.945	**0.965**	0.960	0.962
HCV	0.863	0.787	0.446	0.782	**0.904**	0.818
Heart	0.911	0.913	0.903	**0.925**	0.921	0.912
TelephoneChurn	0.988	0.988	0.985	0.986	0.987	0.988
Thoracic	0.605	**0.668**	0.562	0.582	0.659	0.572
Urinalysis	0.622	**0.783**	0.729	0.726	0.739	0.758
Winequality	0.537	**0.599**	0.534	0.508	0.535	0.499
Average	0.788	0.804	0.765	0.796	**0.807**	0.787

betes datasets and created a finetuning dataset based on this information, applying the procedure described in Subsection 4.2.2. After hyperparameter optimization, Table 5.13 was obtained, displaying the performance of the proposed methods across the various datasets.

Tables 5.12 and 5.13 show that a slight improvement in performance across some methods is noticeable (EDisc and ATC), and an interesting result is obtained in the Diabetes dataset: the maximum test performance goes from 0.842 with the PTC method to 0.856 with the EDisc method, showing that the suggested concepts improved in the dataset where the tuning took place. The same did not occur with the Winequality dataset, where test-time performance did not change significantly. However, since the performance in Diabetes exceeds the strong baselines shown in Table 5.1, there is the possibility of using this finetuning pipeline for a given dataset in order to achieve greater performance. Testing this pipeline with

Table 5.14: Target classification AUROC, age split, default LLaMa3 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Breast	0.560	**0.828**	0.784	0.755	0.806	0.778
Cervical	0.467	0.491	0.432	0.500	0.507	**0.577**
Diabetes	0.697	**0.776**	0.751	0.732	0.669	0.753
HCV	**0.806**	0.801	0.250	0.344	0.799	0.734
Heart	0.786	0.779	0.800	**0.896**	0.781	0.785
Thoracic	0.460	0.621	0.617	**0.644**	0.576	0.548
Urinalysis	0.621	0.652	0.688	0.697	**0.738**	0.686
Average	0.628	**0.707**	0.617	0.653	0.697	0.695

Table 5.15: Target classification AUROC, age split, finetuned LLaMa3 concepts

	Control	EDisc	ECont	PTC	ATC	ContMTL
Breast	0.578	0.795	0.798	0.821	0.711	**0.823**
Cervical	0.466	**0.553**	0.449	0.519	0.453	0.496
Diabetes	0.677	0.685	0.716	0.687	0.668	**0.760**
HCV	0.848	0.795	0.719	0.825	**0.887**	0.875
Heart	0.802	**0.885**	0.798	0.797	0.800	0.872
Thoracic	0.474	**0.649**	0.437	0.607	0.567	0.602
Urinalysis	0.642	**0.680**	0.678	0.670	0.626	0.651
Average	0.641	0.720	0.656	0.704	0.673	**0.726**

other datasets will be left for future work.

When applying age split (Tables 5.14 and 5.15), the maximum average classification AUROC increases by about 0.019 after finetuning (from 0.707 to 0.726, above the random baseline presented in Table 5.10), which is evidence of the positive benefit that this process has on generalisation. In the tests with the finetuned LLM, the performance of the ContMTL model increased by 0.03, because of datasets such as Breast, Diabetes, HCV and Heart, which could indicate that the LLM gained general-purpose sensitivity to the formulas it has to retrieve, and that the method used as a base for concept quality retrieval was benefitted.

Chapter 6

Conclusion

Generalisation is an issue inherent to machine learning, given the inevitable limits on model training data. The present work aimed to tackle it by embedding LLM knowledge into tabular classifiers, using formulas derived from dataset features and applying contrastive and multi-task learning techniques to MLPs. We also evaluated the impact of task-specific LLM finetuning in the improvement of model performance. Across 12 classification datasets, the evaluation of the proposed methods against the control experiments and other baselines allows us to conclude the following:

1. In some cases, MLPs can benefit from concepts as extra features to perform in line with ensemble learners such as *XGBoost*;

2. The proposed methods show an important regularisation effect, by leading to a noticeable increase in classifier AUROC in comparison to the control experiment, regardless of hyperparameter optimisation or knowledge source (be it LLMs or random concepts);

3. LLM knowledge does not appear to have a significant impact on model performance in normal training conditions. However, there is an increase in generalisation in cases where an artificial concept drift is present, as shown by the age-based split;

4. The use of entropy-discretized concepts, by allowing contrastive learning, joins the information gathered from LLMs with the one present in the training dataset and leads to a performance that is slightly superior to continuous methods;

5. Iteration has a mixed effect on model performance, as AUROC can increase or decrease in some datasets but mostly remains the same;

6. Task-specific LLM finetuning positively affects the generalisation capability of the models that apply the proposed methods.

6.1 Future Work

With the takeaways that have been produced by the work, further exploration can take place in the following directions:

1. The conclusions can be made more robust by extracting concepts and applying the experimental setup to a greater variety of LLMs;

2. These takeaways can be thoroughly tested with regression datasets, as well as other types of data, such as images;

3. The proposed methods can be tested in other ML model architectures, including neural network ensembles.

Exploring these directions would be a step further in the applicability of LLMs aiming to improve generalisation.

References

AI@Meta. Llama 3 Model Card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore, December 2023. Association for Computational Linguistics.

Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. Ask Me Anything: A simple strategy for prompting language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=bhUPJnS2g0X.

Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, SIGCSE 2023, pages 500–506, 2023.

Ralph Allan Bradley and Milton E. Terry. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew Peters, Ashish Sabharwal, and Yejin Choi. Adversarial Filters of Dataset Biases. In Hal Daumé III and Aarti Singh, editors, *Proceedings of* *the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1078–1088. PMLR, July 2020. URL https://proceedings.mlr.press/v119/bras20a.html.

Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 961–970, 2015.

Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, July 1997. ISSN 1573-0565.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In Guy Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Suvisoft.

Jintai Chen, Kuanlun Liao, Yao Wan, Danny Z. Chen, and Jian Wu. DANets: Deep Abstract Networks for Tabular Data Classification and Regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3930–3938, June 2022.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference, 2024.

Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. *Neural Computation*, 22(12):3207–3220, December 2010. ISSN 0899-7667.

Peter Clark, Isaac Cowhey, Oren Etzioni, and Tushar Khot. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge, 2018. _eprint: 1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, and Mark Chen. Training Verifiers to Solve Math Word Problems, 2021. _eprint: 2110.14168.

P. Cortez, A. Cerdeira, F. Almeida, and T. Matos. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4): 547–553, 1998.

Min-Yuh Day and Chao-Yu Chen. Artificial Intelligence for Automatic Text Summarization. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 478–484, 2018.

Xiang Deng, Huan Sun, Alyssa Lees, and You Wu. TURL: Table Understanding through Representation Learning. *ACM SIGMOD Record*, 51(1):33–40, June 2022.

Robert Detrano, Andras Janosi, Walter Steinbrunn, Matthias Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern H. Guppy, Stella Lee, and Victor Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American Journal of Cardiology*, 64(5):304–310, 1989. ISSN 0002-9149. doi: https://doi.org/10.1016/0002-9149(89)90524-9. URL https://www.sciencedirect.com/science/article/pii/0002914989905249.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023.

Jacob Devlin, Ming-Wei Chang, Lee Kenton, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, 2019.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez Pérez, and Matthieu Cord. Boosting Few-Shot Visual Learning With Self-Supervision. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8058–8067, 2019.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 1725–1731. AAAI Press, 2017. Place: Melbourne, Australia.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017. ISBN 978-1-4919-6229-9.

Adi Haviv, Jonathan Berant, and Amir Globerson. BERTese: Learning to Speak to BERT. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3618–3623, Online, 2021. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, December 1997. doi: 10.1162/neco.1997.9.8.1735.

Georg F. Hoffmann, Andreas Bietenbeck, Ralf Lichtinghagen, and Frank Klawonn. Using machine learning techniques to generate laboratory diagnostic pathways—a case study. *Journal of Laboratory and Precision Medicine*, 2018.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic Preference Optimization without Reference Model, March 2024. URL http://arxiv.org/abs/2403.07691. arXiv:2403.07691 [cs].

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. Publisher: Pergamon.

Edward J. Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022.

A. G. Ivakhnenko and Valentin Grigoryevich Lapa. *Cybernetics and Forecasting Techniques*. American Elsevier Publishing Company, 1967. ISBN 978-0-444-00020-0. Google-Books-ID: rGFgAAAAMAAJ.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Trent Kyono, Yao Zhang, and Mihaela van der Schaar. CASTLE: Regularization via Auxiliary Causal Graph Discovery. In H. Larochelle, M. Ranzato, and R. Hadsell, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1501–1512. Curran Associates, Inc., 2020.

Phuc H. Le-Khac, Graham Healy, and Alan F. Smeaton. Contrastive Representation Learning: A Framework and Review. *IEEE Access*, 8:193907–193934, 2020. Conference Name: IEEE Access.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 552–561, 2012.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-Task Deep Neural Networks for Natural Language Understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, 2019.

Yun Luo, Fang Guo, Zihan Liu, and Yue Zhang. Mere Contrastive Learning for Cross-Domain Sentiment Analysis. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics*, pages 7099–7111, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.620.

Tiago Martins, Joao M. Cunha, Joao Correia, and Penousal Machado. Towards the Evolution of Prompts with MetaPrompter. In Colin Johnson, Nereida Rodriguez-Fernandez, and Sergio M. Rebelo, editors, *Artificial Intelligence in Music, Sound, Art and Design*, pages 180–195, Cham, 2023. Springer Nature Switzerland.

Tomas Mikolov, Ilya Sutskever, Kai Chen, and Greg S Corrado. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

OpenAI. GPT-4 Technical Report, December 2023. arXiv:2303.08774 [cs].

Jun Peng, Yiyi Zhou, Xiaoshuai Sun, and Liujuan Cao. Knowledge-Driven Generative Adversarial Network for Text-to-Image Synthesis. *IEEE Transactions on Multimedia*, 24:4356–4366, 2022. Conference Name: IEEE Transactions on Multimedia.

Silviu Pitis, Michael R. Zhang, Andrew Wang, and Jimmy Ba. Boosted Prompt Ensembles for Large Language Models, April 2023. URL http://arxiv.org/abs/2304.05970. arXiv:2304.05970 [cs].

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *ICML 2023 Workshop The Many Facets of Preference-Based Learning*, 2023.

Stuart J. Russell, Peter Norvig, Ming-wei Chang, Jacob Devlin, Anca Dragan, David Forsyth, Ian Goodfellow, Jitendra Malik, Vikas Mansinghka, Judea Pearl, and Michael J. Wooldridge. *Artificial intelligence: a modern approach*. Pearson series in artificial intelligence. Pearson, Harlow, fourth edition, global edition edition, 2022. ISBN 978-1-292-40113-3.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. Publisher: ACM New York, NY, USA.

Nabeel Seedat, Nicolas Huynh, Boris van Breugel, and Mihaela van der Schaar. Curated LLM: Synergy of LLMs and Data Curation for tabular augmentation in low-data regimes, 2024. URL https://arxiv.org/abs/2312.12112.

Jack W Smith, James E Everhart, WC Dickson, William C Knowler, and Robert Scott Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the annual symposium on computer application in medical care*, page 261. American Medical Informatics Association, 1988.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, and others. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Daniel S. W. Ting, Yong Liu, Philippe Burlina, and Xinxing Xu. AI for medical imaging goes deep. *Nature Medicine*, 24(5):539–540, May 2018.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, and Xavier Martinet. LLaMA: Open and Efficient Foundation Language Models, February 2023. URL http://arxiv.org/abs/2302.13971. arXiv:2302.13971 [cs] version: 1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Chenguang Wang, Xiao Liu, Zui Chen, Haoyun Hong, Jie Tang, and Dawn Song. DeepStruct: Pretraining of Language Models for Structure Prediction. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 803–823, Dublin, Ireland, May 2022. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits

Reasoning in Large Language Models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, July 2019.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate Before Use: Improving Few-shot Performance of Language Models. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12697–12706. PMLR, July 2021.

Max Zhu, Siniša Stanivuk, Andrija Petrovic, Mladen Nikolic, and Pietro Lio. Incorporating LLM Priors into Tabular Learners. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

Matjaz Zwitter and Milan Soklic. Breast Cancer Dataset, 1988. Ljubljana, Yugoslavia.

Appendices

Appendix A

Iterative Prompting

Below is an example of a prompt used in the iterative process subject to research question 2. It refers to iteration number 4 for the Thoracic dataset and it was fed to GPT-4.

```
I prompted:

‘‘‘
I am building a predictive model that estimates the post-operative life
    expectancy in lung cancer patients.
Each patient is characterized by: diagnosis, forced vital capacity, volume
    that has been exhaled at the end of the first second of forced
    expiration, performance status, presence of pain before surgery,
    presence of haemoptysis before surgery, presence of dyspnoea before
    surgery, presence of cough before surgery, presence of weakness before
    surgery, and T in clinical TNM.
What are helpful higher-level concepts to help with this prediction task?
    Concepts should be engineered features derived from the features
    provided, and aim to capture a fundamental component necessary for the
    prediction task, based on your understanding of medicine. Examples of
    concepts include stripes in images or Gleason score grouping in
    prostate cancer mortality prediction. These concepts should not be
    simple linear combinations of features, but should be more complex and
    non-linear, including the product/quotient between two or more features
     and/or the use of other nonlinear functions.
Please suggest 10 concepts and provide explicit formulae to calculate the
    concept from the features specified above. You must provide explicit
```

~~~
formulae, including all coefficients, and must not use features other than those provided. Provide a brief justification for the concepts you provided.
‘‘‘

After an iterative improvement process, your answer was the following, including a list of concepts, and respective importance explanation:

‘‘‘
Given the performance outcomes from the previously implemented concepts for predicting post-operative life expectancy in lung cancer patients, it’s clear that some concepts showed promising results, while others need refining or replacement to optimize the model’s predictive accuracy. Here, I propose an updated list of five engineered features, focusing on those interactions that seemed most relevant and introducing new modifications to enhance their effectiveness:

### 1. **Advanced TNM Respiratory Impact**
\[ \text{Advanced TNM Respiratory Impact} = \left(\text{T in clinical TNM}^2 \times \text{Forced Vital Capacity}\right) \times \left(\text{Volume exhaled at the end of the first second} + 1\right) \]
**Justification:** This version squares the T in clinical TNM to emphasize the influence of tumor size and location on lung function, incorporating both the Forced Vital Capacity and the first-second exhalation volume to better capture the lung’s operational capacity post-surgery.

### 2. **Comprehensive Performance Status Adjustment**
\[ \text{Comprehensive Performance Status Adjustment} = \left(\text{Performance status} + \text{Presence of pain}\right) \times \left(\text{Symptom Severity Score}^2\right) \]
**Justification:** Refining the interaction by adding the presence of pain directly and squaring the Symptom Severity Score to more heavily weigh the burden of multiple severe symptoms, this concept aims to offer a more robust indicator of the patient’s overall health status affecting recovery.

### 3. **Total Symptom Load Factor**
~~~

\[\text{Total Symptom Load Factor} = \left(\text{Presence of pain} + \text{Presence of haemoptysis} + \text{Presence of dyspnoea} + \text{Presence of cough} + \text{Presence of weakness}\right) \times \left(\frac{\text{Performance status}}{\text{T in clinical TNM} + 1}\right) \]
Justification: A new concept that aggregates all pre-operative symptoms and interacts them with the performance status adjusted by the tumor stage. This factor aims to capture the cumulative effect of symptoms and physical fitness on the patient's recovery trajectory.

4. **Dynamic Respiratory Efficiency Measure**
\[\text{Dynamic Respiratory Efficiency Measure} = \left(\text{Forced Vital Capacity} \times \text{Volume exhaled at the end of the first second}\right) / \left(\text{T in clinical TNM} + 1\right) \]
Justification: This measure multiplies the key respiratory function tests to assess the dynamic lung capacity, divided by the tumor stage to normalize for the impact of cancer severity, providing a direct measure of respiratory health relative to tumor burden.

5. **Pain and Performance Interaction Index**
\[\text{Pain and Performance Interaction Index} = \left(\text{Performance status}^2 \times \left(\text{Presence of pain} + 1\right)\right) / \left(\text{Age} + 1\right) \]
Justification: Introducing a new interaction between performance status and the presence of pain, squared to emphasize their impact on recovery prospects, adjusted for age to account for older patients' typically slower recovery and resilience.

By refining these features and focusing on capturing more detailed interactions and the compounded effects of symptoms, tumor staging, and respiratory capacity, these engineered features aim to enhance the predictive model's accuracy. Integrating these into your multi-layer perceptron should provide a better understanding of the factors that influence life expectancy after lung cancer surgery, leading to more effective predictions.
‘‘‘

The concepts were retrieved:

\[\text{Advanced TNM Respiratory Impact} = \left(\text{T in clinical TNM}^2 \times \text{Forced Vital Capacity}\right) \times \left(\text{Volume exhaled at the end of the first second} + 1\right) \]

\[\text{Comprehensive Performance Status Adjustment} = \left(\text{Performance status} + \text{Presence of pain}\right) \times \left(\text{Symptom Severity Score}^2\right) \]

\[\text{Total Symptom Load Factor} = \left(\text{Presence of pain} + \text{Presence of haemoptysis} + \text{Presence of dyspnoea} + \text{Presence of cough} + \text{Presence of weakness}\right) \times \left(\frac{\text{Performance status}}{\text{T in clinical TNM} + 1}\right) \]

\[\text{Dynamic Respiratory Efficiency Measure} = \left(\text{Forced Vital Capacity} \times \text{Volume exhaled at the end of the first second}\right) / \left(\text{T in clinical TNM} + 1\right) \]

\[\text{Pain and Performance Interaction Index} = \left(\text{Performance status}^2 \times \left(\text{Presence of pain} + 1\right)\right) / \left(\text{Age} + 1\right) \]

An evaluation of the quality of these concepts was undertaken. After they were used as extra outputs in a multi-layer perceptron, the respective increase in performance was measured in a validation dataset. The resulting AUROC increment was calculated for each concept, and the results are presented below:

- Advanced TNM Respiratory Impact: 0.017
- Comprehensive Performance Status Adjustment: 0.029
- Total Symptom Load Factor: 0.019
- Dynamic Respiratory Efficiency Measure: -0.007
- Pain and Performance Interaction Index: 0.025

The MLP without concepts had an AUROC of 0.558. Among all the subsets of this set of 5 concepts, the best-performing MLP had an AUROC of 0.629 and used the following concepts as outputs: Comprehensive Performance Status Adjustment, Total Symptom Load Factor, and Pain and Performance

```
Interaction Index. The worst performing MLP had an AUROC of 0.512 and used the concept Dynamic Respiratory Efficiency Measure as an extra output.

Considering these results, present a new list of 5 relevant concepts, as defined in the initial prompt, to further improve the multi-layer perceptron's performance. You can repeat some concepts you already defined if you find them relevant. The ultimate goal is to have a list of 5 concepts that leads to the greatest increase in MLP performance.
```

The resulting answer from GPT-4 was the following:

```
Given the results from the previous evaluation and the identification of the most impactful concepts, I will focus on refining and enhancing these features to better address the predictive modeling of post-operative life expectancy in lung cancer patients. This new set of engineered features will delve deeper into the interaction among clinical symptoms, respiratory function, and tumor characteristics to maximize the prediction accuracy:

### 1. **Advanced TNM Respiratory Impact (Revised)**
\[ \text{Advanced TNM Respiratory Impact} = \left(\text{T in clinical TNM}^3 \times \text{Forced Vital Capacity}\right) \times \log\left(\text{Volume exhaled at the end of the first second} + 1\right) \]
**Justification:** Intensifying the impact of tumor stage by raising it to the third power to highlight its importance, and changing to a logarithmic scale for the volume exhaled to capture non-linearities in lung function deterioration more accurately.

### 2. **Comprehensive Performance Status Adjustment (Enhanced)**
\[ \text{Comprehensive Performance Status Adjustment} = \left(\text{Performance status} + \text{Presence of pain} \times 2\right) \times \left(\text{Total Symptom Load Factor}\right) \]
**Justification:** Increasing the influence of pain in the interaction, recognizing pain's potential as a significant indicator of recovery challenges, and multiplying by the Total Symptom Load Factor to integrate a broader spectrum of symptoms into the patient's overall health assessment.
```

```
### 3. **Total Symptom Load Factor (Refined)**
\[ \text{Total Symptom Load Factor} = \sum(\text{Presence of pain, haemoptysis, dyspnoea, cough, weakness}) \times \sqrt{\frac{\text{Performance status}}{\text{T in clinical TNM} + 1}} \]
**Justification:** Summing all symptoms and applying a square root to the performance status and tumor stage ratio to mitigate extreme values and better reflect the aggregate impact of symptoms and patient resilience on recovery outcomes.

### 4. **Respiratory Function Composite Index**
\[ \text{Respiratory Function Composite Index} = \left(\text{Forced Vital Capacity} + \text{Volume exhaled at the end of the first second}\right) / \left(\text{T in clinical TNM} + 1\right)^2 \]
**Justification:** Summing the main respiratory function metrics and dividing by the squared T stage to assess the lung capacity relative to tumor burden, providing a more nuanced understanding of the patient's physiological capacity to recover.

### 5. **Pain and Performance Interaction Index (Extended)**
\[ \text{Pain and Performance Interaction Index} = \left(\text{Performance status}^2 \times \left(\text{Presence of pain} + 2\right)\right) / \log \left(\text{Age} + 10\right) \]
**Justification:** Adjusting the influence of pain by adding a factor of 2 and dividing by the logarithm of age plus 10 to smooth the impact of aging, focusing on how performance and pain together modulate recovery trajectories in older versus younger patients.

These updated concepts aim to provide a more precise and effective representation of the interactions between clinical symptoms, respiratory function, and tumor characteristics in lung cancer patients post-surgery. By integrating these refined features into your MLP, the predictive accuracy regarding life expectancy should see significant improvement, hopefully resulting in higher AUROC values.
```